University of Windsor
Faculty of Science
School of Computer Science

COMP-2650
Computer Architecture I: Digital Design
Fall 2024

Lab#	Date	Title	Due Date	Grade Release Date
Lab 9	Week 9	**Canonical SoP**	Nov. 12, Tuesday Midnight	Nov. 18

Step 1. Environment Setup

Our programming environment is the same as the first lab (Lab 01). In this lab, we want to continue the new series of labs about designing a logic circuit. Particularly, in this lab, we want to write the boolean function (expression) for the output binary variables based on the standard form of the sum of minterms. Sum of minterms is also called Canonical Sum of Products (SoP) since each minterm is an AND between the input binary variables (either in normal form X or in complement form X'), e.g., Z'YX', followed by an OR on the minterms, e.g., F(Z,Y,X) = $m_0+m_2+m_3$ = Z'Y'X' + Z'YX' + Z'YX.

1) As we discussed in the lectures, the first step in designing a logic circuit is to build a truth table with columns for input binary variables and columns for output binary variables. Also, we have to create rows for different values of the input binary variables, either 0 or 1 for each input binary variable. For example, given 3 input binary variables and 1 output binary variable, the truth table would have 4 columns and 2^3=8 rows.

2) Next, we have to pick names for the input and output binary variables. For instance, for 3 input binary variables, we can choose Z, Y, X and for the single output binary variable we can choose F.

3) Then, we have to look at those rows that make the output binary variable 1 and write the output binary variable as a Boolean function (expression) of the input binary variables in form of a sum of minterms (canonical sum of products). For instance, F = $\sum$m(0,2,3) = Z'Y'X' + Z'YX' + Z'YX.

4) Finally, we sketch the logic circuit using the schematic symbols of the NOT, AND, and OR logic gates.

In the previous Lab 08, we wrote a program that does the 1st and 2nd steps. That is, we built a program that outputs the truth table by, first, building the left side of the truth table for input binary variables and, then, the right side of the truth table for the output binary variables. On the left side, we had to increment the binary representations of the input binary variables to produce all the different combination of the input binary variables:

```
//from previous Lab 08:
void build_left_side(int truth_table[][INPUT_VARIABLE_COUNT + OUTPUT_VARIABLE_COUNT]){...}
```

For the right side of the truth table, we asked the user for the value of the output binary variable (`'F'`):

```
//from previous Lab 08:
void build_right_side(int truth_table[][INPUT_VARIABLE_COUNT + OUTPUT_VARIABLE_COUNT]){ ...}
```

In the following code, I assume that there are 3 input binary variables (line#04), there are 1 output binary variables (line#05), and as a result, the truth table is going to have $2^{\#\text{input variables}} = 2^3 = 8$.

```
01 #include <stdio.h>
02 #include <math.h>
03
04 #define INPUT_VARIABLE_COUNT 3
05 #define OUTPUT_VARIABLE_COUNT 1
06
07 void build_right_side(int truth_table[][INPUT_VARIABLE_COUNT + OUTPUT_VARIABLE_COUNT]){...}
08 void build_right_side(int truth_table[][INPUT_VARIABLE_COUNT + OUTPUT_VARIABLE_COUNT]){...}
09
10 int main(void) {
11      setbuf(stdout, NULL);
12
```

```
13      int TRUTH_TABLE_ROW_COUNT = (int)pow(2, INPUT_VARIABLE_COUNT);
14      int truth_table[TRUTH_TABLE_ROW_COUNT][INPUT_VARIABLE_COUNT + OUTPUT_VARIABLE_COUNT] = {0};
15      const char variables[INPUT_VARIABLE_COUNT + OUTPUT_VARIABLE_COUNT] =  {'Z', 'Y', 'X', 'F'};
16
17      build_left_side(truth_table);
18      build_right_side(truth_table);
19
20      //printing the header for input variables
21      for(int i = 0; i < INPUT_VARIABLE_COUNT; i = i + 1){
22              printf("%c, ", variables[i]);
23      }
24      printf(" : ");
25
26      //printing the header for output variables
27      for(int i = INPUT_VARIABLE_COUNT; i < INPUT_VARIABLE_COUNT + OUTPUT_VARIABLE_COUNT; i = i + 1){
28              printf("%c", variables[i]);
29      }
30      printf("\n");
31
32      //printing the content of each row
33      for(int i = 0; i < TRUTH_TABLE_ROW_COUNT; i = i + 1){
34
35              //printing the content of each row regarding the input variables
36              for(int j = 0; j < INPUT_VARIABLE_COUNT; j = j + 1){
37                      printf("%d, ", truth_table[i][j]);
38              }
39              printf(" : ");
40
41              //printing the content of each row regarding the output variables
42              for(int j = INPUT_VARIABLE_COUNT; j < INPUT_VARIABLE_COUNT + OUTPUT_VARIABLE_COUNT; j = j + 1){
43                      printf("%d", truth_table[i][j]);
44              }
45              printf("\n");
46      }
47      return 0;
```

A sample run would look like the following then:

```
output value for row# 0 of F1 output variable:1
output value for row# 1 of F1 output variable:0
output value for row# 2 of F1 output variable:0
output value for row# 3 of F1 output variable:0
output value for row# 4 of F1 output variable:1
output value for row# 5 of F1 output variable:1
output value for row# 6 of F1 output variable:0
output value for row# 7 of F1 output variable:0

Z, Y, X,  : F
0, 0, 0,  : 1
0, 0, 1,  : 0
0, 1, 0,  : 0
0, 1, 1,  : 0
1, 0, 0,  : 1
1, 0, 1,  : 1
1, 1, 0,  : 0
1, 1, 1,  : 0
```

Now, in this lab, we want to complete the program to do the 3rd step of the design procedure, that is printing out the Boolean function in the form of a sum of minterms (Canonical Sum of Products). To do so, in a loop on rows, wherever we see 1 in the last column of the truth table, we print out the AND of the input variables based on whether they are 0 (complement form X') or 1 (normal form X). We can write a function to do so and put it after printing out the truth table at line#47 of the above program:

```
void to_minterm(int truth_table[][INPUT_VARIABLE_COUNT + OUTPUT_VARIABLE_COUNT]){
        for(int j = 0; j < OUTPUT_VARIABLE_COUNT; j = j + 1){
                printf("output variable F%d = ", j+1);
                for(int i = 0; i < TRUTH_TABLE_ROW_COUNT; i = i + 1){
                        //to be completed!
                }
                printf("\n");
```

```
    }
}
```

A sample run would be:

```
output value for row# 0 of F1 output variable:1
output value for row# 1 of F1 output variable:0
output value for row# 2 of F1 output variable:0
output value for row# 3 of F1 output variable:0
output value for row# 4 of F1 output variable:1
output value for row# 5 of F1 output variable:1
output value for row# 6 of F1 output variable:0
output value for row# 7 of F1 output variable:0
Z, Y, X,  : F
0, 0, 0,  : 1
0, 0, 1,  : 0
0, 1, 0,  : 0
0, 1, 1,  : 0
1, 0, 0,  : 1
1, 0, 1,  : 1
1, 1, 0,  : 0
1, 1, 1,  : 0
output variable F1 = Z'Y'X'+ZY'X'+ZY'X+
```

As seen, the Boolean function for the only output variable F1 is printed out in the form of the Canonical Sum of Products. We can *optionally* print out the minterm numbers, e.g., in this example we could print out:

```
output variable F1 = Σm(0,4,5)= Z'Y'X'+ZY'X'+ZY'X+
```

Lab Assignment

You should complete the above program that outputs a menu of commands as follows:

```
Enter the command number:
0) Exit
1) Canonical SoP
```

If a user selects (1), the program asks for the value of output variable F1 as follows:

```
output value for row# 0 of F output variable:1
output value for row# 1 of F output variable:0
output value for row# 2 of F output variable:0
output value for row# 3 of F output variable:0
output value for row# 4 of F output variable:1
output value for row# 5 of F output variable:1
output value for row# 6 of F output variable:0
output value for row# 7 of F output variable:0
```

When the user enters the values, the program should print out the truth as shown below:

```
Z, Y, X,  : F
0, 0, 0,  : 1
0, 0, 1,  : 0
0, 1, 0,  : 0
0, 1, 1,  : 0
1, 0, 0,  : 1
1, 0, 1,  : 1
1, 1, 0,  : 0
1, 1, 1,  : 0
```

the program should print out the Boolean function for F1 in the form of a sum of minterms (Canonical SoP) as shown below:

```
output variable F1 = Z'Y'X'+ZY'X'+ZY'X+
```

If the user selects (0), the program ends. Please restrict the user to enter inputs within the range {0,1} for the value of the output variable. For instance, if the user enters 2, -1, ..., print out an error message and come back to ask for correct inputs.

It is required to write a *modular* program. Please put the part of the code that outputs a minterm based on the value of input variables in a new function called `to_minterm()` inside the `main.c` file.

Deliverables
Prepare and submit the program in one single zip file lab10_uwinid.zip containing the following items:

1. The code file (main.c or main.cpp) and executable file (main.exe in Windows or main in Unix/macOS)
2. The result of the four commands in the file results.pdf/png/jpg. Simply make a screenshot of the results.
3. *[Optional and if necessary]* A readme document in a txt file readme.txt. It explains how to build and run the program as well as any prerequisites that are needed. Please note that if your program cannot be built and run on our computer systems, you will lose marks.

`Lab10_hfani.zip`
- (70%) `main.c` => (10%) Left & Right Side of Truth Table, (60%) Printing Canonical SoP
- (05%) `main.exe or main`
- (10%) `results.pdf/jpg/png` → Must match with the program output!
- (Optional) `readme.txt`

(10%) `Modular Programming (using separate header and source files and functions)`
(05%) `Files Naming and Formats`

Please follow the naming convention as you lose marks otherwise. Instead of uwinid, use your own UWindsor account name, e.g., mine is hfani@uwindsor.ca, so, lab10_hfani.zip