챕터4 정리

Ch4: Applications of Boolean Algebra / Minterm and Maxterm Expansions

4.1 Conversion of English Sentences to Boolean Equations

· Steps in designing a single-output combinational switching circuit

1) Find switching function which specifies the desired behavior of the circuit

→ 원하는 동작 특성을 보이는 회로에 대한 기술을 찾음

2) Find a simplified algebraic expression for the function

→ 함수로 사용될 대수적 표현을 찾음

3) Realize the simplified function using available logic elements

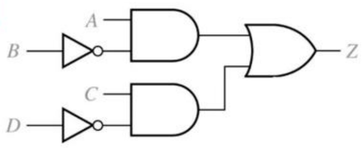
→ 논리적 요소들을 활용한 구현

- Example

1. The alarm will ring(Z) iff the alarm switch is turned on(A) *and* the door is not closed(B'), *or* it is after 6PM(C) and window is not closed(D')

2. Boolean Equation $Z = AB' + CD'$

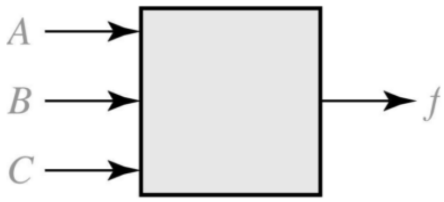
3. Circuit realization



4.2 Combinational Logic Design Using a Truth Table

· 표를 보고 만들기

- Combinational Circuit with Truth Table



A B C	f	f'
0 0 0	0	1
0 0 1	0	1
0 1 0	0	1
0 1 1	1	0
1 0 0	1	0
1 0 1	1	0
1 1 0	1	0
1 1 1	1	0

· When expression for f=1 → 1에 초점을 맞춰서 하는 것을 minterm이라 한다

$f = A'BC + AB'C' + AB'C + ABC' + ABC$ (SOP 형태)

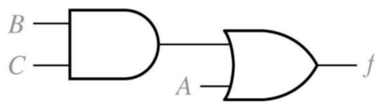
- Original equation

$f = A'BC + AB'C' + AB'C + ABC' + ABC$

- Simplified equation

$f = A'BC + AB' + AB = A'BC + A = A + BC$

- Circuit realization



· When expression for f=0 → 0에 초점을 맞춰서 하는 것을 maxterm이라 한다

$f = (A + B + C)(A + B + C')(A + B' + C)$ (POS 형태)

$f = (A + B)(A + B' + C) = A + B(B' + C) = A + BC$

- When expression for f′=1 (0으로 표현하는 방식을 쉽게 얻는 방법: f′의 1 초점으로 맞춰)

$f' = A'B'C' + A'B'C + A'BC'$

- And take the complement of f′

$f = (A + B + C)(A + B + C')(A + B' + C)$

4.3 Minterm and Maxterm Expansions

· Minterm 표현과 Maxterm 표현

Row No.	A B C	Minterms	Maxterms
0	0 0 0	$A'B'C' = m_0$	$A + B + C = M_0$
1	0 0 1	$A'B'C = m_1$	$A + B + C' = M_1$
2	0 1 0	$A'BC' = m_2$	$A + B' + C = M_2$
3	0 1 1	$A'BC = m_3$	$A + B' + C' = M_3$
4	1 0 0	$AB'C' = m_4$	$A' + B + C = M_4$
5	1 0 1	$AB'C = m_5$	$A' + B + C' = M_5$
6	1 1 0	$ABC' = m_6$	$A' + B' + C = M_6$
7	1 1 1	$ABC = m_7$	$A' + B' + C' = M_7$

→ 길게 표현하기 귀찮으니 m_0, m_1, … 식으로 표현하기로 약속함

· Minterm

- Minterm of n variables is a product of n literals in which each variable appears exactly once in either true (A) or complemented form(A'), but not both.

  → 즉 대충 A나 A'중에 하나만 존재한단 이야기, 곱의 형식이란 이야기

- Minterm expansion, Standard Sum of Product

  $f = A'BC + AB'C' + AB'C + ABC' + ABC$

  $f(A, B, C) = m_3 + m_4 + m_5 + m_6 + m_7$

  $f(A, B, C) = \sum m(3, 4, 5, 6, 7)$

· Maxterm

- Maxterm of n variables is a sum of n literals in which each variable appears exactly once in either true (A) or complemented form(A'), but not both.

  → 즉 대충 A나 A'중에 하나만 존재한단 이야기, 합의 형식이란 이야기

- Maxterm expansion, Standard Product of Sum

  $f = (A + B + C)(A + B + C')(A + B' + C)$

  $f(A, B, C) = M_0M_1M_2$

  $f(A, B, C) = \prod M(0 ,1 ,2)$

· Minterm and Maxterm expansions are complement each other

$f(A, B, C) = m_3 + m_4 + m_5 + m_6 + m_7 \rightarrow f' = m_0 + m_1 + m_2 = \sum m(0, 1, 2)$

$f(A, B, C) = M_0M_1M_2 \rightarrow f' = \prod M(3, 4, 5, 6, 7) = M_3M_4M_5M_6M_7$

$f' = (m_3 + m_4 + m_5 + m_6 + m_7)' = m_3'm_4'm_5'm_6'm_7' = M_3M_4M_5M_6M_7$

$f' = (M_0M_1M_2)' = M_0'M_1'M_2' = m_0 + m_1 + m_2$

4.4 General Minterm and Maxterm Expansions

· General Minterm and Maxterm Expansions

- General truth table for 3 variables (a_i is either '0' or '1')

A B C	F
0 0 0	a_0
0 0 1	a_1
0 1 0	a_2
0 1 1	a_3
1 0 0	a_4
1 0 1	a_5
1 1 0	a_6
1 1 1	a_7

- Minterm expansion for general function

$$F = a_0m_0 + a_1m_1 + a_2m_2 + \ldots + a_7m_7 = \sum_{i=0}^{7} a_im_i$$

a_i =1, minterm m_i is present

a_i =0, minterm m_i is not present

- Maxterm expansion for general function

$$F = (a_0 + M_0)(a_1 + M_1)(a_2 + M_2)\ldots(a_7 + M_7) = \prod_{i=0}^{7}(a_i + M_i)$$

A가 1이면 괄호 안 식은 무조건 1이니 고려 안 해도 됨

a_i =1, a_i + M_i=1, Maxterm M_i is not present

a_i =0, Maxterm is present

- Maxterm에 Complement를 취하면 Minterm이 된다

$$F' = [\prod_{i=0}^{7}(a_i + M_i)]' = \sum_{i=0}^{7} a'_i M'_i = \sum_{i=0}^{7} a'_i m_i$$

→ All minterms which are not present in F are present in F'

- Minterm에 Complement를 취하면 Maxterm이 된다

$$F' = [\sum_{i=0}^{7} a_i m_i]' = \prod_{i=0}^{7}(a'_i + m'_i) = \prod_{i=0}^{7}(a'_i + M_i)$$

→ All maxterms which are not present in F are present in F'

- 일반 식

$$F = \sum_{i=0}^{2^n-1} a_i m_i = \prod_{i=0}^{2^n-1} (a_i + M_i)$$

$$F' = \sum_{i=0}^{2^n-1} a'_i m_i = \prod_{i=0}^{2^n-1} (a'_i + M_i)$$

- minterm의 특성: If i and j are different, $m_i m_j = 0$

  곱으로 표현하면 리터럴과 리터럴의 complement가 겹치기 때문.

- maxterm의 특성: If i and j are different, $M_i + M_j = 1$

  리터럴과 리터럴의 complement가 겹치기 때문.

· Conversion between minterm and maxterm expansions of F and F'

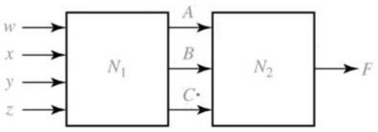
	Minterm Expansion of F	Maxterm Expansion of F	Minterm Expansion of F'	Maxterm Expansion of F'
Minterm Expansion of F		Maxterm nos. are those nos. not on the minterm list for F	List minterms Not present In F	Maxterm nos. Are the same As minterm nos. of F
Maxterm Expansion of F	Minterm nos. Are those nos. Not on the maxterm list forF		Minterm nos. Are the same as maxterm nos. of F	List maxterms not present inF

nos = numbers

4.5 Incompletely Specified Functions

· Don't Care Conditions

- If N_1 output does not generate all possible combinations of A, B, C, the output of N_2(F) has 'don't care' values.



→ N_1 구조에 따라 ABC의 모든 구조가 나오지 않는 경우가 존재함. 일어나지 않는 경우의 수에 대해 N_2는 일어나지 않는 케이스가 존재함; 신경 쓰지 않아도 되는 것

- Truth Table with Don't Cares

A B C	F
0 0 0	1
0 0 1	X
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	0
1 1 0	X
1 1 1	1

- Finding Function: 강의 PDF의 예시(Page.17)

Case 1: assign '0' on X's

→ X를 0으로 보고 처리함

Case 2: assign '1' to the first X and '0' to the second 'X'

→ 첫 번째를 1, 두번째에 0으로 보고 처리함

Case 3: assign '1' on X's

→ X를 1로 보고 처리함

- Don't Care가 있을 때의 표현

- Minterm expansion for incompletely specified function

$$F = \sum m(0,3,7) + \sum d(1,6)$$

Don't Cares

- Maxterm expansion for incompletely specified function

$$F = \prod M(2,4,5) \prod D(1,6)$$

4.6 Examples of Truth Table Construction

· Example 1: Binary Adder

a	b	Sum	
0	0	0 0	0+0=0
0	1	0 1	0+1=1
1	0	0 1	1+0=1
1	1	1 0	1+1=2

→

A	B	X	Y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

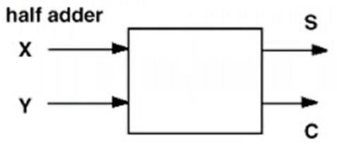
$$X = AB , Y = A'B + AB' = A \oplus B$$

X를 carry, Y를 sum

4.7 Design of Binary Adders and Subtracters

· Half Adder: bit단위에서의 합과 carry만을 고려함

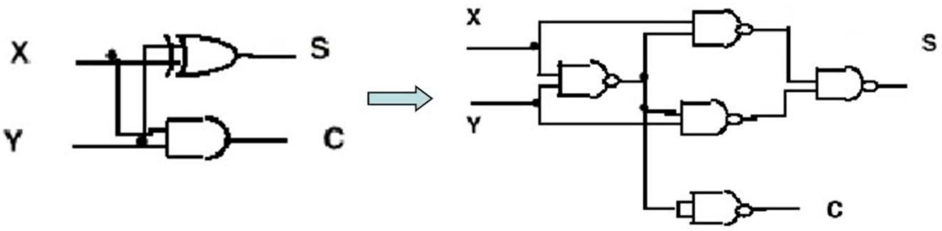
Half Adder



X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$S = X \oplus Y$

$C = XY$

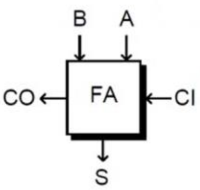


아까 위의 식에서 S와 C만 바꾼 거임

오른쪽 식은 나중에 NAND, NOR만으로 회로 구성 시에 사용하는 방식. 나중에 후술

· Full Adder: 아랫단에서 올라오는 carry까지 고려함

Full Adder

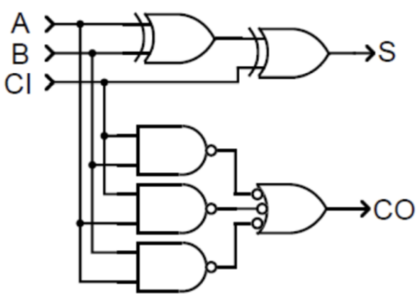


CI	B	A	S	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

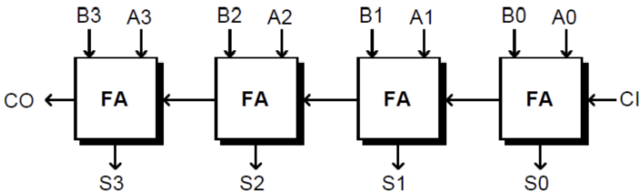
$$
\begin{aligned}
S &= A'\bullet B'\bullet C + A\bullet B'\bullet CI' + A\bullet B\bullet CI + A'\bullet B\bullet CI' \\
&= B'\bullet(A'\bullet CI + A\bullet CI') + B\bullet(A'\bullet CI + A'\bullet CI') \\
&= B'\bullet(A\oplus CI) + B\bullet(A\oplus CI)' \\
&= B\oplus(A\oplus CI) \\
&= A\oplus B\oplus CI \\
CO &= A\bullet CI + B\bullet CI + A\bullet B
\end{aligned}
$$

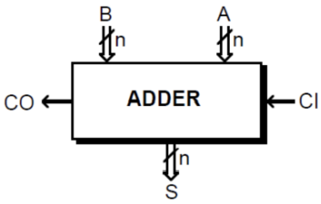
CI = Carry In, CO = Carry Out

Full Adder



- Carry Ripple Adder or Ripple Carry Adder (Cascading Full Adder, 둘 다 같은 표현임)





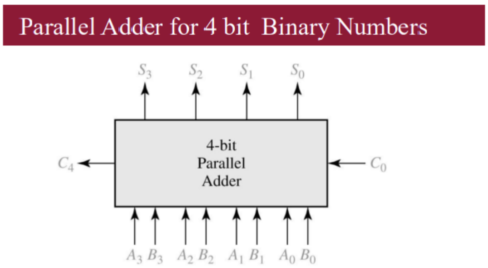
→4비트 단위 수를 더한다 했을 때

LSB부터 시작해서 Full Adder를 직렬로 연결함. 맨 처음 CI는 0으로 할당함.

비트 수가 커진다면 일일이 그리는 것이 번거로움 → 아래 그림과 같이 표현.

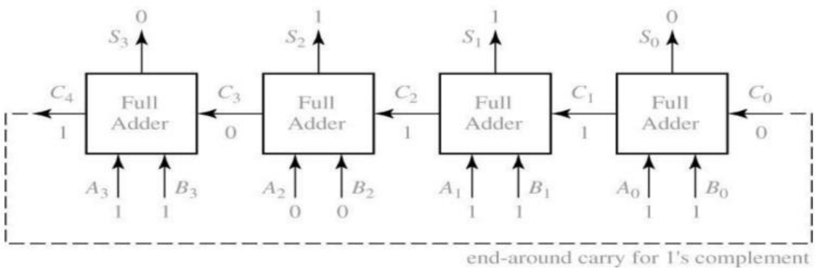
두 줄 짜리 화살표에서의 n은 비트 수를 표시하는 것임

- Parallel Adder for 4 bit Binary Numbers



10110 (carries)
  1011
+1011
10110

- Parallel adder composed of four full adders ← Carry Ripple Adder (slow!)



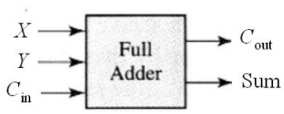
Carry가 물결처럼 넘어가는 것 같다 하여 carry ripple adder라고 함

점선은 end-around carry을 기술적으로 표현한 것임

1의 보수에서 나오는 그 end-around carry임

이렇게 하면 음수의 덧셈 또한 처리할 수 있다.

- Truth Table for a Full Adder



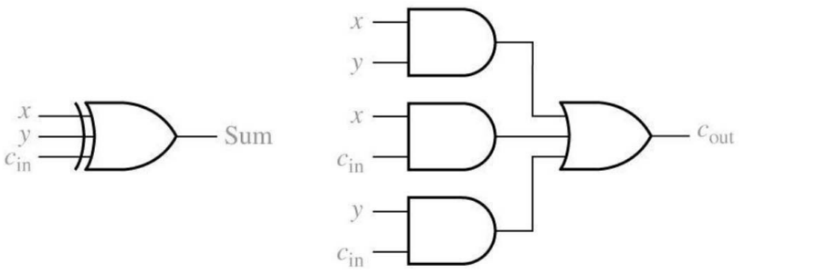
Truth Table for a Full Adder

X	Y	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

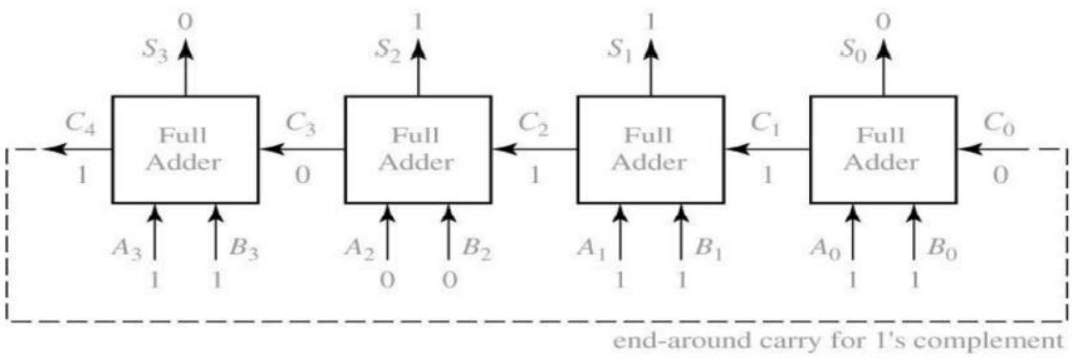
- Sum & Carry

$$\begin{aligned} Sum &= X'Y'C_{in} + X'YC'_{in} + XY'C'_{in} + XYC_{in} \\ &= X'(Y'C_{in} + YC'_{in}) + X(Y'C'_{in} + YC_{in}) \\ &= X'(Y \oplus C_{in}) + X(Y \oplus C_{in})' = X \oplus Y \oplus C_{in} \end{aligned}$$

$$\begin{aligned} C_{out} &= X'YC_{in} + XY'C_{in} + XYC'_{in} + XYC_{in} \\ &= (X'YC_{in} + XYC_{in}) + (XY'C_{in} + XYC_{in}) + (XYC'_{in} + XYC_{in}) \\ &= YC_{in} + XC_{in} + XY \end{aligned}$$



- When 1's complement is used, the end-around carry is accomplished by connecting C4 to C0 input.



1의 보수 체계가 사용되면 정확한 연산을 위해 end-around carry 메커니즘이 필요함

- Overflow(V) when adding two signed binary number

충분한 자릿수를 사용하지 않았을 때 사용하는 현상.

정확한 연산이 일어나는지 확인하기 위해 오버플로우를 체크하는 함수가 필요함

$V = A'_3B'_3S + A_3B_3S'$

→ 음수 음수 들어왔는데 양수 되거나 혹은 양수 양수 들어왔는데 음수 되는 경우만

· Subtracters: 덧셈으로도 충분히 구현 가능하지만 굳이 만들고 싶다면야..

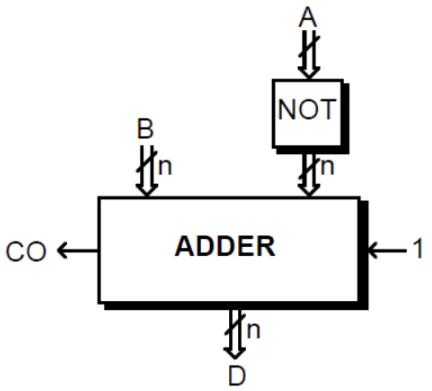
- Binary Subtracter using full adder

기존의 덧셈기로도 뺄셈 작용을 수행할 수 있다.

Convert an adder into a subtracter by inverting the subtrahend and setting the CI to 1

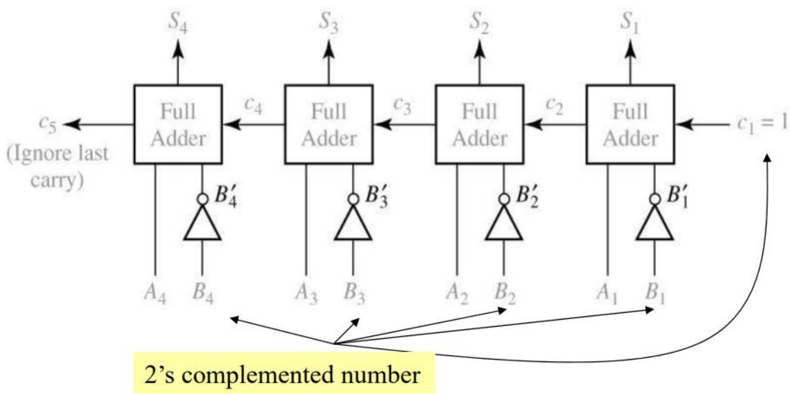
빼는 수에 음수를 취한다(마이너스를 취함): 양수 1은 0으로 0은 1로 한 후 1을 더함

: 2's complemented number



→ B – A 를 계산: 인버터를 취하고 1 을 더함

Subtraction is done by adding the 2's complemented number to be subtracted

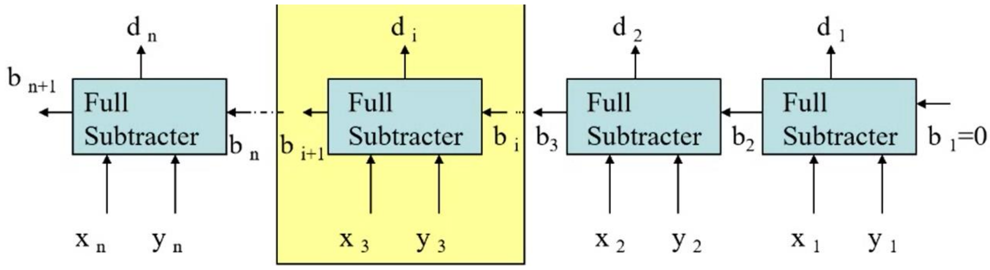


→ A – B 를 계산: 전부 complement 취한 후 1 더함

2 의 보수 체계는 최상위 carry 무시

- Alternative Subtracters - Parallel Subtracter

Full Adder 자체를 보수 체계 안 쓰고 그냥 기본 게이트를 써서 구성하고 싶다면



Truth Table for a Full Subtracter

x_i	y_i	b_i	b_{i+1}	d_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

여기서의 b: borrow, 즉 아랫단에 빌려준 수를 의미함