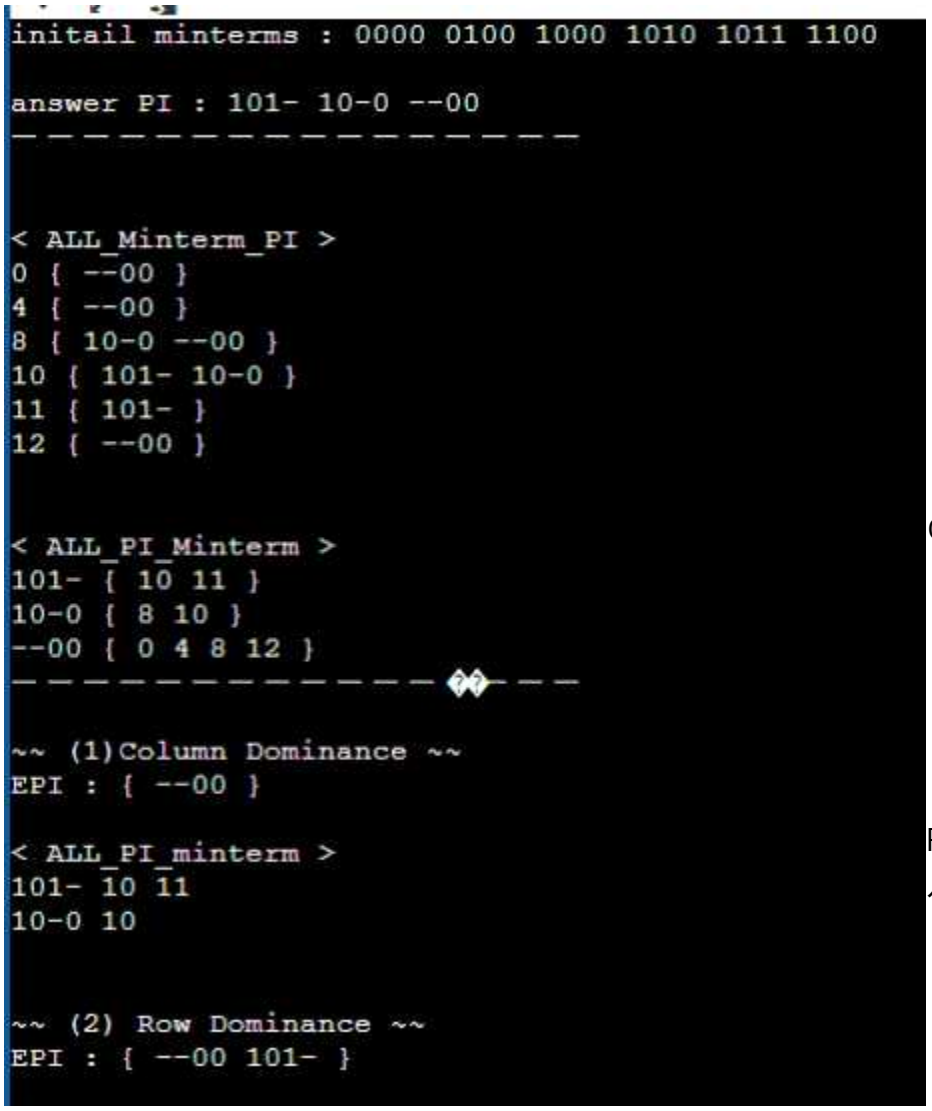
논리회로설계 추가 도전과제 보고서

<간단 설명>
: 기존의 PI 과제 코드에 덧붙여서, RD와 CD 계산을 위해
각 PI와 해당 써클에 cover된 minterm들을 같이 묶어서 표현한 ALL_PI_Minterm 2차원 벡터,
반대로 각 Minterm들과 해당 셀을 cover하는 PI를 같이 표현한 ALL_Minterm_PI 2차원 벡터
를 표현하기 위해 Make_PI_minterm 함수와 Make_minterm_PI 함수를 만들었음. 그리고 그렇게 생성된 AMP는 Find_CD 함수에서 Column Dominance 계산을 하여 첫 번째 EPI를 찾았고, APM을 Find_RD 함수에서 마저 계산해서 Secondary EPI를 찾도록 함.



입력값
: 4, 6, 0, 4, 8, 10, 11, 12

PI : 101-, 10-0, --00을
ALL_Minterm_PI,
ALL_PI_Minterm 버전으로
각각 표현했고,

Column Dominance에 AMP를 넣어서 첫번째 EPI인 --00을 찾은 후

--00이 cover하는 minterm
0,4,8,10을 APM에서 삭제하는
단계를 거친 뒤

Row Dominance에서 APM을 넣어서 Secondary EPI인 101-을 찾음.

(아래는 소스코드)

```
#include <iostream>
#include <string>
#include <vector>
#include <bitset>
#include <algorithm>
#include <cmath>
using namespace std;

int HD;
bool bAvailCombine = false;
vector <string> combine(vector<string> Imp, int i, int j);
vector<string> binary_to_Decimal(vector<string> PI_minterm);

bool availCombine( vector<string> &Imp){
    int iAvailCombine = 0;
    vector<string> AvailImp;
    vector<string> tmp;
    for(int i=0; i<Imp.size()-1; i++){
        for(int j=i+1; j<Imp.size(); j++){
            HD = 0;

            for(int k=0; k<Imp[0].size(); k++){
                if(Imp[i][k] != Imp[j][k])
                    HD++;
            }
            if(HD == 1){
                vector<string> tmp = combine(Imp, i, j);
                AvailImp.insert(AvailImp.end(), tmp.begin(), tmp.end());
            }else
                continue;
        }
    }
    sort(AvailImp.begin(), AvailImp.end());
        AvailImp.erase(unique(AvailImp.begin(),AvailImp.end()),AvailImp.end());


    if(!AvailImp.empty()) {bAvailCombine = true;  return true; }
    else { bAvailCombine = false; return false; }
}
vector<string> combine(vector<string> Imp, int i, int j){
    string tmp;
    vector<string> Vtmp;
    for(int k=0; k<Imp[0].size(); k++){
        if(Imp[i][k] == Imp[j][k]){
            if(Imp[i][k] == '2')
                tmp += '2';
            else
```

```
                tmp += Imp[i][k];
        }else if(Imp[i][k] != Imp[j][k]){
                tmp += '2';
        }
    }
    Vtmp.push_back(tmp);
    tmp = "";
    return Vtmp;
}
vector<string> compare(vector<string> Imp){
    bAvailCombine = false;
    string tmp;
    vector<string> newImp;
    int count;

    for(int i=0; i<Imp.size(); i++){
        int count = 0;
        for(int j=0; j<Imp.size(); j++){
            HD = 0;
            for(int k=0; k<Imp[0].size(); k++){       // 각 2진수끼리 값 비교
                if(Imp[i][k] != Imp[j][k])
                    HD++;
            }
            if(HD == 1){                                    // HD == 1 이면 병합시킴.
                vector<string> tmp = combine(Imp, i, j);
                newImp.push_back(tmp[0]);
                count++;
            }
        }
        if(count==0)           // 인접한 게 하나도 없으면 바로 push
            newImp.push_back(Imp[i]);
    }

    // for(int i=0; i<newImp.size(); i++){
    //       cout << "sort 전) newImp[" << i << "] : " << newImp[i] << endl;
    // }cout << endl;
    sort(newImp.begin(), newImp.end());
        newImp.erase(unique(newImp.begin(),newImp.end()),newImp.end());
    // for(int i=0; i<newImp.size(); i++){
    //       cout << "sort 후) newImp[" << i << "] : " << newImp[i] << endl;
    // }
    return newImp;
}
vector<string> solution (vector<int> minterm){
    vector<string> minterms;

    for(int i=2; i<minterm.size(); i++){
```

```
        string n;
        switch (minterm[0]) {
            case 1:{n = bitset<1>(minterm[i]).to_string(); break;}
            case 2:{n = bitset<2>(minterm[i]).to_string(); break;}
            case 3:{n = bitset<3>(minterm[i]).to_string(); break;}
            default :n = bitset<4>(minterm[i]).to_string();
        }
        minterms.push_back(n);
    }

    cout << "initail minterms : " ;
    for(int i=0; i<minterms.size(); i++){
         cout << minterms[i] << " ";
    } cout << endl;

    vector<string> Imp;
    for(int i=0; i<minterms.size(); i++)          // 2진수 변환해서 Imp벡터에 담기
        Imp.push_back(minterms[i]);

    if(availCombine(Imp) == 1)             // 첫 minterms들 중에서 하나라도 결합가능하면
        while(availCombine(Imp) == 1){        // 결합 불가능할 때까지 반복문으로 계속 결합시킴.

            compare(Imp).swap(Imp);
        }
    vector<string> Pi;
    Imp.swap(Pi);

    for(int i=0; i<Pi.size(); i++){                    // 2를 -로 바꾸기
        for(int j=0; j<Pi[0].size(); j++){
            if (Pi[i][j] == '2')
                Pi[i][j] = '-';
        }
    }
    return Pi;
}

vector<string> Make_PI_minterm(string PI){
    vector<string> PI_minterm;
    int count;

    PI_minterm.push_back(PI);

    count = 0;
    for(int j=0; j<PI.size(); j++){     // 각 인덱스 4자리 탐색
        if(PI[j] == '-')                  // '-' 개수 세기
            count++;
    }
```

```
    if(count == 0){
     PI_minterm.push_back(PI);
    }
    else if(count == 1){
        // cout << "if) count==1" << endl;
        for(int j=0; j<PI.size(); j++){   //각 인덱스 4자리 탐색
            if(PI[j] == '-')
                for(char a='0'; a<'2'; a++){
                    PI[j] = a;                          // 가령 000- 일 경우
                    PI_minterm.push_back(PI);          // tmp_minterm = {0000, 0001}
                }
        }
    }
    else if(count ==2){
        for(int j=0; j<PI.size(); j++)
            if(PI[j] == '-')
                for(char a='0'; a<'2'; a++){
                    PI[j] = a;
                    for(int k=0; k<PI.size(); k++)
                        if(PI[k] == '-')
                            for(char b='0'; b<'2'; b++){
                                PI[k] = b;
                                PI_minterm.push_back(PI);
                                PI[k] = '-';
                            }


                }


    }       // {101-, 1010, 1011}

    binary_to_Decimal(PI_minterm).swap(PI_minterm);

    return PI_minterm;

}

vector<string> binary_to_Decimal(vector<string> PI_minterm){
        int decimal;
        for(int i=1; i<PI_minterm.size(); i++){   //4번
            decimal = 0;
            string sDecimal = "";
            for(int j=0; j<PI_minterm[i].size(); j++){   //4자리 탐색
                int n = PI_minterm[i][j] - '0';
                decimal += n * pow(2, PI_minterm[i].size()-1-j);
                sDecimal = to_string(decimal);
```

```
            }
            PI_minterm.erase(PI_minterm.begin()+i);
            PI_minterm.insert(PI_minterm.begin()+i, sDecimal);
        }
        return PI_minterm;        // {101-, 10, 11}
}

vector<vector<string>> Make_minterm_PI(vector<vector<string>> ALL_Minterm_PI, vector<string> PI_minterm){

        // PI_minterm : {101-,10,11}을 받아와서
        // ALL_Minterm_PI의 minterm 10, 11에 해당 minterm을 cover하는 '101-' 즉' PI를 담아줌.
        // 초기 ALL_Minterm_PI : {{0}, {4}, {8}, {10}, {11}, {12}}

        for(int p=1; p<PI_minterm.size(); p++){
            for(int q=0; q<ALL_Minterm_PI.size(); q++){
                if(PI_minterm[p] == ALL_Minterm_PI[q][0])
                    ALL_Minterm_PI[q].push_back(PI_minterm[0]);
            }
        }
        return ALL_Minterm_PI;
}


vector<string> Find_CD(vector<vector<string>> ALL_Minterm_PI){
    vector<string> EPI;
    vector<string> tmpEPI;
                // ALL_Minterm_PI : {{0, --00}, {4, --00}, {8, 10-0, --00},
                //                  {10, 101-, 10-0}, {11, 101-}, {12, --00}}
    for(int i=0; i<ALL_Minterm_PI.size(); i++){                    // 6번
        int isdominated = 0;
        if(ALL_Minterm_PI[i][0] != "X"){
            string tmp = "";
            string tmp2 = "";
            for(int j=1; j<ALL_Minterm_PI[i].size(); j++){          // 각 minterm의 PI들 하나씩 지목
                tmp = ALL_Minterm_PI[i][j];
            }
            for(int k=0; k<ALL_Minterm_PI.size(); k++){
                for(int s=1; s<ALL_Minterm_PI[k].size(); s++){
                    tmp2 = ALL_Minterm_PI[k][s];
                }
            }
            if(tmp == tmp2)
                isdominated++;
            if(isdominated>0){
                if(ALL_Minterm_PI[i][0] != "X")
                    tmpEPI.push_back(ALL_Minterm_PI[i][1]);
            }
```

```
            }
        }EPI.push_back(tmpEPI[0]);
        return EPI;
}

vector<string> Find_RD(vector<string> EPI, vector<vector<string>> ALL_PI_minterm){
        //EPI : --00
        //ALL_PI_minterm : {101-, 10, 11}, {10-0, 8, 10}, {--00, 0, 4, 8, 12}

        vector<string> vtmp;
        // EPI와 중복 minterm 제거
        for(int i=0; i<EPI.size(); i++){    // --00  1번
                for(int j=0; j<ALL_PI_minterm.size(); j++){  // 3번
                        if(EPI[i] == ALL_PI_minterm[j][0]){  //--00,0,4,8,12 벡터 뺌
                                ALL_PI_minterm[j].swap(vtmp);
                                ALL_PI_minterm.erase(ALL_PI_minterm.begin() + j);
                        }
                }
        }

        for(int i=0; i<ALL_PI_minterm.size(); i++){
                for(int j=1; j<ALL_PI_minterm[i].size(); j++){
                        for(int k=1; k<vtmp.size(); k++){
                                if(vtmp[k] == ALL_PI_minterm[i][j])
                                        ALL_PI_minterm[i].erase(ALL_PI_minterm[i].begin() + j);
                        }
                }
        }

        cout << endl << "< ALL_PI_minterm >" << endl;
        for(int p=0; p<ALL_PI_minterm.size(); p++){
                for(int q=0; q<ALL_PI_minterm[p].size(); q++)
                        cout << ALL_PI_minterm[p][q] << " ";
                cout << endl;
        }
        cout << endl;

        if((int)ALL_PI_minterm[0].size() > (int)ALL_PI_minterm[1].size()){
                EPI.push_back(ALL_PI_minterm[0][0]);
        }else{
                EPI.push_back(ALL_PI_minterm[1][0]);
        }



        return EPI;

}
```

```
int main(){
    vector<int> minterm = {4, 6,
                           0, 4, 8, 10, 11, 12};

    vector<string>answerPI;
    solution(minterm).swap(answerPI);
    cout << endl << "answer PI : ";
        for(int i=0; i<answerPI.size(); i++){
            cout << answerPI[i] << " ";
        }cout << endl ;
    cout << "ーーーーーーーーーーーーーーーーー" << endl;


    // 기존 answerPI를 각각 Minterm_PI와 PI_Minterm 2차원벡터로 변환

    vector<vector<string>> ALL_Minterm_PI;  // 2차원 벡터 AMP
    for(int i=2; i<minterm.size(); i++){
        vector<string> tmp_minterm_PI;
        tmp_minterm_PI.push_back(to_string(minterm[i]));
        ALL_Minterm_PI.push_back(tmp_minterm_PI);
        //ALL_Minterm_PI = {{0}, {4}, {8}, {10}, {11}, {12}}
    }cout << endl;

    vector<vector<string>> ALL_PI_Minterm;
    for(int i=0; i<answerPI.size(); i++){
        vector<string> PMtmp;
        Make_PI_minterm(answerPI[i]).swap(PMtmp);  // 101-으로 {101-,10, 11} 만듦

        Make_minterm_PI(ALL_Minterm_PI, PMtmp).swap(ALL_Minterm_PI);

        ALL_PI_Minterm.push_back(PMtmp);
    }


    // ALL_Minterm_PI 2차원 벡터 출력
    cout << endl << "< ALL_Minterm_PI >" << endl;
    for(int p=0; p<ALL_Minterm_PI.size(); p++){
        cout << ALL_Minterm_PI[p][0] << " { ";
        for(int q=1; q<ALL_Minterm_PI[p].size(); q++)
            cout << ALL_Minterm_PI[p][q] << " ";
        cout << "}" << endl;
    }       cout << endl;
    // ALL_PI_Minterm 2차원 벡터 출력
    cout << endl << "< ALL_PI_Minterm >" << endl;
    for(int i=0; i<ALL_PI_Minterm.size(); i++){  // 3번
        cout << ALL_PI_Minterm[i][0] << " { ";
```

```
        for(int j=1; j<ALL_PI_Minterm[i].size(); j++){
            cout << ALL_PI_Minterm[i][j] << " ";
        }cout << "}" << endl;
    }


    cout << "————————————————————" << endl;

    vector<string> EPI;

    Find_CD(ALL_Minterm_PI).swap(EPI);   cout << endl << "~~ (1)Column Dominance ~~" << endl;

            cout << "EPI : { ";
            for(int i=0; i<EPI.size(); i++) {cout << EPI[i] << " ";} cout << "}" << endl;
            ㅎ
    Find_RD(EPI, ALL_PI_Minterm).swap(EPI);   cout << endl << "~~ (2) Row Dominance ~~" << endl;

            cout << "EPI : { ";
            for(int i=0; i<EPI.size(); i++) {cout << EPI[i] << " ";} cout << "}" << endl;


    return 0;
}
```