CAD Lab1 0811562 何祁恩

1. Generate binary header

```cpp
#ifndef BINARY_ARRAY_H
#define BINARY_ARRAY_H

#include <string>
#include <vector>
using namespace std;

const vector<vector<string>> binary_strings = {
    {"0"},
    {"0", "1"},
    {"00", "01", "10", "11"},
    {"000", "001", "010", "011", "100", "101", "110", "111"},
    {"0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111", "1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111"},
    {"00000", "00001", "00010", "00011", "00100", "00101", "00110", "00111", "01000", "01001", "01010", "01011", "01100", "01101", "01
    {"000000", "000001", "000010", "000011", "000100", "000101", "000110", "000111", "001000", "001001", "001010", "001011", "001100",
    {"0000000", "0000001", "0000010", "0000011", "0000100", "0000101", "0000110", "0000111", "0001000", "0001001", "0001010", "0001011
    {"00000000", "00000001", "00000010", "00000011", "00000100", "00000101", "00000110", "00000111", "00001000", "00001001", "0000101
};

#endif // BINARY_ARRAY_H
```

Because there is a maximum limit of 8 variables, I precomputed all possible binary values beforehand. During program execution, it is only necessary to reference this table.

2. Parser

```
################## Parser Summary ####################
Num Vars: 4
ON Set:
         4 (0100)
         5 (0101)
         6 (0110)
         8 (1000)
         9 (1001)
        10 (1010)
        13 (1101)

DC Set:
         0 (0000)
         7 (0111)
        15 (1111)

Implicants:
       0000
       0100
       1000
       0101
       0110
       1001
       1010
       0111
       1101
       1111
################ End Parser Summary ##################
```

From the table above, read the on-set and don't-care set, and store them as binary.

3. Quine McCluskey

```
################## Prime Implicant ####################
Prime Implicants : 7
        -000
        -1-1
        0-00
        01--
        1-01
        10-0
        100-
################## Prime Implicant ####################
```

After applying the Quine-McCluskey algorithm, the prime implicants are obtained.

4. Row Reduction

$$P = (P1+P6)(P6+P7)P6(P2+P3+P4)(P3+P5)P4(P5+P7)=1$$

		4	5	6	8	9	10	13
P1	0,4 (0-00)	×						
P2	0,8 (-000)				×			
P3	8,9 (100-)				×	×		
P4	8,10 (10-0)				×		×	
P5	9,13 (1-01)					×		×
P6	4,5,6,7 (01- -)	×	×	×				
P7	5,7,13,15 (-1-1)		×					×

If at this moment, all the prime implicants are directly modeled as a POS SAT problem to be solved, it would require calculating so many possibilities of SOP, which is 2 * 2 * 1 * 3 * 2 * 1 * 2. If the number of variables increases, the possibilities of SOP will be even more. Therefore, essential prime implicants should be identified first, followed by finding out which of these essential prime implicants cannot cover the on-set. Then, all the prime implicants that have the potential to cover these uncovered on-sets should be identified. Using Petrick's Method thereafter would be more efficient. From the figure above, we can determine that the Essential Prime Implicants are P4 and P6, and the on-sets that these two Essential Prime Implicants cannot cover are 9 and 13. Therefore, the prime implicants that can cover 9 and 13 are deduced to be P3, P5, and P7. Consequently, it can be inferred that the simplified SAT problem becomes (P3+P5)(P5+P7), with only 2 * 2 possibilities, which is significantly faster compared to directly using the Petrick method. I refer to prime implicants such as P3, P5, and P7, which can cover the on-sets that essential prime implicants cannot cover, as candidate prime implicants.

```
################## Uncover ON Set ######################
Uncover ON Set:
         9 (1001)
        13 (1101)
################## Uncover ON Set ######################

################ Essential Prime Implicant #############
Essential Prime Implicants : 2
        01--
        10-0
################ Essential Prime Implicant #############

################ Candidate Prime Implicant #############
Candidate Prime Implicants : 3
        -1-1
        1-01
        100-
################ Candidate Prime Implicant #############

#################### Product Of Sum ####################
Product of Sum (POS):
        (1-01 + 100-)
        (-1-1 + 1-01)
#################### Product Of Sum ####################
```

5. Petrick's Method:

After obtaining the POS, multiplying it out yields all the feasible solutions for this SAT problem. However, if the POS expression is still extensive, expanding it back into SOP would still require substantial computational resources. Therefore, in this case, I utilized the branch and bound technique extensively. Throughout the expansion process, I continuously recorded the length of the shortest solution obtained earlier. If the current length exceeds this recorded value, I pruned that branch, ceasing any further computation. This was done because continuing down that path would not yield a better solution.

```
#################### Product Of Sum ##################
Product of Sum (POS):
        (1-01 + 100-)
        (-1-1 + 1-01)
#################### Product Of Sum ##################

#################### Sum Of Product ##################
Sum of Product (SOP):
        (-1-1 * 1-01)
        (1-01)
        (-1-1 * 100-)
        (1-01 * 100-)
#################### Sum Of Product ##################
```

6. Find Minimum Cover

After obtaining the SOP, the next step is to calculate which one has fewer literals. Sort them according to the requirements of the problem and then output the results.