

비교기 설계 및 확장에 용이한 인코딩 방법

박안수*, 정태상
중앙대 제어계측학과*, 중앙대학교 전기전자공학부

Useful Encoding Method for Comparator Design and Expansion

ANN-SOO PARK*, Tea-Sang Chung

Dept. of Control & Instrumentation Eng*, School of Electrical and Electronics Eng. Chung-Ang Univ.

Abstract - 본 논문에서는 비교기의 설계 및 확장에 쉽게 이용할 수 있는 2bit 인코딩 방법을 제안한다. 그리고 제안한 인코딩 방법을 이용하여 현재 비교기로 널리 사용하는 74LS85와 새로 설계한 5bit비교기에서의 직/병렬 N-bit로 확장했을 때의 응용방법을 비교한다. 또한 magnitude 비교기를 이용한 디코더 회로를 꾸며 음수 영역까지 확장할 수 있음을 보인다.

1. 서 론

많은 디지털 시스템의 구성에 있어서 연산을 수행 할 때 값을 제어 할 수 있는 비교기는 핵심 구성요소이다. 기본적인 비교기는 n-bit 크기를 갖는 두 피연산(operand)의 입력 값(A,B)을 받아들여 생기는 2ⁿ개의 경우의 수를 디코딩하여 AEQB(A=B),AGTB(A>B),ALTB(A<B)의 세 가지 상태로 해석해 낸다. 비교범위의 수가 넓어지면 n-bit 비교기를 하나의 셀(cell)로 구성하여 여러 개를 서로 연결(cascading)하거나, 확장(expansion)하여 사용한다. 이렇게 셀단위로 grouping하여 비교하려면 결과 값을 상단에 전달하여야 한다. 74LS85 4-bit 비교기는 이러한 확장을 목적으로하는 부가적인 입력단을 포함하고 있는 상용 칩의 대표적인 예이다.

본 논문에서는 74LS85의 효율적으로 확장할 수 있는 새로운 인코딩 방법을 제시하고, 이와 동시에 이 인코딩 방식에 적합한 새로운 5-bit 비교기 셀을 설계하여 직/병렬 확장의 예를 보임으로써 제시한 인코딩 방법의 유용성을 확인한다. 새로운 인코딩방법을 이용하면 보다 쉽게 확장할 수 있고, 음수영역까지 간단한 디코더 회로만으로 비교할 수 있음을 보인다.

다음 2.1-2.2절에서는 비교기의 일반적인 내용과 본 논문에서 대상으로 한 74LS85에 대한 분석과 이의 기본적인 직/병렬 연결 방법에 대해 기술한다. 또한 2.3-2.4절에서는 본 논문에서 제안한 인코딩 방법을 기술하고 이를 이용한 5-bit 비교기의 설계 내용과 직/병렬 연결 구조를 제시하고 3절에 결론을 기술하였다.

2. 본 론

2.1 일반적인 비교기 설계

먼저 일반적인 비교기의 설계를 살펴보면 피연산자 A,B를 단일(single) 비트 이진수라고 가정하고, 구성한 비교기 구현 진리표는 다음과 같다.

표 1. Truth table for an one-bit comparator

A	B	AEQB	A>B	A<B
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

각각의 출력을 식으로 표현하면 다음과 같다.

$$\begin{aligned} AEQB &= \overline{A}\overline{B} + AB = \overline{A} \oplus B \\ A>B &= \overline{A}\overline{B} \\ A<B &= AB \end{aligned} \quad (1)$$

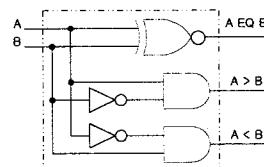


그림 1. Logic for a single-bit comparator

위와 같은 설계를 할 때는 $A_i, \overline{A}_i, B_i, \overline{B}_i, \overline{A}_i \oplus B_i$ 신호가 독립적으로 필요하게 되어 피연산자의 크기가 커질수록 비교기의 내부 로직은 복잡해진다.

2.2 74LS85에 대한 분석

2.2.1 74LS85에서 사용한 기술들

74LS85(이하 '85)는 4-bit magnitude 이진 비교기이다. 이 칩은 좀 더 효과적인 병렬 확장성을 고려하여 설계되었다.

'85에서 특별히 고려한 테크닉은 식(2)과 같다.

$$\begin{aligned} (A_i \equiv B_i) &= \overline{A_i} \overline{B_i} + A_i B_i \\ &= \overline{A_i} B_i + A_i \overline{B_i} = \overline{A_i} \oplus B_i = A_i \odot B_i \quad (2) \\ (A_i > B_i) &= A_i B_i = A_i \overline{A_i} \overline{B_i} \\ (A_i < B_i) &= \overline{A_i} B_i = \overline{A_i} B_i B_i \end{aligned}$$

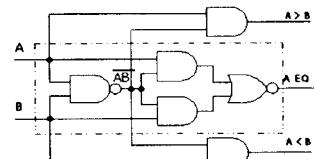


그림 2. Modified a single-bit comparator

식(2)은 내부를 설계할 때 $A_i, B_i, \overline{A}_i \oplus B_i$ 만 필요로 하는데 $\overline{A}_i B_i$ 신호를 $\overline{A}_i \oplus B_i$ 내부에서 공통으로 가져다 쓸 수 있게 하여 내부구조를 간단히 하였다.

표 2를 보면 직렬 연결 입력 $A > B$, $A < B$, $A = B$ 은 least significant position 으로 사용한다. 윗 부분 세 출력들은 mutually exclusive하다. 일반적인 비교기에 아랫부분의 출력을 첨가해 놓았는데 이것은 병렬 확장 scheme에 이용하려는 feed-forward conditions을 반영한다.

표 2. Function Table for 74LS85

Comparing Inputs		Cascading Inputs		Outputs		
A_3, B_3	A_2, B_2	A_1, B_1	A_0, B_0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$
$A_3 > B_3$	x	x	x	x	x	1
$A_3 < B_3$	x	x	x	x	x	0
$A_3 = B_3$	$A_2 > B_2$	x	x	x	x	1
$A_3 = B_3$	$A_2 < B_2$	x	x	x	x	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	x	x	x	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	x	x	x	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	x	x	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	x	x	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	1	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	1	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	x	x	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	1	1	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	0	1

Feed-forward conditions을 고려하여 세운 74LS85의 출력 식을 표현하면 다음과 같다.

$$AGTB_{out} = (A > B) = \overline{A \leq B} = \overline{X}$$

$$X = A \leq B = a_1 + b_1 + c_1 + d_1 + e_1 + f_1$$

$$a_1 = (A_3 \equiv B_3) = \overline{A_3}B_3 = \overline{A_3}B_3B_3$$

$$b_1 = (A_3 \equiv B_3)(A_2 < B_2) = (A_3 \equiv B_3)\overline{A_2}B_2B_2$$

$$c_1 = (A_3 \equiv B_3)(A_2 \equiv B_2)\overline{A_1}B_1B_1 \quad (3-a)$$

$$d_1 = (A_3 \equiv B_3)(A_2 \equiv B_2)(A_1 \equiv B_1)\overline{A_0}B_0B_0$$

$$e_1 = (A_3 \equiv B_3)(A_2 \equiv B_2)(A_1 \equiv B_1)(A_0 \equiv B_0)ALTB_{in}$$

$$f_1 = (A_3 \equiv B_3)(A_2 \equiv B_2)(A_1 \equiv B_1)(A_0 \equiv B_0)AGTB_{in}$$

$$ALTB_{out} = (A < B) = \overline{A \geq B} = \overline{Y}$$

$$Y = A \geq B = a_2 + b_2 + c_2 + d_2 + e_2 + f_2$$

$$a_2 = (A_3 > B_3) = A_3\overline{B_3} = A_3\overline{A_3}B_3$$

$$b_2 = (A_3 \equiv B_3)(A_2 > B_2) = (A_3 \equiv B_3)A_2\overline{A_2}B_2$$

$$c_2 = (A_3 \equiv B_3)(A_2 \equiv B_2)A_1\overline{B_1} \quad (3-b)$$

$$d_2 = (A_3 \equiv B_3)(A_2 \equiv B_2)(A_1 \equiv B_1)A_0\overline{A_0}B_0$$

$$e_2 = (A_3 \equiv B_3)(A_2 \equiv B_2)(A_1 \equiv B_1)(A_0 \equiv B_0)AGTB_{in}$$

$$f_2 = (A_3 \equiv B_3)(A_2 \equiv B_2)(A_1 \equiv B_1)(A_0 \equiv B_0)AEQB_{in}$$

$$AEQB_{out} = (A_3 \equiv B_3)(A_2 \equiv B_2) \quad (3-c)$$

$$(A_1 \equiv B_1)(A_0 \equiv B_0)AEQB_{in}$$

2.2.2 74LS85의 직/병렬 연결 활용방법

'85는 expansion 입력들을 least significant input로 사용하고 있다. cascading 연결할 때 least significant words의 세 출력을 다음 윗단의 corresponding 입력들에 연결한다. 직렬 연결에서는 값을 LSB (less significant bit)에서 MSB(more significant bit)로 가며 비교하는 방식이므로 비효율적이고 연결 셀수가 늘어날수록 ripple의 문제점이 커지게 된다.

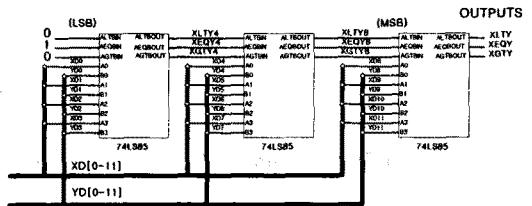


그림 3. 12 bit serial cascading of 74LS85

'85는 ($AEQB_{in}$)=0을 고정시켜놓고 5-bit 비교기로도 사용 할 수 있다. 병렬확장을 할 때에는 expansion 입력들을 least significant 셀을 제외하고 5번째 입력으로 사용한다. 맨 하단은 serial scheme [($A > B$)=0, ($A < B$)=0, ($A = B$)=1] 처럼 연결해야 한다. 만약 $A_4 > B_4$, $A_4 < B_4$ [$A_4B_4 = 10, A_4B_4 = 01$]이라면 expansion 입력의 연결 방법은 맞다. $A_4 = B_4$ ($A_4B_4 = 00, A_4B_4 = 11$)이고, $A_8A_7A_6A_5 \neq B_8B_7B_6B_5$ 이라면 비교결과는 A_4B_4 의 입력 값에 상관없다. 로직은 이런 expansion 입력들을 무시할 수 있도록 설계되어있다. 특별히 고려해야 할 경우는 $A_4 = B_4$. $A_8A_7A_6A_5 = B_8B_7B_6B_5$ 이다. 이 경우에는 $A > B$, $A < B$ 의 expansion 출력들은 00 이거나 11이다. 이 expansion 출력들은 다음 단계 comparing 입력 쌍(pair) 중 하나와 다시 연결하기 때문에 다음 단에도 맞게 해석된 결과 값을 얻을 수 있다. (그림4)

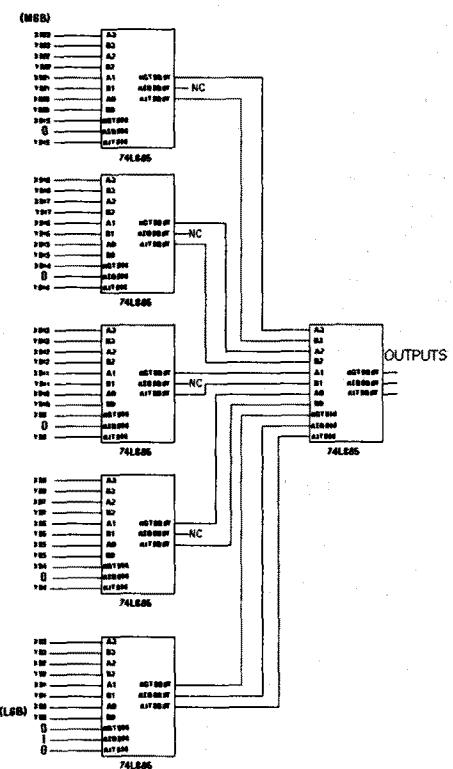


그림 4. 24 bit Parallel expansion of 74LS85

2.3 새로운 2 bit 인코딩방법

2.3.1 제안하는 인코딩방법

위에서 살펴본 것과 같이 74LS85를 expansion하기 위해서는 여러 가지 제한 조건들이 많다. 수의 범위가 커지면 커질수록 설계된 하나의 비교기 셀을 여러 개 연결한다. N-bit 비교기로 사용하려면 각 입력과 출력을 set으로 단일 셀단위 설계 테크닉이 필요하다. 입력과 출력을 primary I/Os(inputs/outputs), secondary I/Os로 구분하고 secondary I/Os는 셀들 사이에 정보를 주고받는데 이용한다. 두 비트를 사용하여 intercell 간에 secondary I/Os를 인코딩하면 three piece of information(AGTB, ALTB, AEQB)을 효과적으로 다음 단으로 전해 줄 수 있다.

본 논문에서 제안하는 인코딩 방법은 표3과 같다.

표 3. New 2-bit encoding method

I/O		Function
$S_{II/O2}$	$S_{II/O1}$	
0	0	$A_i = B_i$
1	0	$A_i > B_i$
0	1	$A_i < B_i$

이 인코딩 방법은 출력으로 나오는 1의 값의 위치에 따라 피연산자의 크기를 알아낼 수 있다. 또한 출력과 입력을 동시에 사용할 수 있다. 그러므로 n-bit비교기를 확장할 때 필요한 하단에서 비교결과를 받아들이는 입력 단이 따로 필요 없다. 하단까지의 비교결과 값을 상단의 입력에 그대로 연결하여 사용하므로 N-bit으로 까지의 확장비교가 쉽다.

표 4. Modified truth table for a single cell of a binary comparator

Present condition	Next condition							
	-Primary inputs A, B							
secondary inputs	00	01	11	10				
$S_2 S_h$	$S_{02} S_{01}$	$S_{02} S_{01}$	$S_{02} S_{01}$	$S_{02} S_{01}$				
$A = B$	0 0	0 0	0 1	0 0	1 0			
$A > B$	1 0	1 0	0 1	1 0	1 0			
$A < B$	0 1	0 1	0 1	0 1	1 0			
Don't care	x x	x x	x x	x x	x x			

위의 방식은 그 내부 구현 회로에 상관없이 $A > B$, $A < B$ 의 출력만으로 셀들 사이를 연결하고 마지막 단에 간단한 AND gate 만으로 디코딩하여 원하는 출력 값을 얻을 수가 있다.

2.3.1 새로운 인코딩방법에 맞춘 5bit 비교기

제안하는 인코딩방법을 고려하여 5-bit 비교기 설계하면 다음과 같다.

$$AGTB_{out} = (A > B)$$

$$A > B = a_1 + b_1 + c_1 + d_1 + e_1$$

$$a_1 = (A_4 \neq B_4) = A_4 \overline{B_4} = A_4 \overline{A_4 B_4}$$

$$b_1 = (A_4 \equiv B_4)(A_3 \neq B_3) = (A_4 \equiv B_4)A_3 \overline{A_3 B_3}$$

$$c_1 = (A_4 \equiv B_4)(A_3 \equiv B_3)A_2 \overline{A_2 B_2} \quad (4-a)$$

$$d_1 = (A_4 \equiv B_4)(A_3 \equiv B_3)(A_2 \equiv B_2)A_1 \overline{A_1 B_1}$$

$$e_1 = (A_4 \equiv B_4)(A_3 \equiv B_3)(A_2 \equiv B_2)(A_1 \equiv B_1)A_0 \overline{A_0 B_0}$$

$$ALTB_{out} = (A \triangleleft B)$$

$$A \triangleleft B = a_2 + b_2 + c_2 + d_2 + e_2$$

$$a_2 = (A_4 \triangleleft B_4) = \overline{A_4 B_4} = \overline{A_4} \overline{B_4} B_4$$

$$b_2 = (A_4 \equiv B_4)(A_3 \triangleleft B_3) = (A_4 \equiv B_4) \overline{A_3 B_3} B_3$$

$$c_2 = (A_4 \equiv B_4)(A_3 \equiv B_3) \overline{A_2 B_2} B_2 \quad (4-b)$$

$$d_2 = (A_4 \equiv B_4)(A_3 \equiv B_3)(A_2 \equiv B_2) \overline{A_1 B_1} B_1$$

$$e_2 = (A_4 \equiv B_4)(A_3 \equiv B_3)(A_2 \equiv B_2)(A_1 \equiv B_1) \overline{A_0 B_0} B_0$$

2.3.2 새로 설계한 5-bit 비교기의 직/병렬 활용 방법

새로 설계된 5-bit 비교기에서는 expansion 입력들이 따로 자리수가 정해있는 것이 아니기 때문에 MSB에서 LSB로 비교할 수 있다. 이것은 실제 우리가 값을 비교하는 방법이다. 그러므로 그림3에 비해 훨씬 더 효과적인 비교를 한다.

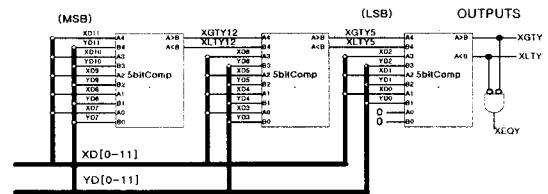


그림 5. 12 bit serial cascading of 5-bit comparator

여러 개의 셀을 연결하여 블록화하고, 여러 개의 블록을 다시 확장시켜 사용하면 원하는 크기의 비교기를 만들 수 있다. 그림4에 비해 N의 크기가 커지면 커질수록 더 효율적이며 블록화 단위들을 연결할 때에도 동일한 방법으로 사용할 수 있다.

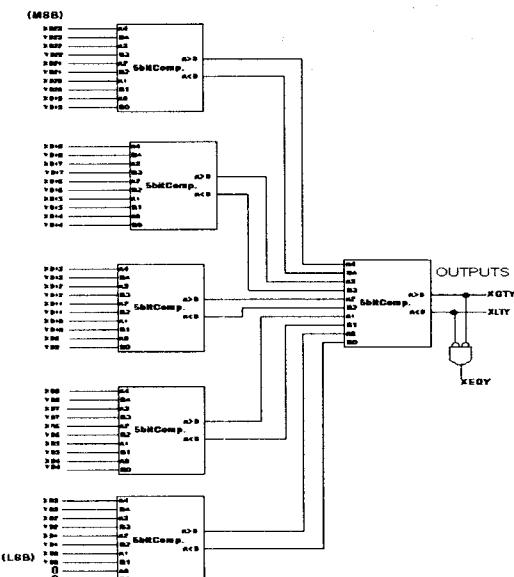


그림 6. 24 bit parallel expansion of 5-bit comparator

2.4 Magnitude 비교기를 이용하여 부호 있는 수 까지의 확장성

때때로 프로그램은 양수나 음수를 다루기를 원하거나 또는 양수만을 다루기를 원한다. 비교기도 이 두 가지를 모두 다룰 수 있어야 한다. N-bit 입력 값에서 가장 최상의 비트(Top bit)를 부호 비트(sign-bit)로 사용하여 부호 있는 수를 표현하는 2의 보수(two's complement)법을 사용한다. 부호 비트 0은 양수를 의미하고 부호 비트 1은 음수를 의미한다. 현재 사용하고 있는 크기(magnitude)비교기를 이용해 부호 있는 수까지 확장할 수 있다.

표 5. Function Table for expanding of signed number system

Top bit	unsigned comp		signed comp			
	A_T	B_T	$A >_u B$	$A <_u B$	$A >, B$	$A <, B$
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	1	0	0
0	0	1	1	x	x	x
0	1	0	0	x	x	x
0	1	0	1	1	0	0
0	1	1	0	x	x	x
0	1	1	1	x	x	x
1	0	0	0	x	x	x
1	0	0	1	x	x	x
1	0	1	0	0	1	0
1	0	1	1	x	x	x
1	1	0	0	0	0	0
1	1	0	1	0	1	1
1	1	1	0	1	0	0
1	1	1	1	x	x	x

표 5를 보면 부호 없는 수의 비교 결과값과 primary 입력값 중 최상의 비트를 입력으로 하고, 부호 있는 수의 비교 결과를 출력으로 한다. $A >, B = 1$, $A <, B = 1$ 인 경우는 제안된 2-bit 인코딩에서 예측되는 값이 아니므로 결과 값을 Don't care 처리한다. 나머지 부분은 수학적으로 예측되지 않는 값들이므로 Don't care 처리한다.

위의 함수표를 이용해 로직 최소화 기법으로 디코더회로를 만들 수 있다.

$$\begin{aligned} A <, B &= \overline{B_T}(A >, B) + A_T(A <, B) + \overline{A_T}(A <, B) \quad (5) \\ A >, B &= \overline{A_T}(A >, B) + B_T(A >, B) + \overline{B_T}(A <, B) \end{aligned}$$

식(5)과 같은 디코딩방법은 크기 비교 뿐 아니라 부호 있는 수의 비교도 할 수 있으므로 정수범위까지 비교범위를 넓힐 수 있다.

3. 결 론

본 논문에서는 새롭게 2bit 인코딩 방법을 제안하고, 현재 널리 사용되고 있는 74LS85와 제안한 인코딩 방법으로 새로 설계한 5-bit 비교기를 비교 분석해 보았다. 제안한 인코딩 방법은 입력 값과 출력 값으로 동시에 쓸 수 있으므로 전단에서 비교 결과 값을 받는 별도의 추가적인 입력 단이 필요 없다. 이 방법은 기존의 74LS85뿐만 아니라 새로운 5-bit 비교기를 N-bit 비교기로 확장시키기 쉽고, 나아가 n-bit 비교기의 설계에도 이용할 수 있다. 또한 크기 비교기를 이용해 부호 있는 수 비교도 쉽게 할 수 있다.

최근 디지털 시스템을 디자인할 때 프로그래밍적으로 처리하는 방법을 많이 사용하지만, 좀 더 빠르고 정확하게 구현하려면, GATE LEVEL 단위의 설계가 필수적이다. 그러기 위해선 두 수를 비교하는 완전한 비교장치를 만들어서 하드웨어적으로 설계해 연산하는 방법이 필요하다. 본 논문에서는 비교기를 설계해 모듈화하여 사용할 수도 있고, 그 확장성 또한 효과적이고 용이하다.

(참 고 문 헌)

- [1] John F. Wakerly, "Digital design : principles and practices". Prentice-Hall International Editions, 215-222, 1990
- [2] "Datasheets of 74LS85", Texas Instruments