

**Khulna University of Engineering & Technology (KUET)**

**Department of Electronics and Communication Engineering**

|  |
| --- |
| *Project Report* |

**Course No**: *ECE-2104*

**Name of the Experiment:** *To determine truth table, simplified Boolean expression from truth table, Verilog program of this expression and obtain the timing diagram from the given product of sum (POS) form of the given Boolean expression*

**Submitted by:**

**Name:** *Md. Abu Hanif Khan*

**Roll No:** *2009054*

Date of Submission: 08/05/2023 **Year:** *2nd*

**Under The Guidance of:**

Dr. Pallab Kumar Choudhury Naymur Rahman

Professor Lecturer

Department of ECE, KUET Department of ECE, KUET

****

***Table of contents:***

|  |  |  |
| --- | --- | --- |
| **Serial No.** | **Topics Name** | **Page No** |
| 01 | Objectives | 03 |
| 02 | Introduction | 03 |
| 03 | Theory | 03 |
| 04 | Program | 14 |
| 05 | Result Analysis | 20 |
| 06 | Discussion | 24 |
| 07 | Conclusion | 24 |
| 08 | Reference | 25 |

***Objectives:***

After we have finished this project, we will be able -

1. To know about POS and SOP form.
2. To learn how to simplify a Boolean expression.
3. To learn about k-map and how to implement it to a Boolean expression.
4. To learn about Verilog HDL and determine the Verilog program (in gate level, data flow and behavioral modelling) for the simplified expression.
5. To write down test bench and timing diagram of this expression.

***Introduction:***

A Boolean expression is a logical statement that is either TRUE or FALSE. In 1854 George Boole introduced a systemic treatment of logic and developed for this purpose an algebraic system now called Boolean algebra.

The truth table of a logic system (used in digital electronic circuit) describes the output(s) of the system for given input(s).

A Karnaugh map or a K-map refers to a pictorial method that is utilized to minimize various Boolean expressions without using the Boolean algebra theorems along with the equation manipulations. A Karnaugh map can be a special version of the truth table. Karnaugh maps reduced logic functions more quickly and easily compared to Boolean algebra. By reduce we mean simplify, reducing the number of gates and inputs.

Verilog is a HARDWARE DESCRIPTION LANGUAGE (HDL). It is a language used for describing a digital system like a network switch or a microprocessor or a memory or a flip−flop. It means, by using HDL we can describe any digital hardware at any level. Designs, which are described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits. We can use Verilog HDL for designing hardware and for creating test entities to verify the behavior of a piece of hardware.

Verilog supports a design at many levels of abstraction. The major three are −

* Behavioral level
* Register-transfer level
* Gate level

***Theory:***

***Truth table:*** A truth table is a graphical representation of all possible combinations of input values and their corresponding output values in a logical expression. It is commonly used in logic and computer science to analyze and evaluate the behavior of logical operators and circuits.

Let’s take an example of a simple logical expression with two input variables, A and B, and one output variable, C. The expression is:

C=A AND B

Here, “AND” is a logical operator that returns true (1) only if both input values are true (1), and false (0) otherwise. The truth table for this expression would be as follows:

|  |  |  |
| --- | --- | --- |
| A | B | C |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

In this truth table, we list all possible combinations of input values for A and B, and calculate the corresponding output value for C based on the logical operation of AND. For example, when A=0 and B=0, the output C is 0, as both A and B are false (0). Similarly, when A=1 and B=1, the output C is 1, as both A and B are true (1). The truth table helps us understand the behavior of the logical expression for all possible input combinations.

***SOP form:*** The sum-of-products (SOP) form is a way of representing Boolean functions using a standard format. It involves expressing the function as a logical OR of logical ANDs, where the ANDs are product terms and the ORs are sum terms.

A sum-of-products form can be formed by adding (or summing) two or more product terms using a Boolean addition operation. Here the product terms are defined by using the AND operation and the sum term is defined by using OR operation.

In summary, SOP form is a powerful tool for simplifying and representing Boolean functions in a standardized way, which can make it easier to implement them in hardware circuits

**Examples**

F=AB + ABC + CDE

F=(AB) + ABC + CD

To obtain the sum-of-products (SOP) form of a Boolean function, you can follow these steps:

Create an AND term for each input combination that results in a HIGH output.

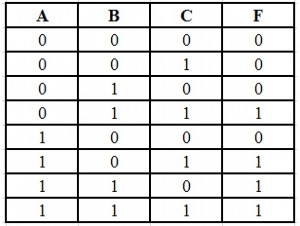
For each input variable, include the variable itself in the term if its value is 1, and include its complement if its value is 0.

Combine all of the resulting AND terms using the OR operation to get the final output function in SOP form.

For example, let's consider the Boolean expression for the majority function:

F = A’BC + AB’C + ABC + ABC

Truth table:

[](https://www.electronicshub.org/wp-content/uploads/2015/08/table2.jpg)

***POS form:*** The product-of-sums (POS) form is a method of simplifying Boolean expressions used in logic gates. In the POS form, all of the variables are written as OR terms, which are added together to form sum terms. All of these sum terms are then multiplied together using the AND operation to get the final product-of-sums form. This is the opposite of the SOP form, and can also be thought of as the dual of SOP.

**Examples**

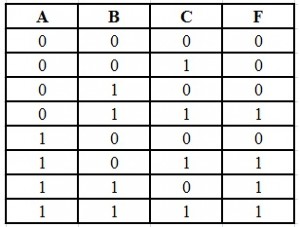
(A+B) \* (A + B + C) \* (C +D)

(A+B) ̅ \* (C + D + E ̅)

POS form can be obtained by

* Writing an OR term for each input combination, which produces LOW output.
* Writing the input variables if the value is 0, and write the complement of the variable if its value is 1.
* AND the OR terms to obtain the output function.

Ex: Boolean expression for majority function F = (A + B + C) (A + B + C ‘) (A + B’ + C) (A’ + B + C)

[](https://www.electronicshub.org/wp-content/uploads/2015/08/EX.jpg)

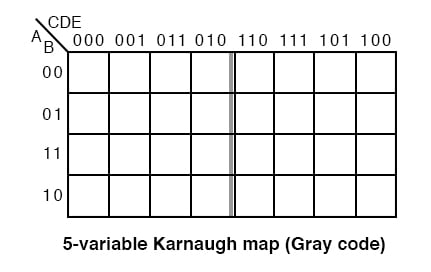
***K-map:*** Karnaugh map or K-map is a map of a function used in a technique used for minimization or simplification of a Boolean expression. It results in less number of logic gates and inputs to be used during the fabrication.

Boolean expression can be simplified using Boolean algebraic theorems but there are no specific rules to make the most simplified expression. However, K-map can easily minimize the terms of a Boolean function.Unlike an algebraic method, K-map is a pictorial method and it does not need any Boolean algebraic theorems.

K-map is basically a diagram made up of squares. Each of these squares represents a min-term of the variables. If n = number of variables then the number of squares in its K-map will be 2n. K-map is made using the truth table. In fact, it is a special form of the truth table that is folded upon itself like a sphere. Every two adjacent squares of the k-map have a difference of 1-bit including the corners.

Karnaugh map can produce Sum of product (SOP) or product of Sum (POS) expression considering which of the two (0,1) outputs are being grouped in it. The grouping of 0’s result in Product of Sum expression & the grouping of 1’s result in Sum of Product expression. The expression produced by K-map may be the most simplified expression but not unique. There can be more than 1 simplified expression for a single function but they all perform the same

#### Five Variable K-map



The older version of the five variable K-map, a Gray Code map or reflection map, is shown above. The top (and side for a 6-variable map) of the map is numbered in full Gray code

Simplifying (POS) form using a K-map, we can follow these steps:

1. Creating the truth table for the function with 5 input variables and 1 output variable.
2. Identify the maxterms for the function. These are the terms where the output is 0. We can find them by examining the truth table and writing down the input combinations that result in a 0 output.
3. Creating a K-map with 5 input variables, and mark the cells that correspond to the maxterm as 0’s.
4. Identifying groups of adjacent 0's in the K-map that can be combined to form sum terms of the function. Each group should contain a power of 2 number of cells (1, 2, 4, 8, or 16), and each cell in the group should be adjacent to at least one other cell in the group.
5. Writing down the sum terms that correspond to the groups of adjacent 0's, and combining them using the AND operator to form the product-of-sums (POS) form of the function.
6. Use De Morgan's laws and Boolean algebra rules to simplify the POS form of the function, if possible.
7. Finally, writing the simplified function in POS form.

***Verilog HDL:***

Verilog is a hardware description language (HDL) used to model and design digital circuits and systems. Verilog allows designers to describe the behavior and structure of a digital system using a textual representation, which can then be used to synthesize and simulate the system. Verilog is widely used in the design of digital circuits and systems.

Verilog is a Hardware Description Language

* Used to describe digital system
* Described designs are independent of technology
* Easier to design and debug

***Types of modelling:***

1. Switch Level Modelling
2. Gate Level Modelling
3. Data Flow Modelling
4. Behavioral Modelling

***Gate Level Modelling:***

 Verilog supports a few basic logic gates known as primitives.

 Predefined modules which can be instantiated in the code if needed.

 For example, not, and, NAND, OR, NOR, XOR, and XNOR.

***Data flow modelling:***

 Uses several operators that act on operands to produce the desired results.

 Describes hardware in terms of the flow of data from input to output.

 Continuous assignment is used to drive a value to a net or wire.

 A continuous assignment statement is represented by an ‘assign’ statement.

assign [delay] LHS-net = RHS-expression

***Behavioral modelling:***

 Describes the behavior of the circuit.

 The highest level of abstraction.

 always and initial block are used in behavioral modeling

***Structure of Verilog code:***

module name \_ of \_ the \_module (port1, port2);

input port1, port2, ……; //declare input ports

output port5, port6, port7, …; //declare output ports

…………………….................... // statements

………………………………….. // statements

end module

***Data Type in Verilog:***

It has several data types that are used to represent different types of values.

**Wire:** It is a type of net that connects various components in a circuit. A wire can be used to represent a signal or a bus.

**Reg:** It is a type of storage element that stores a single value. It can be used to store values between clock cycles.

**Integer:** It is a data type that represents signed or unsigned integers.

**Real:** It is a data type that represents real numbers.

**Time:** It is a data type that represents time in Verilog simulations.

**Parameter:** It is a data type used to define constants that can be used throughout the design.

**Enum:** It is a data type used to define enumerated data types.

**Array**: It is a data type used to store a collection of elements of the same type.

**Struct:** It is a data type used to group related variables together.

**Union**: It is a data type used to define a type that can hold one of several different types of data.

***VERILOG CONDITIONAL STATEMENTS using case:***

***case***

 Use of if-else may is inconvenient if number of conditions to be checked is quite high.

 The case statement checks if the given expression matches one among the other expressions inside the list and branches.

 A case statement begins with the case keyword and closes with the endcase keyword.

case (<expression>)

case\_item1;

case\_item2;

case\_item3;

case\_item4;

begin

//multiple statement for case 4

end

default:

endcase

***Verilog always block:***

¬ An always block always executes (free running process).

¬ Statements inside an always block are executed sequentially.

always @ (event)

begin [multiple statements]

end

¬ The symbol @, it means that the block will only be executed under the conditions stated in the parenthesis.

¬ The statements inside the always block will be executed whenever the variables in the parenthesis after the @ symbol changes.

***Problem:***

Write down the truth table of the given product of sum (POS) form of the given Boolean expression, and then find out its simplified Boolean expression using K-map. After these, write down three separate Verilog design programs for the simplified Boolean expression in gate level, data flow level, and behavioral level modeling (use the "case" statement) respectively). Write Verilog testbench programs to determine the truth table and to obtain the timing diagram for your designed system.

**Boolean Expression:** F (𝑎, 𝑏, 𝑐, 𝑑, 𝑔) = 𝛱 (2,4,7,13,19,24,26)

**Solution:** This is an expression of 5 variables and the expression is given in POS form. We can find the position from the truth table and can simplify the expression using k-map.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | b | c | d | g | F |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

**Simplification using K-**map:**

**Solution:**

Maxterm=IIM(2,4,7,13,19,24,26)

Variable=a,b,c,d,g

Group1 Group2 Group3 Group4 Group5 Group6

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| c d g  a b | c+ d+ g  000 | c+ d+ g’  001 | c+ d’+ g’  011 | c+ d’+ g  010 | c’+ d’+ g  110 | c’+d’+ g’  111 | c’+ d+ g’  101 | c’+ d+ g  100 |
| a+ b  00 | 1  0 | 1  1 | 1  3 | 0  2 | 1  6 | 0  7 | 1  5 | 0  4 |
| a+ b’  01 | 1  8 | 1  9 | 1  11 | 1  10 | 1  14 | 1  15 | 0  13 | 1  12 |
| a’+ b’  11 | 0  24 | 1  25 | 1  27 | 0  26 | 1  30 | 1  31 | 1  29 | 1  28 |
| a’+ b  10 | 1  16 | 1  17 | 0  19 | 1  18 | 1  22 | 1  23 | 1  21 | 1  20 |

**Group-1: 2 Cell Grouping (24,26)**

Positions = 24,26  
Simplified Expression = a'+b'+c+g

**Group-2: 1 Cell Grouping (19)**

Positions = 19  
Simplified Expression = (a'+b+c+d'+g')

**Group-3 : 1 Cell Grouping (2)**

Positions = 2  
 Simplified Expression = (a+b+c+d'+g)

**Group-4 : 1 Cell Grouping (7)**

Positions = 7  
Simplified Expression = (a+b+c'+d'+g')

**Group-5: 1 Cell Grouping (13)**

Positions = 13  
Simplified Expression = a+b'+c'+d+g')

**Group-6 : 1 Cell Grouping (4)**

Positions = 4  
Simplified Expression = (a+b+c'+d+g)

**Simplified Expression** = (a'+b'+c+g) \* (a'+b+c+d'+g') \* (a+b+c+d'+g) \* (a+b+c'+d'+g') \* (a+b'+c'+d+g') \* (a+b+c'+d+g)

***Circuit Diagram of the simplified form:***

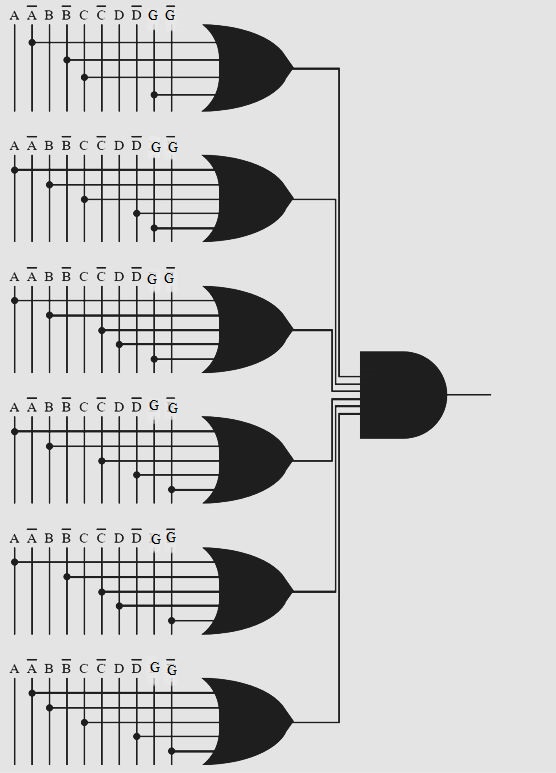


Fig-01: Circuit diagram of the Boolean expression

* Problem in gate level, data flow and behavioral modelling

***Code for Gate level Modelling:***

//gate level modelling of maxterm

module gate\_level(A,B,C,D,G,F);

input A,B,C,D,G; //input declaration

output F; //output declaration

wire a,b,c,d,g,w1,w2,w3,w4,w5,w6;

not(a,A); //logical not gate operation

not(b,B); //logical not gate operation

not(c,C); //logical not gate operation

not(d,D); //logical not gate operation

not(g,G); //logical not gate operation

or(w1,a,b,C,G); //logical or gate operation of(a,b,C,G) stored in w1

or(w2,a,B,C,d,g); //logical or gate operation of(a,B,C,d,g) stored in w2

or(w3,A,B,C,d,G); //logical or gate operation of(A,B,C,d,G) stored in w3

or(w4,A,B,c,d,g); //logical or gate operation of(A,B,c,d,g) stored in w4

or(w5,A,b,c,D,g); //logical or gate operation of(A,b,c,D,g) stored in w5

or(w6,A,B,c,D,G); //logical or gate operation of(A,B,c,D,G) stored in w6

and(F,w1,w2,w3,w4,w5,w6); //logical and gate operation of(w1,w2,w3,w4,w5,w6) stored in F

endmodule

***Code for Behavioral Modelling:***

//behavioral model

module behavioral\_model(A,B,C,D,G,F);

input A,B,C,D,G;

output F;

wire A,B,C,D,G;

reg F;

always @\* // An always block always executes

begin //The symbol means that the block will only be executed under the conditions stated in the parenthesis.

case ({A,B,C,D,G})

//case statement checks if the given expression matches one among the other expressions inside the list and branches.

5'b00010,

5'b00100, //it's a 5bit register,b is used for binary

5'b00111,

5'b01101,

5'b10011,

5'b11000,

5'b11010: F = 0;

//Max\_term 2,4,7,13,19,24,26

default: F = 1; //set default case as1

endcase

end

endmodule

***Code for Data flow Modelling:***

//DATA flow modelling

module data\_flow(A,B,C,D,G,F);

input A,B,C,D,G; //input declaration

output F; //output declaration

wire w1,w2,w3,w4,w5,w6;

assign w1=~A|~B|C|G; // OR(|), here w1 is assigned to (A'+B'+C+G)

assign w2=~A|B|C|~D|~G; // OR(|), here w2 is assigned to (A'+B+C+D'+G')

assign w3=A|B|C|~D|G; // OR(|), here w3 is assigned to (A+B+C+D'+G)

assign w4=A|B|~C|~D|~G; // OR(|), here w4 is assigned to (A+B+C'+D'+G')

assign w5=A|~B|~C|D|~G; // OR(|), here w5 is assigned to (A+B'+C'+D+G')

assign w6=A|B|~C|D|G; // OR(|) here w6 is assigned to (A+B+C'+D+G)

assign F=w1 & w2 & w3 & w4 & w5 & w6; // AND(&),F is assigned to and operation of ( w1 w2 w3 w4 w5)

endmodule

***Code for Test bench:***

module test\_besnch( );

reg a, b, c, d, g;

wire F;

behavioral\_model b1(a,b,c,d,g,f);//lingking statement

initial begin

$dumpfile("dump.vcd");

$dumpvars(1);

a = 0; b = 0; c = 0; d = 0; g = 0;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f); //display keyword shows the all variables in the console window

a = 0; b = 0; c = 0; d = 0; g = 1;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 0; b = 0; c = 0; d = 1; g = 0;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 0; b = 0; c = 0; d = 1; g = 1;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 0; b = 0; c = 1; d = 0; g = 0;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 0; b = 0; c = 1; d = 0; g = 1;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 0; b = 0; c = 1; d = 1; g = 0;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 0; b = 0; c = 1; d = 1; g = 1;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 0; b = 1; c = 0; d = 0; g = 0;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 0; b = 1; c = 0; d = 0; g = 1;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 0; b = 1; c = 0; d = 1; g = 0;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 0; b = 1; c = 0; d = 1; g = 1;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 0; b = 1; c = 1; d = 0; g = 0;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 0; b = 1; c = 1; d = 0; g = 1;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 0; b = 1; c = 1; d = 1; g = 0;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 0; b = 1; c = 1; d = 1; g = 1;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 1; b = 0;c = 0; d = 0; g = 0;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 1; b = 0; c = 0; d = 0; g = 1;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 1; b = 0; c = 0;d = 1; g = 0;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 1; b = 0; c = 0; d = 1; g = 1;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 1; b = 0; c = 1; d = 0; g = 0;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 1; b = 0; c = 1; d = 0; g = 1;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 1; b = 0; c = 1; d = 1; g = 0;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 1; b = 0; c = 1; d = 1; g = 1;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 1; b = 1; c = 0; d = 0;g = 0;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 1; b = 1; c = 0; d = 0; g = 1;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 1; b = 1; c = 0; d = 1; g = 0;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 1; b = 1; c = 0; d = 1; g = 1;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 1; b = 1; c = 1; d = 0; g = 0;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 1; b = 1; c = 1; d = 0; g = 1;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 1; b = 1; c = 1; d = 1; g = 0;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

a = 1; b = 1; c = 1; d = 1; g = 1;

#100; $display("a=%b,b=%b,c=%b,d=%b,g=%b,F=%b",a,b,c,d,g,f);

end

endmodule

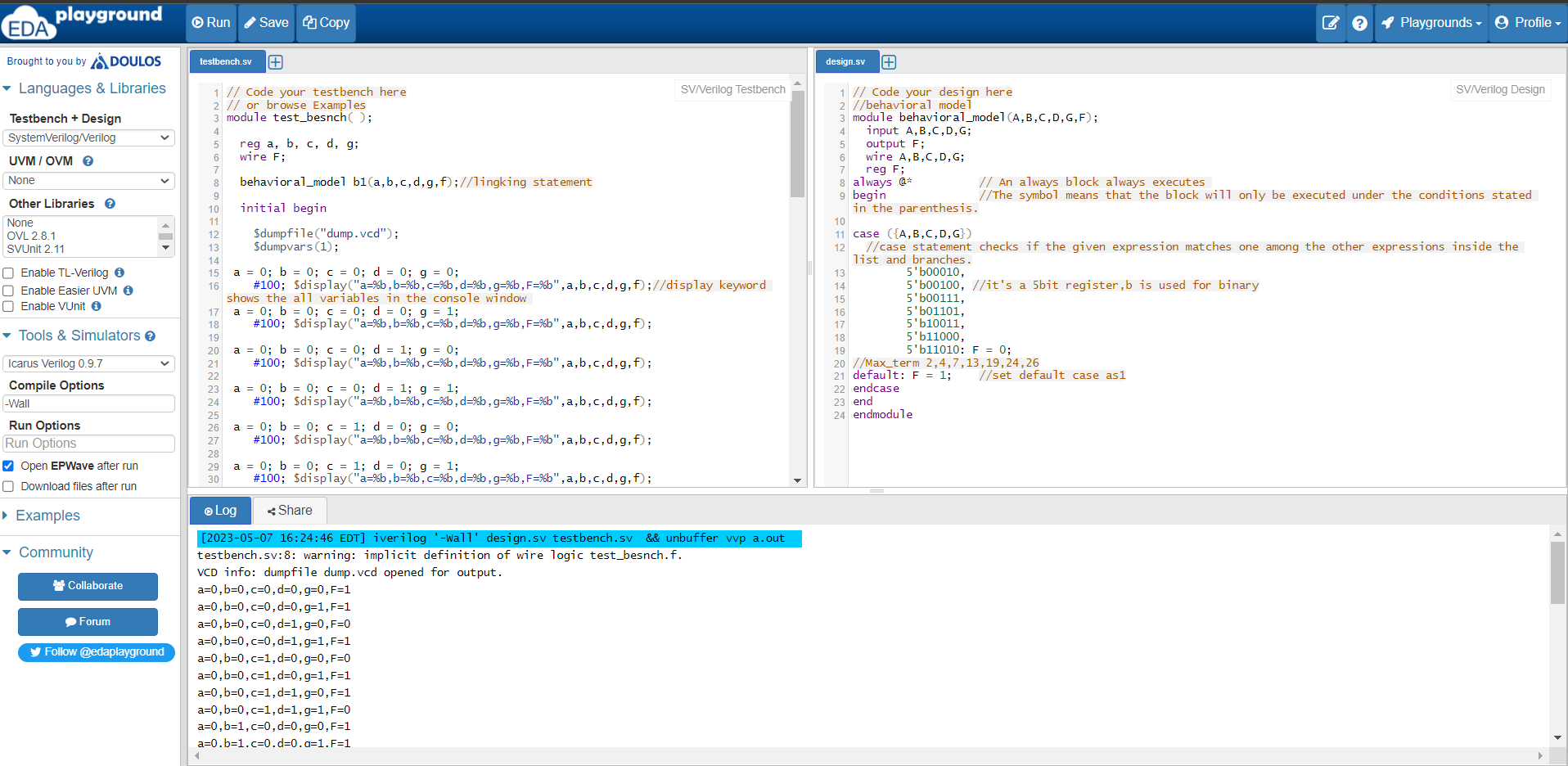
***Result Analysis:***

In this project ,we are given POS form of a Boolean expression. We have to implement it in Verilog HDL. We made truth table and identify the maxterm by numbering 0.Then we simplify it using K-map and found the simplified form,

F= (a'+b'+c+g) \* (a'+b+c+d'+g') \* (a+b+c+d'+g) \* (a+b+c'+d'+g') \* (a+b'+c'+d+g') \* (a+b+c'+d+g)

[page:12-13]

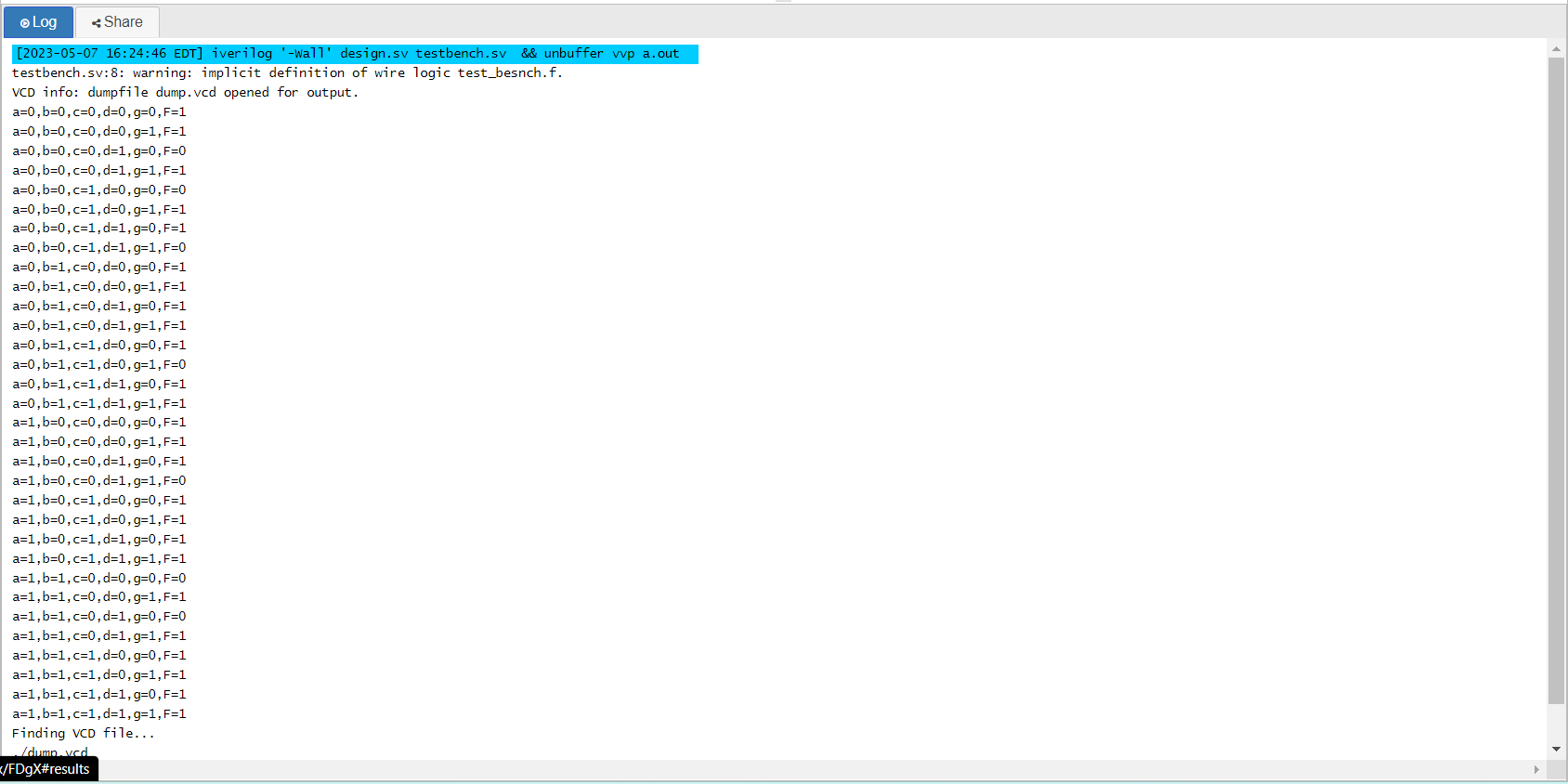
Then we use Verilog to design gate level, data flow, behavioral model. For the simulation purpose of our testbench and design segments, we use EDA playground simulator. Here is a screenshot of simulating testbench:



Test bench Section

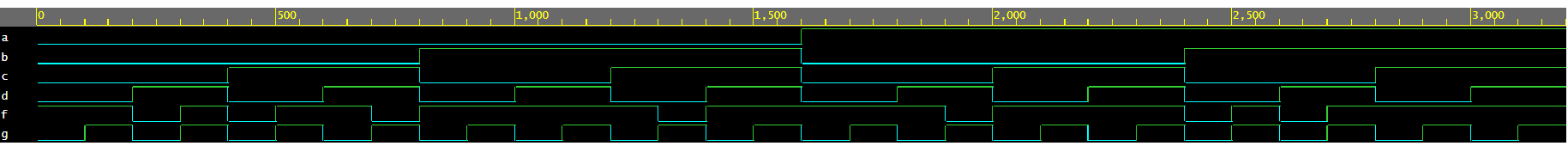
Fig-02:Screenshot of simulating Environment Design Section

We designed data flow,gate level modelling and behavioral modelling and run the test bench and got our expected result. In Test bench we use display command,so that we can see the value of every variable or the truth table in console section.After running test bench we got our expected webshape.Here we use 5 variable as input(a,b,c,d,g) and the output (F). Hence, we get 2^5=32 input and among them we have 7(2,4,7,13,19,24,26) maxterm. After running test bench we see in these 7 positions output F=0 which was our desire result;



console window truthtable

fig-03: Console window containing truth table

The output is:Fig-4: Timing diagram of the designed circuit; outpt(f), input(a,b,c,d,g)

We got 0 in 7 positions which were the maxterm in our pos form;

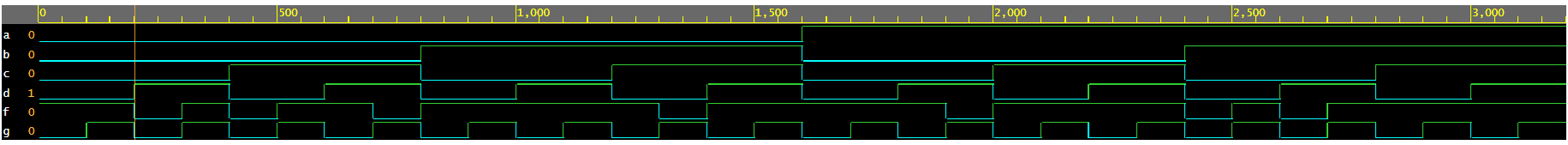


Fig-5: Timing diagram of the designed circuit at input 00010 (2) and output(f)=0

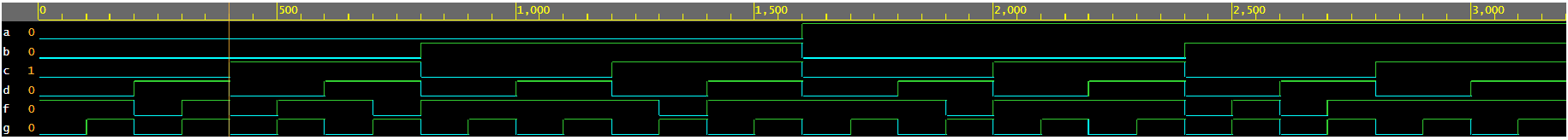
******

Fig-6: Timing diagram of the designed circuit at input 00100 (4) and output(f)=0

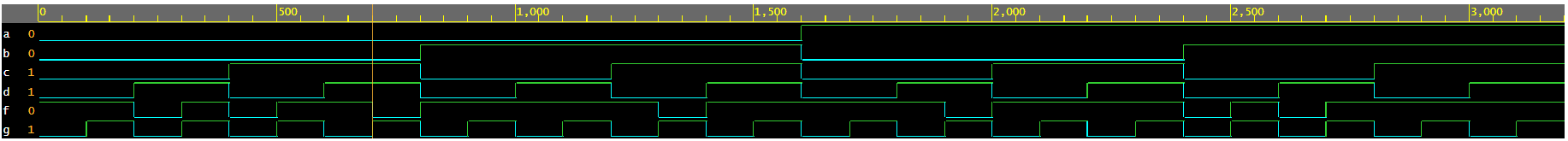
******

Fig-7: Timing diagram of the designed circuit at input 00111 (7) and output(f)=0

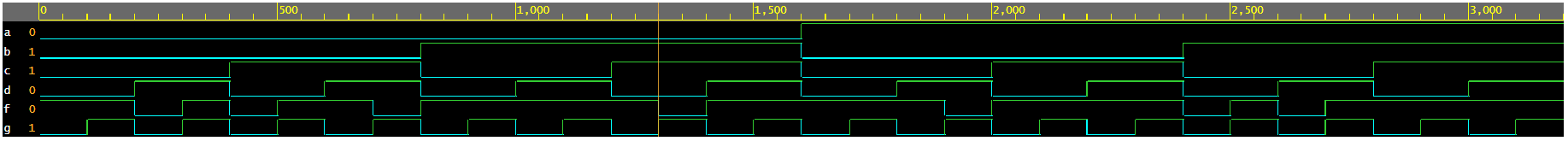
******

Fig-8: Timing diagram of the designed circuit at input 01101(13) and output(f)=0

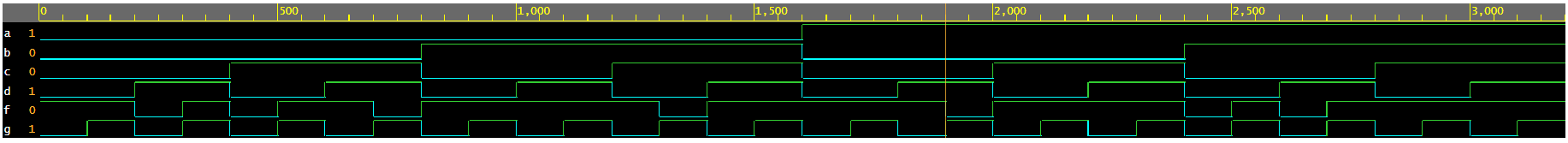
******

Fig-9: Timing diagram of the designed circuit at input 10011(19) and output(f)=0

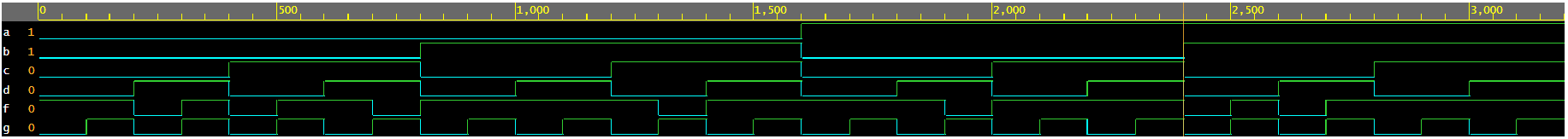
******

Fig-10: Timing diagram of the designed circuit at input 11000(24) and output(f)=0

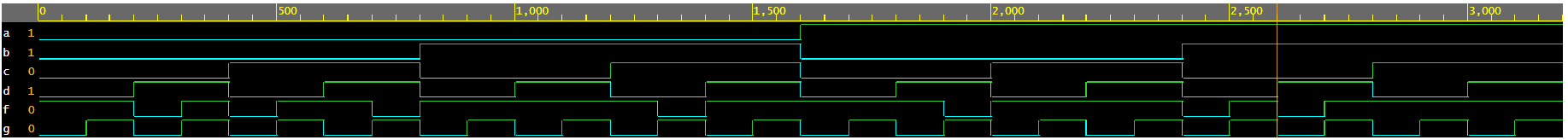
******

Fig-11: Timing diagram of the designed circuit at input 11010(26) and output(f)=0

***Discussion:***

In this project we have used k map for simplification Boolean expression. Then we used Verilog to design gate level, data flow modelling and behavioral modelling. As the test bench run perfectly, so we can say we did our project perfectly.

In k map, we first identified the maxterm and then identified the group. We found 6 different groups. Our simplified form was multiplication of these 6 groups. In circuit diagram we use OR, NOT, AND gate.

In test bench we have used display command, as a result we can find truth table in console section. In behavioral model we used conditional statement case, which is easier than if-else statement.

The goal of k-map is to simplify any Boolean expression to reduce circuit gate or circuit cost. Again Verilog is very helpful for designing circuits. So this project is very important for us and we successfully did this.

***Conclusion:***

The project of converting a POS expression to a simplified Boolean expression using K-map is an interesting and challenging task. It involves several stages of conversion and optimization, including sop form to truth table, then simplification using k-map conversion and Verilog implementation.

One of the key advantages of using Verilog to implement the simplified Boolean expression is that it can be easily synthesized into hardware, allowing the design to be implemented on an FPGA or ASIC. This provides a more efficient and reliable implementation than using software-based solutions.

The success of this project depends on the accuracy and completeness of the conversion process. The conversion from POS to simplification Boolean expression requires careful consideration of the Boolean laws and optimization techniques. The goal is to minimize the number of terms and variables while maintaining the same logical function. Thus we can reduce the number of logic gates using in a circuit and minimize the cost

Overall, this project is a valuable exercise in digital logic design and Verilog programming. It provides a practical application of Boolean algebraic laws and Verilog syntax while demonstrating the importance of optimization and synthesis in hardware design.

***Reference:***

1. Digital Logic and Computer Design

-by M. Morris Mano

1. <https://byjus.com/gate/introduction-of-k-map-karnaugh-map-notes/#:~:text=What%20is%20K%20mapping%20used,version%20of%20the%20truth%20table>.
2. <https://www.tutorialspoint.com/vlsi_design/vlsi_design_verilog_introduction.htm>
3. <https://www.electronicshub.org/boolean-logic-sop-form-pos-form/>
4. <https://www.allaboutcircuits.com/textbook/digital/chpt-8/larger-5-6-variable-karnaugh-maps/>