



# Quine-McCluskey

HW2

컴퓨터정보공학부/2015722031/박태성

| 디지털논리회로 1 | 2019 년 4 월 23 일

## Problem statement

처음에 boolean expression 을 간소화시키는 방법으로 karnaugh map 을 배운다. 이 편리한 방법에는 치명적인 단점이 있다. Input bit length 가 4 개 이상이 되면 사람이 계산하기 복잡하다는 것이다. 따라서 input bit length 에 제한이 없는 Quine-McCluskey(QM)를 이용하여 간소화시킨다. 그러나, 이 또한 사람보다 컴퓨터가 계산하는 것이 빠르다. 따라서 C++로 QM algorithm 을 구현하려고 한다.

## Psuedo code and flow chart

- Psudo code

Main()

BEGIN

1. Txt file 에서 데이터를 읽어온다.
2. 읽어온 데이터에서 binary 정보만 따로 저장한다.
3. Hamming distance 가 1 인 minterm 끼리 비교해서 보수를 삭제한다.
4. Prime implicant(PI)만 남을 때까지 과정 3 을 반복한다.
5. PI table 을 만들어서 essential PI 를 구한다.
6. Essential PIs 를 통해 tansistor 의 개수를 계산한다.
7. Essential PIs 와 cost of transisotr 를 txt 로 저장한다.

END

Reduce()

BEGIN

1. If(is\_hamming\_distance\_is\_o)  
    Replace\_complements()  
    Endif
2. If(! In\_vector())  
    Minterms\_b.append()  
    Endif
3. Return Minterms\_b

END

Is\_hamming\_distance\_is\_o()

BEGIN

1. Flag <- 0
2. For 0<=i<=a.lenght(),i++  
    If(a[i]!=b[i])  
        Flag++  
    Endif
3. Return (flag==1)

END

```

Replace_complements()
BEGIN
1. Temp <- ""
2. For o<=i<=a.lenght(),i++
    If(a[i]!=b[i])
        Temp <- temp + "-"
    Else
        Temp <- temp+a[i]
    Endif
3. Retrurn temp
END

```

```

In_vector()
BEGIN
1. For o<=i<=a.size(),i++
    If(a[i].compare(b)==o)
        Return true
    Else
        Return false
    Endif
END

```

```

Vectors_equal()
BEGIN
1. If(a.size()!=b.size())
    Return false
    Endif
2. For o<=i<=a.size(),i++
    If(a[i]!=b[i])
        Return false
3. Return true
END

```

```

Identify_minimal_PI_set()
BEGIN
1. **PI <- *[minterms_b.size()]
2. Make_pair(cnt, minterms_b[i])
3. Vector <int> entry_of_PI
END

```

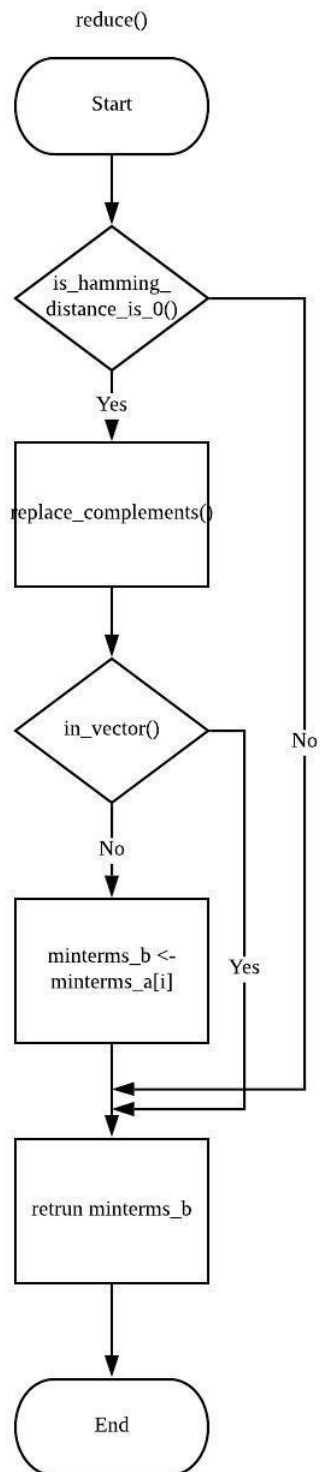
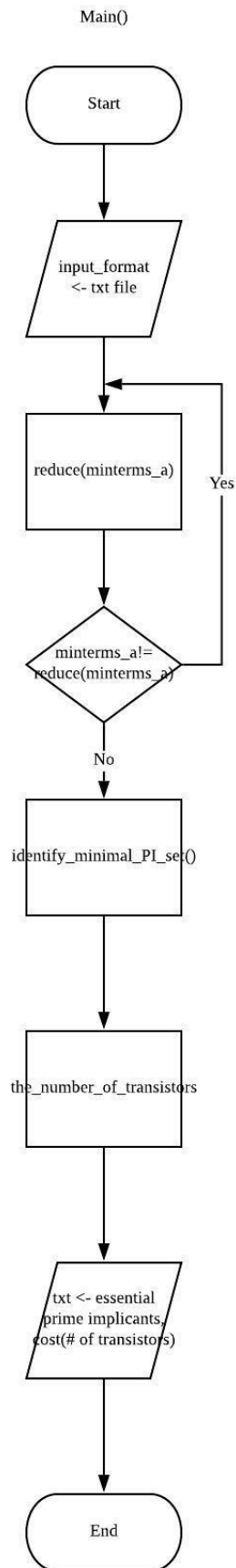
```

The_number_of_transitors()
BEGIN
1. Num_of_OR <- entry_of_PI.size()
2. NUM_of_AND
3. NUM_of_NOT

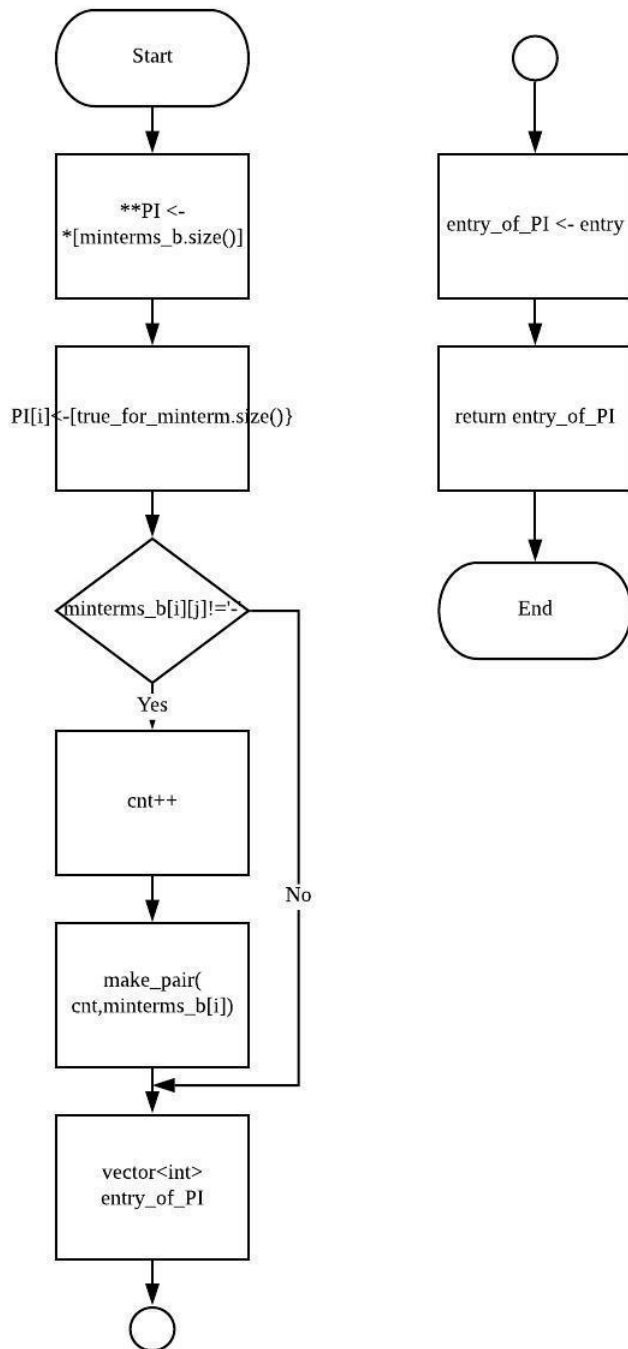
```

4. Cost <- (Num\_of\_OR\*2+2)+ (NUM\_of\_AND \*2+2)+ NUM\_of\_NOT\*2  
END

- Flow chart



Identify\_minimal\_PI\_set()



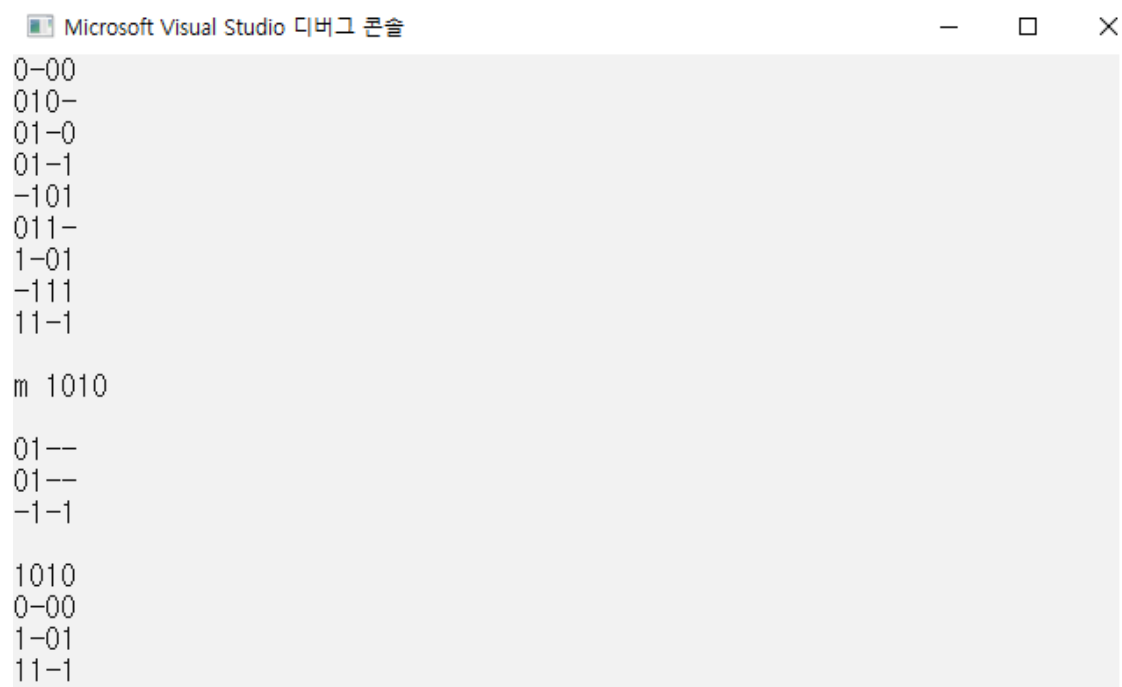
## Verification strategy

처음 전략은 각 minterm 들의 1 의 개수를 세서 pair 로 1 의 개수와 string type 으로 처리된 데이터를 묶으려고 했었다. 1 번 reduce 를 거치면 string type 으로 binary 데이터를 저장한다. 이때 반복문으로 reduce 과정을 거치게 하려면 1 번 reduce 를 거친 데이터가 원본 데이터에 덮어 씌워져야하는데 위의 방법은 형식 차이 때문에 불가능했다. 게다가, 1 번 reduce 된 데이터들 중에 중복되는 minterm 이 발생하여 이미 있는 minterm 인지 확인해주는 in\_vector()를 만들 필요가 있었다.

다음 그림은 Pair 로 1 의 개수와 txt 파일에서 가져온 정보를 묶는 과정이다.

```
pair<int, string> p;  
  
for (int i = 1; i < cnt; i++) {  
    int one = 0;  
    for (int j = 2; j < size + 2; j++) {  
        if (tmp[i][j] == '1')  
            one++;  
    }  
    p = make_pair(one, tmp[i]);  
    v.push_back(p);  
}
```

다음 그림은 in\_vector()와 원본 데이터와 1 번 reduce 된 데이터의 type 이 맞지 않았을 때 출력된 결과이다.



```
Microsoft Visual Studio 디버그 콘솔  
0-00  
010-  
01-0  
01-1  
-101  
011-  
1-01  
-111  
11-1  
  
m 1010  
  
01--  
01--  
-1-1  
  
1010  
0-00  
1-01  
11-1
```

반복문은 reduce 를 거친 데이터가 직전 데이터와 같으면 멈추는 조건을 갖는다. 따라서, reduce 를 거치는 것이 선행되어야 한다. 따라서 do while 문을 사용하였다.

Coverage table 을 코드로 구현하는 것에서 아이디어가 떠오르지 않아 어려움이 있었다. 각 binary 자리를 비교하여 체크하고 체크된 개수의 합을 따져 essential prime

implicant 를 구하였다. Essential prime implicant 를 통해 transistor 개수를 구하는 것은 식을 세워 계산하도록 구현하였다.

Quine-McCluskey 를 구현하면서 모듈화의 중요성과 사전 계획이 중요하다는 것을 느꼈다. 코드를 짜다가 미궁에 빠져서 내가 어떤 문제를 해결하고자 했는지 잊어버리는 경우가 많았다. 따라서 코딩 전에 슈도코드나 플로우차트를 그려놓고 방향을 잃어버리면 참고했다.

PI 테이블을 만들지 않아도 합의 법칙( $x+x'=1$ )을 이용해서 prime implicants 들의 boolean equation 을 간략화하면 essential prime implicant 들의 합으로 표현할 수 있다고 웹에서 찾았다. 그래서, prime implicants 들의 boolean equation 을 만들어 주는 함수를 코드에 추가해서 간략화 해보려고 했다. 그러나 합의 법칙만으로 간략화가 되지 않아 결국 PI 테이블을 만들었다. 웹에서 참고한 자료는 다음과 같다.

#### 4. 합 $f =$ 주항들의 합

$$f = (1,5) + (5,7) + (6,7) + (0,1,8,9) + (0,2,8,10) + (2,6,10,14) \\ = a'c'd + a'bd + a'bc + b'c' + b'd' + cd'$$

여분의 주항을 제거하기 위해 합의 정리 사용  $\rightarrow$  6.2 주항차트를 이용하면 더 빠름

$$f = a'bd + b'c' + cd'$$

과제 폴더에 첨부한 input\_minterm.txt~input\_minterm5.txt 로 테스트 해보았을 때 8 비트까지는 정상 작동하는 것을 확인하였다.

```
result.txt - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
00-0001-
00000--0
00--0000
000-1001
00-0001-
0011-000
00111110
0-000-00
0-00-000
11111000
Cost(# of transistors): 274
```



## Testbench

- input\_minterm.txt
- input\_minterm1.txt
- input\_minterm2.txt
- input\_minterm3.txt
- input\_minterm4.txt
- input\_minterm5.txt

제출 폴더에 함께 제출한 test file 이다. 이중 input\_minterm1 과 input\_minterm5 는 각각 10 비트, 9 비트를 필요로 한다. 10 비트 test file 은 비트 수가 많고 9 비트 test file 은 데이터의 양이 많다. 두 test file 모두 나의 코드에서는 정상적인 값이 출력되지 않았다. 10 비트 test file 은 PI table 에서 모두 걸러내지 못한 것으로 추측하고 9 비트 test file 은 string class 에서 처리할 수 있는 데이터의 양을 초과하여 compiler 가 오류를 내보내는 것으로 추측하고 있다.