1999년도 대한전기학회 정기총회 및 추계학술대회 논문집 1999. 11. 20

74LS49와 74LS49의 디자인에 사용된 로직최소화에 대한 분석

유 준 복*, 정 태 상
중앙대 제어계측학과*, 중앙대 전자전기공학부

Analysis of the Logic Minimization in the Design of 74LS49 and 74LS47 BCD-to-Seven-Segment Decoders

Jun-Bok You*, Tea-Sang Chung
Dept. of Control & Instrumentation Engneering,Chung-Ang Univ.*
School of Electrical and Electronics Engineering.Chung-Ang Univ.

Abstract - The 74LS49 and 74LS47 chips are MSI circuits and are used for decoding the BCD input and driving seven-segment displays. The logic of these chips are often used not only as component chips in the commercial digital systems, but are used as library components in fairly complicated ASIC designs. Thus, the understanding of the logic characteristics of these chips is beneficial for future applications. It was analyzed reversely that the design of these chips includes a special logic minimization technique, which neither documented nor reported. This paper is to analyze the function of the logic and the special minimization technique adapted in the design of 74LS49 and 74LS47 chips.

1. 서 론

대다수의 시스템에서 데이터 멀티플랙싱, 디지털 디스플레이, 디지털-아날로그 변환, 메모리 어드레싱 등의 기능으로 디코딩회로는 반드시 필요하다.[3]

디코더는 n 개의 입력 값을 이용해 2"의 출력 값들 중의 하나를 선택하는 조합논리회로이다. 그 입력 값들과 출력 값들과의 관계는 진리표를 이용해서 나타내어진다. 그리고, 이것은 기능의 확장을 위해 서로 연결되어 사용되어지기도 하고, enable 신호나 chipselect 신호로써 데이터 분배의 기능을 수행하기도 한다. 대부분의 디코딩 기능들은 칩 화되어 상용으로 사용되는 것이 일반적이다.

여러 디코딩기능의 칩들 중에서 우리가 관심을 가지는 것은 BCD-to-seven-segment디코더, 74LS49와 74LS47이다. 특히, 이 칩들의 디자인을 역으로 분석하여, 디코더회로의 내부적인 로직 형태와 이에 사용된 특징적인 로직최소화기법에 대해 알아보려고 한다.

여기서, BCD는 binary-coded decimal의 약자로 4 bit을 가지고 decimal number를 표현하는 방법으로, 0에서 9까지의 수는 0000부터 1001까지의 이진표현으로 표현되어지고, 1010부터 1111 까지는 사용되어지지 않는다. 다시 말하면, 4bit으로 표현 가능한 16 가지의 경우 중에서 단지 10가지의 경우만을 사용한다.

Seven-segment 디코더는 4bit의 BCD값을 입력데이터로 받아서, seven-segment코드를 출력하는 것이다. Seven-segment display는 7개의 개별적인 segment들로 구성되어 있고, 입력 값에 따라 각각의 segment들이 on 또는 off 되어 입력되는 seven-segment코드에 상응하는 수, 0에서 9까지, 를 표시하는 장치이다.[1]

2. BCD-to-Seven Segment display 디코더, 74LS49 와 74LS47.

74LS49와 74LS47은 기본적으로는 동일한 디코딩회로를 가지고 있고 부수적인 기능 면에서 약간의 차이를 가진다. 74LS47은 전류를 접지로 흘려보내 display segment를 직접 구동시키는 open-collect출력 트랜지스터를 가지고 있다. 또한 스트로브, 램프 테스트, 그리고 blanking(전체 segment의 off)등의 기능의 부가적인 입력신호들을 가지고 있다. 반면에 74LS49는 직접구동 트랜지스터를 가지고 있지 않다. 즉, 1의 논리적인 출력을 가지고 display segment를 구동한다. 그리고, 74LS47의 부가적인 기능 중에서 blanking 기능만을 제외한 나머지 기능은 가지고 있지 않다.[2]

2.1 74LS49에 대한 분석

74LS47의 디자인은 74LS49의 디자인을 기초로 한다. 다시 말하면, 74LS47은 74LS49를 기능적으로 확장시켜 좀 더 효과적인 디코딩을 위해 만들어진 것이다. 따라서, 우선 74LS49에 대해 분석한 후에 74LS47의 부가적인 기능들에 대해서 알아보도록 하겠다.

표 1은 74LS49에 대한 진리표이다. 입력 BI (blanking input)를 제외하면, 74LS49의 각각의 출력들은 product-of-sums 형태로 최소화가 된다. 이때, 정의되어지지 않은 BCD 입력에 대해서는 don't care 처리를 했다.[1]

Inputs					Outputs						
BI	D	C	B	A	a	b	c	d	e	f	g
1	×	×	×	×	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1	1	1	1	0
0	0	0	0	1	0	1	1	0	0	0	0
0	0	0	1	0	1	1	0	1	1	0	1
0	0	0	1	1	1	1	1	1	0	0	1
0	0	1	0	0	0	1	1	0	0	1	1
0	0	1	0	1	1	0	1	1	0	1	1
0	0	1	1	0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1	0	0	0	0
0	1	0	0	0	1	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	0	1	1
0	1	0	1	0	0	0	0	1	1	0	1
0	1	0	1	1	0	0	1	1	0	0	1
0	1	1	0	0	0	1	0	0	0	1	1
0	1	1	0	1	1	0	0	1	0	1	1
0	1	1	1	0	0	0	0	1	1	1	1
0	1	1	1	1	0	0	0	0	0	0	0

표 1 74LS49에 대한 진리표.

2.1.1 74LS49의 로직최소화 방법.

Seven-segment decoder의 입력데이터가 유효하지 않은 BCD코드인 경우에는, 출력 값이 결정되어지지

않는다. 그러나, 이 진리표의 경우에는, 사용되어지지 않는 입력의 경우에 대해서도 그 출력 값들이 명확히 정해져 있다. 이것이 74LS49 디자인에서의 중요한 점이라 할 수 있겠다.

먼저, 진리표를 이용해서, Seven-segment 각각에 대해서 Karnaugh map들을 구할 수 있다. 그 중하나, 'a' segment에 대한 것이 그림 1이다.

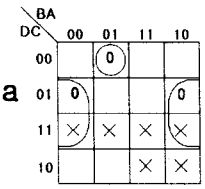

그림 1 'a' segment에 대한 Karnaugh map.

정의되지 않은 출력에 대해서는 ×(don't care)로 표시되었다. 이것을 0를 기준으로 하여,product-of-sums 형태로 최소화하면 다음과 같다.

$$a = (D + C + B + \overline{A}) \cdot (\overline{C} + A).$$

실제로 이것은 74LS49의 'a' segment의 정확한 논리함수가 아니다.

여기서 우리는 두 가지 점에 대해서 좀 더 고려해 봐야 하겠다. 하나는 BI에 관한 것이고, 다른 하나는 don't care의 특별한 이용에 대한 것이다.

BI 입력은 필요에 따라 모든 seven-segment를 off 시키는 기능을 하며, 다른 4개의 BCD입력과는 구분되어 처리되고 있다. 만일 BI가 1이라면, 모든 출력 값은 0이 되고, 그 결과로 seven-segment display에는 어떤 값도 표시되지 않는다. BI가 0이라면, 칩은 정상적으로 동작하여 입력에 의한 수를 표시한다.

따라서, 모든 출력에 대해서 $\overline{BI}$ 가 곱의 형태로 포함된다. 'a' segment의 경우는 다음과 같다.

$$a = (D + C + B + \overline{A}) \cdot (\overline{C} + A) \cdot \overline{BI}$$

이것은 다섯 변수 Karnaugh map을 이용해서 각각의 segment 출력에 대한 논리함수를 구하는 경우와 결과적으로 일치한다. 여기서 다섯 변수는 BCD 입력(A, B, C, D) 과 BI이다.

don't care에 대해 고려해 보면, 진리표로부터 'a' segment에 대한 새로운 Karnaugh map을 구할 수 있다. 이번 경우는 정의되지 않은 출력에 대해서도 진리표의 값을 참고하여 최소화에 적용하였다.

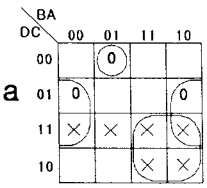

그림 2 'a' segment에 대한 새로운 Karnaugh map.

결과적으로 don't care로 구성된 하나의 새로운 sum 항이 추가되었다. 일반적으로, don't care는 발생되지 않는 입력에 대한 출력이므로 0 혹은 1로 처리되어도 정상적인 회로의 작동에는 영향을 미치지 않는다. 따라서, don't care의 효율적인 사용으로 좀 더 최적화 된 회로를 구현할 수 있다. 하지만, 단지 don't care만으로 이루어진 항을 발생되지 않은 입력에 관한 것이므로, 그 항을 포함하는 것은 로직 디자인 관점에서는 불필요한 회로의 추가를 의미하기 때문에 고려대상에서 제외된다.

그럼에도 불구하고, 그런 형태의 항이 위의 'a' segment의 경우와 'b', 'c' segment 대한 로직최소화 단계에서 포함되었다.

2.1.2 Necessary Redundant

우선, 'a', 'b', 'c' segment의 경우를 제외한 나머지 경우들에 대해 분석해 보겠다. 그림 3은 'd' segment에 대한 Karnaugh map을 나타내고 있다.

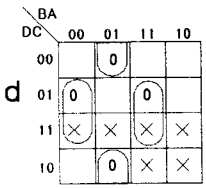

그림 3 'd' segment에 대한 Karnaugh map.

이 Karnaugh map으로부터 얻어진 논리함수는 다음과 같다.

$$\begin{aligned} d &= (\overline{C} + B + A) \cdot (\overline{C} + \overline{B} + \overline{A}) \cdot (C + B + \overline{A}) \cdot \overline{BI} \\ &= (\overline{C} \cdot \overline{BI} + B + A) \cdot (\overline{C} \cdot \overline{BI} + \overline{B} \cdot \overline{BI} + \overline{A} \cdot \overline{BI}) \\ &\qquad \cdot (C + B + \overline{A} \cdot \overline{BI}) \end{aligned}$$

'd' 에 대한 논리함수는 다음의 〔정리 1〕을 적용해서 변환하였다.

〔정리 1〕

$$\begin{cases} x(y+z) = xy + xz \\ \qquad\quad = xy + x \cdot x \cdot z = x(y + xz) \\ \qquad\quad = x \cdot x \cdot y + xz = x(xy + z) \\ \\ x(y+z) \neq xy + z \\ \qquad\quad \neq y + xz \end{cases}$$

〔정리 1〕은 이 칩의 디자인에 있어서 중요한 역할을 하는 한가지이다. 이것을 이용해서 출력에 대한 논리함수의 표현에 사용되는 신호를 두 가지 형태로 간단히 할 수가 있다. 그 신호형태의 하나는 BCD 입력(A, B, C, D)이고, 다른 하나는 BCD입력의 보수와 $\overline{BI}$의 곱형태($\overline{A} \cdot \overline{BI}$, $\overline{B} \cdot \overline{BI}$, $\overline{C} \cdot \overline{BI}$, $\overline{D} \cdot \overline{BI}$)이다. 〔정리 1〕을 이용해서 변환된 'd'의 논리함수를 보면, 두 가지 형태의 신호로 표현됨을 확인할 수 있다.

'e', 'f', 'g'의 경우는 'd'에서와 마찬가지로 간단한 두 가지 형태의 신호들로 표현된다. 이것이 가능한 이유는, 'd', 'e', 'f', 'g' 경우에는 본질적으로 BCD입력의 보수들만으로 구성된 하나의 sum항을 각각 가지고 있기 때문이다. 이 특별한 형태의 sum항이 있어야, 〔정리 1〕을 적용하여 논리함수를 두 종류의 신호형태만으로 표현할 수 있게된다. 'd'의 경우에서는, ($\overline{C} + \overline{B} + \overline{A}$)이 그것이다.

하지만, 'a', 'b', 'c'의 경우는 다르다. 이들의 논리함수에는 이와 같은 형태의 sum항이 없기 때문에, 〔정리 1〕을 적용해서 앞의 경우에서 사용된 것과 같은 입력들 (A, B, C, D, $\overline{A} \cdot \overline{BI}$, $\overline{B} \cdot \overline{BI}$, $\overline{C} \cdot \overline{BI}$, $\overline{D} \cdot \overline{BI}$)만으로 표현하는 것이 불가능하다.

따라서, 'a', 'b', 'c'의 경우에서도 〔정리 1〕의 적용을 위해, 강제적으로 이들 경우의 논리함수에 BCD입력의 보수들만으로 구성되고, 이로 인해서 출력 값이 변화되지 않는 sum항을 추가하여야 한다.

이것이 don't care만으로 구성된 sum항을 포함시키는 이유이다. 이런 sum항을 necessary redundant라 한다.

다시 'a'의 경우를 살펴보면, 그림 2로부터, 논리함수는 necessary redundant를 포함하여 다음과 같이 구해진다.

$$a = (D+C+B+\overline{A})\cdot(\overline{C}+A)\cdot(\overline{D}+\overline{B})\cdot\overline{BI} \quad (\overline{D}+\overline{B}: \text{necessary redundant})$$

$$= (D+C+B+\overline{A}\cdot\overline{BI})\cdot(\overline{C}\cdot\overline{BI}+A)\cdot(\overline{D}\cdot\overline{BI}+\overline{B}\cdot\overline{BI})$$

necessary redundant의 추가로, 'a'의 경우에서도 BCD입력의 보수들만으로 구성된 sum항 $(\overline{D}+\overline{B})$이 존재하고, [정리 1]을 이용해서 위에서 사용된 8가지 종류의 신호만을 가지고 표현하는 것이 가능해 졌다. 'b'와 'c'의 경우에서도 각각 $(\overline{D}+\overline{B})$와 $(\overline{D}+\overline{C})$가 necessary redundant로 논리함수에 추가된다.

결과적으로, $\overline{BI}$ 가 개별적으로 처리되어지지 않고, 반드시 BCD입력의 보수들과의 곱의 형태로만 사용되어지는 것을 의미하게 된다. 전체 출력에 대한 회로를 구성할 때, 먼저 A, B, C, D, $\overline{A}\cdot\overline{BI}$, $\overline{B}\cdot\overline{BI}$, $\overline{C}\cdot\overline{BI}$, $\overline{D}\cdot\overline{BI}$ 신호들만 만들어진다면, 모든 출력들은 이 신호들의 조합으로 구현되어 질 수 있다. [정리 1]과 necessary redundant를 이용해서 4bit BCD 코드와 추가의 BI(blanking input)을 입력을 가지는 seven-segment 디코더, 74LS49의 디자인을 간단하고 효율적으로 한 것이다.

2.2 74LS47에 대한 분석

74LS47은 74LS49를 기본으로 하여 디자인되었다. 이것은 74LS49 디자인에서 사용된 특징적인 방법들을 바탕으로 하여, 몇 가지 기능들을 추가해서 디자인되었다.

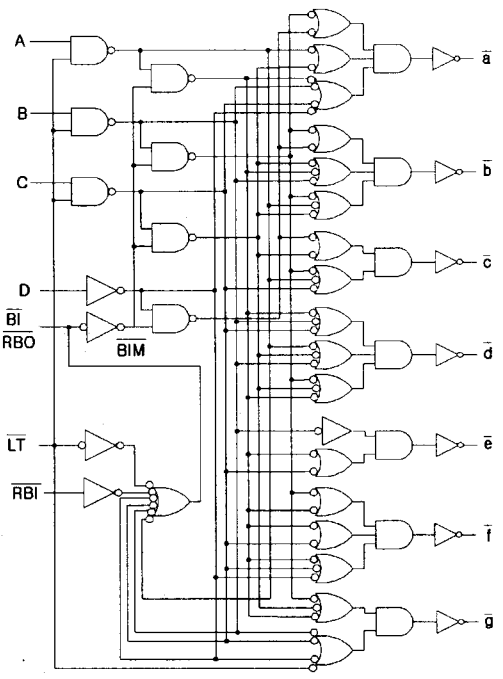

그림 4 74LS47의 로직 다이어그램(logic diagram).

먼저, 74LS47은 앞서는 0(leading zero)과 뒤따르는 0(trailing zero)의 위치의 seven-segment를 모두 off 시키는(blanking) 기능과 seven-segment의 상태를 점검하기 위한 램프 테스트(lamp test)의 기능을 가지고 있다. 이를 위해 74LS49보다 새로운 2개의 입력, RBI(ripple blanking input), LT(lamp test)과 1개의 출력, RBO(ripple blanking out)를 추가로 처리해야 한다. 앞서는 0이라는 것은 상위의 입력 값이 모두 0이고 현재의 값도 0인 seven-segment를 지칭하고, 뒤따르는 0은 하위의 모든 값이 0이고 현재의 값도 0일 경우를 지칭하는 것이다. 여러 seven-segment display를 연결하여 사용할 때, 위와 같은 경우의 seven-segment를 off시키는 것이 효과적인 display를 가능하게 하기 때문이다.

2.2.1 추가된 기능들의 처리 방법 분석

(1) Blanking input master (BIM:모든 seven-segment를 off.)

Blanking기능은 기본적으로 BI(blanking input)입력이 들어오면 작동한다. 또는, 램프 테스트 상태가 아닌 경우 (LT=0)에 현재 단이 leading zero일 경우, 즉, 현재의 BCD 입력이 0($\overline{DCBA}$)이고 전단의 74LS47의 RBO (ripple blanking output: 앞의 모든 값이 0이라는)신호가 현재 단의 RBI로 입력되는 상황에서 동작하게 된다. 실제로 BIM은 외부의 핀으로 할당된 것은 아니고, blanking이 일어나는 여러 조건들을 하나의 신호로 표현한 것이다. BIM은 다음과 같고, 위치는 그림 4에 표시되어 있다. .

$$BIM = BI + (\overline{LT}\cdot RBI\cdot\overline{DCBA})$$

$$\overline{BIM} = \overline{BI}\cdot\overline{\overline{LT}\cdot RBI\cdot\overline{DCBA}}$$

a) $\overline{\overline{LT}\cdot RBI\cdot\overline{DCBA}}$

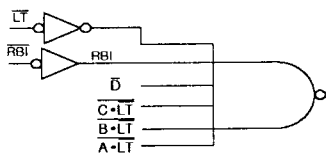

그림 5

b) $\overline{BI}\cdot\overline{\overline{LT}\cdot RBI\cdot\overline{DCBA}}$

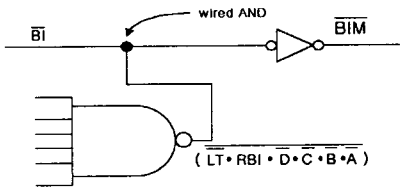

그림 6

(2) 램프 테스트 (LT:모든 seven-segment를 on.)

LT=1이면, $\overline{BIM}=\overline{BI}$ 또는 $BIM=BI$이다. 즉, 이 때의 blanking은 외부의 BI에 의해서만 동작되어진다. 만일 BI=1이면, LT=1일지라도, blanking이 동작한다. BI가 LT에 우선 순위를 가지고 있는 것이다. 만일 LT=1이고, BI=0라면, 모든 seven-segment는 on된다. 이 동작은 LT=1를 이용해서 입력의 상태를 CBA=000으로 만들어 seven-segment가 0(0000)또는 8(1000)을 표시하도록 하고(그림 7), 이와 함께 'g' segment를 on시켜(그림 8) 결과적으로 8(1000)을 표시해 모든 seven-segment가 on되도록 한다.

$$g = (D+C+B+LT)\cdot(\overline{C}\cdot\overline{BI}+\overline{B}\cdot\overline{BI}+\overline{A}\cdot\overline{BI})$$

LT=1, BI=0, C=B=A=0,

$$g = (D+1)\cdot(\overline{0}\cdot\overline{0}+\overline{0}\cdot\overline{0}+\overline{0}\cdot\overline{0}) = 1$$

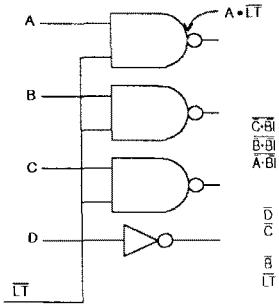

그림 7

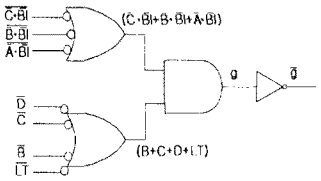

그림 8

(3) Ripple Blanking Out (RBO:상위부터 현재까지 모두 0 임을 나타냄.)
a)상위의 74LS47로부터 RBI입력이 있고 (RBI=1),
b)램프 테스트 상태가 아니며(LT=0),
c)현재의 BCD입력이 0 (DCBA=0000).
위의 세 가지 조건이 동시에 만족되면, 현재의 0값이 앞서는 0임을 의미하고, RBO 신호가 다음 단의 74LS47의 RBI신호로 전달된다.

$$RBO = RBI\cdot\overline{LT}\cdot\overline{D}\,\overline{C}\,\overline{B}\,\overline{A}$$

$$\overline{RBO} = \overline{RBI\cdot\overline{LT}\cdot\overline{D}\,\overline{C}\,\overline{B}\,\overline{A}}$$

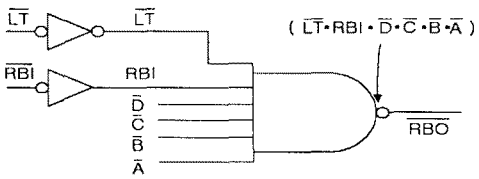

그림 9

74LS49의 디자인에서 사용된 방법들, [정리 1] 과 necessary redundant,과 지금까지 분석한 추가적인 기능들의 처리방법을 이용해서, 2개의 추가적인 입력과 1개의 출력을 포함한 전체의 74LS47 디자인에서도, 74LS49와 마찬가지로, 단지 8 종류의 신호 ($A\cdot\overline{LT}$, $B\cdot\overline{LT}$, $C\cdot\overline{LT}$, D, $\overline{A\cdot\overline{LT}\cdot\overline{BIM}}$, $\overline{B\cdot\overline{LT}\cdot\overline{BIM}}$, $\overline{C\cdot\overline{LT}\cdot\overline{BIM}}$, $\overline{D\cdot\overline{BIM}}$)만을 이용해서 모든 출력을 구현해 낼 수 있다. 단지, 추가된 기능들의 처리로 74LS49의 경우와 약간 다른 형태의 신호를 필요로 한다.

3. 결 론

이 논문은 BCD-to-seven segment 디코더인 74LS49와 74LS47의 디자인을 역으로 분석한 것이다. 이 칩들은 일반적으로 사용되지 않는 로직최소화방법을 효과적으로 이용해서 매우 간결하게 디자인되었다.
[정리 1]과 necessary redundant의 적용은 매우 독특하고, 74LS49와 74LS47의 출력들을 쉽게 구현할 수 있도록 했다. 이 방법을 사용하지 않았다면, $\overline{BI}$는 다른 신호들과 분리되어서 처리되어야 하고, 그 결과로 각각의 출력들을 구현하는데 있어서 더 복잡한 회로를 필요로 하게 된다.
이 기법은 다른 많은 분야에서 넓게 응용될 수 있을 것이다. 이 칩의 경우는 비교적 간단한 회로의 구성을 가지고 있지만, 복잡한 회로 상에서 위의 방법의 적용은 상당한 효과를 얻을 수 있을 것으로 기대된다.
램프 테스트, RBI, RBO 신호들의 처리는 일반적인 논리회로에 새로운 기능을 추가할 경우에 응용되어질 수 있을 것이다. 이 방법들은 기본 디자인에 변형이나 손상을 주지 않으면서 새로운 기능을 확장시키고 있다. 램프 테스트는 약간의 수정을 통해서 다른 회로의 기능 테스트로 충분히 사용되어 질 수 있을 것이며, RBI 와 RBO의 기능은 같은 종류의 칩들을 연결해서 사용할 경우 상호간의 신호처리 방법으로 응용되어질 수 있을 것으로 보인다.

[참고문헌]

[1] John F. Wakerly, "Digital design: principles and practices", Prentice-Hall International Editions, 257-259, 1990.
[2] Texas Instruments Inc., "Designing with TTL Integrated Circuits", McGraw-Hill, 181-200, 1971.
[3]"Datasheets of 74LS47 and 74LS49",Texas Instruments.