논리회로설계/ 결과보고서

1. Row Dominance

```
void rd(vector<string> &pi, vector<vector<bool>> &piMap) {
   vector<int> delIdx;
   bool flag;
   for (int i = 0; i < piMap.size(); i++) {
      for (int j = 0; j < piMap.size(); j++) {
         flag = true;
         if (i == j) {
            continue;
         }
         for (int k = 0; k < piMap[0].size(); k++) {
            if (piMap[i][k] != piMap[j][k]) {
               if (piMap[i][k] == false) {
                  flag = true;
                  break;
               }
               if (piMap[i][k] == true) flag = false;
            }
         }
         if (!flag) {
            delIdx.push_back(j);
         }
      }
   }
   sort(delIdx.begin(), delIdx.end());
   delIdx.erase(unique(delIdx.begin(), delIdx.end()), delIdx.end());
   for (int i = 0; i < delIdx.size(); i++) {
      pi.erase(pi.begin() + delIdx[i] - i);
      piMap.erase(piMap.begin() + delIdx[i] - i);
   }
   checkInterChange(pi, piMap);
}
```

```
        5   7   15
-111 |  0   1    1
0-01 |  1   0    0
01-1 |  1   1    0
111- |  0   0    1

<rd>
        5   7   15
-111 |  0   1    1
01-1 |  1   1    0
```

구현된 pi들과 테이블을 인자로 받고

```
void rd(vector<string> &pi, vector<vector<bool>> &piMap) {
```

for문을 순회하면서 i와 j를 각각의 pi로써 동작시켰다.

```
   for (int i = 0; i < piMap.size(); i++) {
      for (int j = 0; j < piMap.size(); j++) {}}
```

i와 j가 같은 경우에는 스스로를 확인하기에 확인하지 않았다.

```
 if (i == j)
            continue;
```

이후 각 민텀들을 커버하는지 확인하며 같거나 다르더라도 i가 트루인 경우에는 순회하고 그렇지 않은 경우에는 다음 pi를 확인하도록 break 시켰다.

```
for (int k = 0; k < piMap[0].size(); k++) {
            if (piMap[i][k] != piMap[j][k]) {
               if (piMap[i][k] == false) {
                  flag = true;
                  break;
               }
               if (piMap[i][k] == true) flag = false;
            }
```

```
        }
        if (!flag) {
            delIdx.push_back(j);
        }}}
```

이후 RD가 진행된 pi들과 pimap을 수정하였다

```
pi.erase(pi.begin() + delIdx[i] - i);
piMap.erase(piMap.begin() + delIdx[i] - i);
```

체크인터체인지 함수는 테이블을 순회하며 피아이들이 같은 민텀들을 커버한다면 인덱스를 저장하고 인터체인저블한 pi중 아래에 있는 피아이를 삭제했습니다.

```
    checkInterChange(pi, piMap);
```

2. Column Dominance

```
void cd(vector<string> &pi, vector<int> &tmpMinterm,
        vector<vector<bool>> &piMap) {
    vector<int> delIdx;
    bool flag;
    for (int i = 0; i < piMap[0].size(); i++) {
        for (int j = 0; j < piMap[0].size(); j++) {
            flag = true;
            if (i == j) {
                continue;
            }
            for (int k = 0; k < piMap.size(); k++) {
                if (piMap[k][i] != piMap[k][j]) {
                    if (piMap[k][i] == true) {
                        flag = true;
                        break;
                    }
                    if (piMap[k][i] == false) flag = false;
                }
            }
            if (!flag) {
                delIdx.push_back(j);
            }
        }
    }
    sort(delIdx.begin(), delIdx.end());
    delIdx.erase(unique(delIdx.begin(), delIdx.end()), delIdx.end());
    for (int i = 0; i < delIdx.size(); i++) {
        tmpMinterm.erase(tmpMinterm.begin() + delIdx[i] - i);
        for (int j = 0; j < piMap.size(); j++) {
            piMap[j].erase(piMap[j].begin() + delIdx[i] - i);
        }
    }
    checkInterChange(pi, piMap);
}
```

```
<rd>
          5   7   15
-111 |    0   1    1
01-1 |    1   1    0

<cd>
          5   15
-111 |    0    1
01-1 |    1    0
```

for문을 순회하면서 i와 j를 각각의 minterm으로 동작시켰다

```
for (int i = 0; i < piMap[0].size(); i++) {
      for (int j = 0; j < piMap[0].size(); j++) {
```

이후 각 민텀들이 커버되는지 확인하며 같거나 다르더라도 i가 거짓인 경우에는 순회하고 그렇지 않은 경우에는 다음 pi를 확인하도록 break 시켰다.

```
        for (int k = 0; k < piMap.size(); k++) {
            if (piMap[k][i] != piMap[k][j]) {
                if (piMap[k][i] == true) {
                    flag = true;
                    break;
                }
```

```
            if (piMap[k][i] == false) flag = false;
         }
      }
      if (!flag) {
         delIdx.push_back(j);
      }
```

체크인터체인지 함수는 테이블을 순회하며 피아이들이 같은 민텀들을 커버한다면 인덱스를 저장하고 인터체인저블한 pi중 아래에 있는 피아이를 삭제했습니다.

```
   checkInterChange(pi, piMap);
```

3. Rd & Cd & sEPI

```
while (true) {
      vector<vector<bool>> compare = piMap;
      cout << "<rd>\n";
      rd(pi, piMap);
      print(tmpMinterm, pi, piMap);

      cout << "<cd>\n";
      cd(pi, tmpMinterm, piMap);
      print(tmpMinterm, pi, piMap);

      vector<string> tmpepi = findEpi(pi, tmpMinterm, piMap);
      secondEpi.insert(secondEpi.begin(), tmpepi.begin(), tmpepi.end());

      if (compare == piMap) {
         if (piMap.size()) cout << "go to patrick \n";
         break;
      }
      cout << "<Sepi>\n";
      print(tmpMinterm, pi, piMap);
   }
```

로우 도미넌스와 컬럼 도미넌스를 더이상 진행할 수 없을때까지 진행하기위해 반복문을 사용했습니다.

반복문의 탈출 조건은 얕은 복사를 진행한 compare 벡터가 rd cd를 진행한 pimap과 같다면 더 이상 cd rd를 할 수 없다고 판단해 반복문을 탈출 했습니다.

여기서 테이블의 사이즈가 0 이상이라면 패트릭 메소드를 사용하라고 출력했습니다.

Rd 와 cd 함수를 실행시킨 후 findepi 함수를 실행시켜 epi를 찾고 삭제시켰습니다

```
vector<string> findEpi(vector<string> &pi, vector<int> &tmpMinterm,
                       vector<vector<bool>> &piMap) {
   vector<string> Epi;
   vector<pair<int, int>> epiCheak;

   for (int i = 0; i < tmpMinterm.size(); i++)
      epiCheak.push_back({tmpMinterm[i], 0});
```

```
    for (int i = 0; i < pi.size(); i++)
        countPi(epiCheak, pi[i], pi[0].size(), piMap[i]);
    cout << "<map>\n";
    print(tmpMinterm, pi, piMap);

    // epi 를 도출하고 삭제
    for (int i = 0; i < pi.size(); i++)
        countEpi(epiCheak, pi[i], pi[i], Epi, pi[0].size());

    Epi.erase(unique(Epi.begin(), Epi.end()), Epi.end());

    for (int i = 0; i < Epi.size(); i++) {
        deleteEpi(Epi[i], pi[0].size(), tmpMinterm, piMap);
        int idx = find(pi.begin(), pi.end(), Epi[i]) - pi.begin();
        pi.erase(remove(pi.begin(), pi.end(), Epi[i]), pi.end());
        piMap.erase(piMap.begin() + idx);
    }
    checkInterChange(pi, piMap);
    delEmpty(pi, piMap);

    return Epi;
}
```

4. 실행결과 {4, 7, 0, 1, 5, 7, 10, 14, 15}

```
         0   1   5   7  10  14  15
-111 |   0   0   0   0   0   0   0
0-01 |   0   0   0   0   0   0   0
000- |   0   0   0   0   0   0   0
01-1 |   0   0   0   0   0   0   0
1-10 |   0   0   0   0   0   0   0
111- |   0   0   0   0   0   0   0

<map>
         0   1   5   7  10  14  15
-111 |   0   0   0   1   0   0   1
0-01 |   0   1   1   0   0   0   0
000- |   1   1   0   0   0   0   0
01-1 |   0   0   1   1   0   0   0
1-10 |   0   0   0   0   1   1   0
111- |   0   0   0   0   0   1   1

         5   7  15
-111 |   0   1   1
0-01 |   1   0   0
01-1 |   1   1   0
111- |   0   0   1

<rd>
         5   7  15
-111 |   0   1   1
01-1 |   1   1   0

<cd>
         5  15
-111 |   0   1
01-1 |   1   0

<map>
         5  15
-111 |   0   1
01-1 |   1   0
```

```
<Sepi>
not found


<rd>
not found


<cd>
not found


<map>
not found


-111 0-01 000- 01-1 1-10 111- EPI 000- 1-10 SEPI -111 01-1
터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```