2차 과제 보고서

학과: 컴퓨터정보공학부

학번: 2024402047

이름: 하성준

1. 문제 정의

Quine–McCluskey 알고리즘을 이용하여 임의 개수의 입력 비트에 대해 SOP(Sum-of-Products) 형태로 Boolean function을 최소화하고, 이를 실제 회로로 구현시 필요한 트렌지스터의 수를 구하는 프로그램을 작성한다.

2. 알고리즘 설명

2-1. 전체 흐름

i. 입력 파일(input_minterm.txt)에서 비트 수(bit) 및 minterm/ don't care(term) 목록 파싱
ii. 초기 Implicant 그룹화 및 prime implicant 후보 수집
iii. 반복 결합(combine) 과정을 통해 새로운 Implicant 그룹 생성
iv. 결합 불가능해질 때까지 반복 후, 중복 제거 및 don't care 전용 implicant 제거
v. 커버 차트 생성 → Essential PI 추출 → 남은 minterm에 대해 Petrick 방법 적용
vi. 최종 implicant 집합을 기반으로 패턴 및 트랜지스터 비용 계산 → 결과 파일 (result.txt) 생성

2-2. 주요 클래스 및 함수

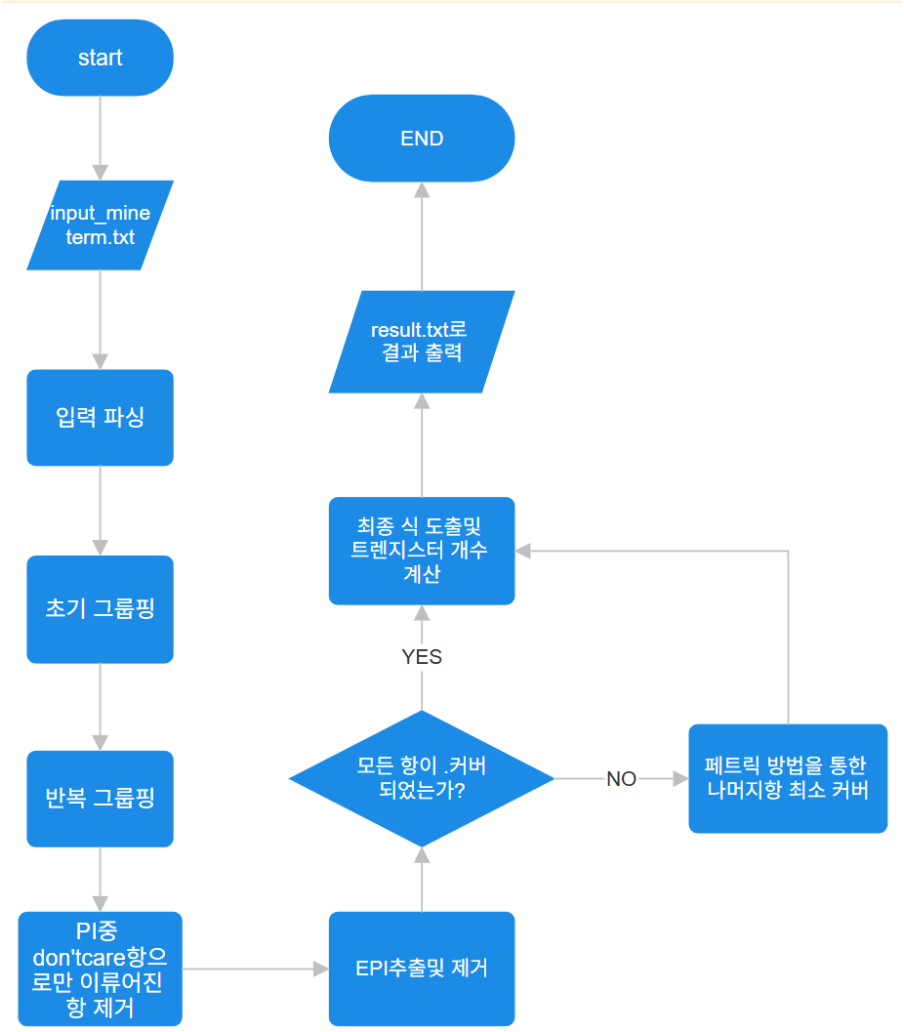
i. Term 클래스: Minterm 또는 don't care 값 저장 및 이진 문자열 반환
ii. Implement 클래스: Implicant(결합된 term 집합) 및 mask 관리, prime 여부 표시
iii. CanCombine(a,b): mask 동일, 단일 비트 차이 여부 판단
iv. removeRedundantTerms(): 중복 prime implicant 제거
v. generateCoverChart(): Prime Implicant × Minterm 커버 차트 생성
vi. findEssentialPIs(): Essential Prime Implicant 인덱스 식별
vii. removeEssentialCoverage(): Essential Prime Implicant 가 커버하는 minterm 차트에서 제거
viii. buildPetrick(): Petrick 방법으로 남은 minterm 커버 가능한 implicant 조합 생성
ix. selectMinCombination(): Petrick 조합 중 최소 개수 조합 선택

2-3. Pseudo Code

```
Read bit from input file
Parse each line into Term(is_minterm, value)
for each pair (i, j) of Terms:
    if single_bit_diff(terms[i], terms[j]):
        add Implement(diff_mask, terms[i], terms[j])
        mark both as non-prime
for each Term still prime:
    add Implement(0, term) to prime_impls
while changed:
    next_group = {}
    for each pair (p, q) in current_group:
        if CanCombine(p, q):
            mk = p.mask ∪ diff(p, q)
```

```
                next_group.add(Implement(mk, p.terms, q.terms))
                mark p,q as non-prime
        collect remaining prime from current_group into prime_impls
        current_group = next_group
removeRedundantTerms(prime_impls)
remove all prime_impls with only don't care
remove don't care Terms from terms list
generateCoverChart(prime_impls, terms, chart, coverMap)
essential = findEssentialPIs(chart)
removeEssentialCoverage(chart, coverMap, essential)
combos = buildPetrick(chart)
petrickSel = selectMinCombination(combos)
finalSet = essential ∪ petrickSel
for each idx in finalSet:
        compute pattern string with '_' for mask bits
        accumulate inverter cost (2 trans.) for 각 변수당 최초 inversion
        accumulate AND gate cost: 2×(input_count)+2
add OR gate cost: 2×|finalSet|+2
write patterns and totalCost to result.txt
```

2-4. Flow chart



3. 검증 전략 및 설명 포함 예시

설명	비트 수	term	기대 출력	실제 출력
과제 기본 예제	5	d 00000 m 00100 m 00101 m 00110 m 01001 m 01010 d 00111 d 01101 d 01111 m 11111	001__ 01_01 01010 _1111 Cost (# of transistors): 60	01010 001__ 01_01 _1111 Cost (# of transistors): 60
단일 minterm	3	m 010	010 Cost (# of transistors):12	010 Cost (# of transistors): 12
단일 2input-and	2	m 11	11 Cost (# of transistors):6	11 Cost (# of transistors): 6
단일 2input-xor	2	m 01 m 10	01 10 Cost (# of transistors): 22	01 10 Cost (# of transistors): 22
모든항 결합 불가	5	m 00000 m 00111 m 11100 m 11111	00000 00111 11100 11111 Cost (# of transistors): 68	00000 00111 11100 11111 Cost (# of transistors): 68
Essential Prime Implicant 만으로 모든항 커버 불가	6	m 000000 m 000001 m 000011 m 000111 m 001111 m 011111 m 111111	00000_ 0000_1 00_111 _11111 Cost (# of transistors): 68	00000_ 0000_1 00_111 _11111 Cost (# of transistors): 68

4. 매우 어려운 테스트 케이스

4-1. 입력

4-2. 6 000000 m 000001 m 000011 m 000111 m 001111 m 011111 m 111111

4-3. 예상 출력

00000_
0000_1
00_111
_11111
Cost (# of transistors): 68

4-4. 실제 출력

```
00000_
0000_1
00_111
_11111
Cost (# of transistors): 68
```