



**School of Computer Science  
Faculty of Science**

**COMP-2650: Computer Architecture I: Digital Design  
Winter 2021**

Lab#	Date	Title	Due Date	Grade Release Date
Lab 08	Week 08	Truth Table	March 02, 2021 Tuesday Midnight AoE Wednesday 7 AM EDT	March 08, 2021

This lab's objectives will be to master the topics in logic circuit design by implementing the algorithms with a programming language, herein, C/C++.

**Step 1. Environment Setup**

Our programming environment is the same as the first lab (Lab 01). In this lab, we want to start a new series of labs about designing a logic circuit. Particularly, in this lab, we want to create a truth table for a given number of input binary variables and output binary variables.

- 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 the 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 this lab, we want to write a program that does the 1<sup>st</sup> and 2<sup>nd</sup> steps. That is, we want to write a program that outputs the truth table. In the following code, I assume that there are 3 input binary variables (line#04), there is 1 output binary variable (line#05), and as a result, the truth table is going to have  $2^{(\text{\#input variables})} = 2^3=8$ .

I defined the truth table as a 2-D array of integer values with size 8 rows  $\times$  4 columns (line#15). Please pay attention that in C/C++, the '^' symbol is reserved for the bitwise XOR operator and cannot be used for the power operator (line#11,12). In C/C++, we can use the pow(a, b) function available in math.h library to return a to the power of b as shown in line#14. Also, note that the format specifier for char is "%c".

```
01 #include <stdio.h>
02 #include <math.h>
03
04 #define INPUT_VARIABLE_COUNT 3
05 #define OUTPUT_VARIABLE_COUNT 1
```

```

06
07 int main(void) {
08
09     setbuf(stdout, NULL);
10
11     //Wrong! ^ operator in C/C++ is the bitwise XOR logic operator.
12     //int TRUTH_TABLE_ROW_COUNT = 2^INPUT_VARIABLE_COUNT;
13
14     int TRUTH_TABLE_ROW_COUNT = (int)pow(2, INPUT_VARIABLE_COUNT);
15     int truth_table[TRUTH_TABLE_ROW_COUNT][INPUT_VARIABLE_COUNT + OUTPUT_VARIABLE_COUNT] = {0};
16     const char variables[INPUT_VARIABLE_COUNT + OUTPUT_VARIABLE_COUNT] = {'Z', 'Y', 'X', 'F'};
17
18     //printing the header of truth table with variable names for inputs and outputs
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;
48 }

```

**Commented [A1]:** Depending on the C/C++ compiler, some allows us to initialize all the elements of the matrix to 0 by this. However, some does not allow this. So, you have to write for loops to initialize the elements. Please look at the discussion board for a possible solution, proposed by students.

In the above code, first, we output the header of the truth table (line#18-30) given the names of variables are defined in an array of chars (line#16). Then, we output the content of the truth table. If you run the code you would see the following result:

```

Z, Y, X,   : F
0, 0, 0,   : 0
0, 0, 0,   : 0
0, 0, 0,   : 0
0, 0, 0,   : 0
0, 0, 0,   : 0
0, 0, 0,   : 0
0, 0, 0,   : 0
0, 0, 0,   : 0
0, 0, 0,   : 0

```

As you can see, the code has two problems: *i)* it outputs only one possibility in the input binary variables: Z=0, Y=0, X=0, *ii)* it does not ask or determine where the output binary variable should be 1.

To fix the previous code, first, I have to create all the possibilities of the input binary variables on the left side of the truth table such that the output looks like the following:

```

Z, Y, X,   : F
0, 0, 0,   : 0

```



```
0, 0, 1, : 0
0, 1, 0, : 0
0, 1, 1, : 0
1, 0, 0, : 0
1, 0, 1, : 0
1, 1, 0, : 0
1, 1, 1, : 0
```

As seen, if I put the different possibilities of the input variables in increasing order of binary numbers, they look like incrementing the previous possibility, that is,  $000 \rightarrow 001 \rightarrow \dots \rightarrow 110 \rightarrow 111$ . I can use either the signed-magnitude addition or the signed-2's-complement addition functions in `arithmetic.h` that I wrote in Lab04 and Lab05 to do an increment. Or I can add a new function that does an increment to a given unsigned binary:

```
arithmetic.h
void func_increment(int a[], int result[]);

arithmetic.cpp
#define MAX 8 //Byte = 8 bits
void func_increment(int a[], int result[]){...}
```

**Commented [A2]:** In this lab, we only need to increment from 0 to 7. So, we only need 3 bits indeed.

Then, I can change my code to fix the input binary variable part of the truth table.

```
void build_left_side(int truth_table[]){
    for(int i = 0; i < TRUTH_TABLE_ROW_COUNT - 1; i = i + 1){

        int row[INPUT_VARIABLE_COUNT] = {0};
        int result[INPUT_VARIABLE_COUNT] = {0};

        //accessing the elements of the i-th row
        ...

        //increment
        func_increment(row, result);

        //put into the next row: (i+1)-th row
        ...

    }
}
```

Regarding the second problem, we can ask the user for the value of the output binary variable ('F').

```
void build_right_side(int truth_table[]){
    for(int i = 0; i < TRUTH_TABLE_ROW_COUNT; i = i + 1){

        //for each output variable F, ...
        for(int j = 0; j < OUTPUT_VARIABLE_COUNT; j = j + 1){
            printf("output value for row# %d of %c output variable:", i, ...);
            ...
        }

    }
}
```

A sample run would look like the following then:

```
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
```

```
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
```

### Lab Assignment

You should complete the above program under the name of “Project” that 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
```

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:

- 1) For increment, you can re-use the function in [arithmetic.h](#) or write a new function called `func_increment()`.
- 2) Put the part of the code that completes the left part of the truth table in a new function called `build_left_side()` inside the [main.c](#) file.
- 3) Put the part of the code that asks for the values of the output variable in a new function called `build_right_side()` inside the [main.c](#) file.

### Deliverables



You will prepare and submit the program in one single zip file `Lab08_UWinID.zip` containing the following two items:

1. The entire project folder `Project` including the code file (`main.c` or `main.cpp`) and executable file (`main.exe` in windows or `main` in mac)
2. The result of the commands in the file `Results.pdf`. Simply take screenshots of the results and save (print) them **into a single pdf**.
2. [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.***

In sum, your final `Lab08_UWinID.zip` file for the submission includes 1 folder (entire project folder), 1 image (results snapshot) and 1 txt (report). ***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,

`Lab08_hfani.zip`

- (75%) `Project` =>(45%) Left Side of Truth Table, (30%) Right Side of the Truth Table
  - o [any required library, header or source files]
  - o `main.c` or `main.cpp` => Must be compiled and built with no error!
  - o `main.exe` or `main`
- (10%) `Results.pdf`
- (Optional) `ReadMe.txt`

(10%) Modular Programming (using separate functions)

(5%) Files Naming and Formats and Folder Structure