THE CONSTRUCTION OF A FOUR-BIT SEVEN-SEGMENT DISPLAY DECODER

UNIVERSITY OF THE PHILIPPINES VISAYAS

COLLEGE OF ARTS AND SCIENCES

DIVISION OF PHYSICAL SCIENCES AND MATHEMATICS

CMSC 130 - LOGIC DESIGN AND DIGITAL COMPUTER CIRCUITS

2ND SEMESTER AY 2021-2022

ASSIGNMENT 2. SEVEN-SEGMENT DISPLAY DECODER IN VHDL

PREPARED BY:

DELA CRUZ	,	LLOYD WALLYS
LESCANO	,	RENMAR
ORAÑO	,	MYRTLLE GEM

MARCH 28, 2022

Abstract

In this laboratory exercise, a seven-segment decoder program for 4-bit binary values was demonstrated. A solution was developed for the seven-segment decoder utilizing the Karnaugh Map method and VHDL. A truth table showing the desired bit output served as a guide for the team to identify the respective Boolean functions. Each function was mapped to simplify the required logic functions in solving the problem. The solution produced the desired outcome during testing, demonstrating the method's functionality and validity for this project. Concurrently, this approach was converted into logic and then into programs. The software was developed and compiled using GHDL, and the simulation was visualized using GTKWave. The developed program processed 4-bit inputs and correctly generated high output signals for all values denoting prime numbers. Furthermore, it also generated correct corresponding signal outputs for each segment to visually represent the sixteen hexadecimal characters.

Table of Contents

List of Figures

Design

Problem Statement:

In this assignment, the main objective is to develop a VHDL program, *SevenSegDecoder*, that accepts a 4-bit input representing the integers 0_{10} to 15_{10} and make use of a 7-Segment Display. A 7-segment display is composed of seven LEDs (Light-Emitting Diodes) arranged in a rectangular grid. A segment (or each LED) illuminates when its value is 1, indicating that it is turned on; otherwise, it is turned off. To properly display the characters from 0_{16} to F_{16}, the LEDs must be programed to turn on or off in particular sets of order.

Overview:

For a systematic procedure, the following objectives were pursued:

1. generate a truth table for every character display representation
2. identify and map the functions of each character
3. simplify the Boolean functions using the k-maps
4. translate the solution in code using VHDL
5. visualize and verify the output for Prime and SevenSeg

7-Segment Display:

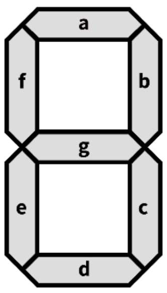


Fig. 1. A 7-segment display with each segment labeled from a to g

As illustrated in *Figure 1*, each of the seven segments was assigned a letter from *a* to *g*. These assignments were used as a reference in determining the pattern of the segments that must illuminate for each character.

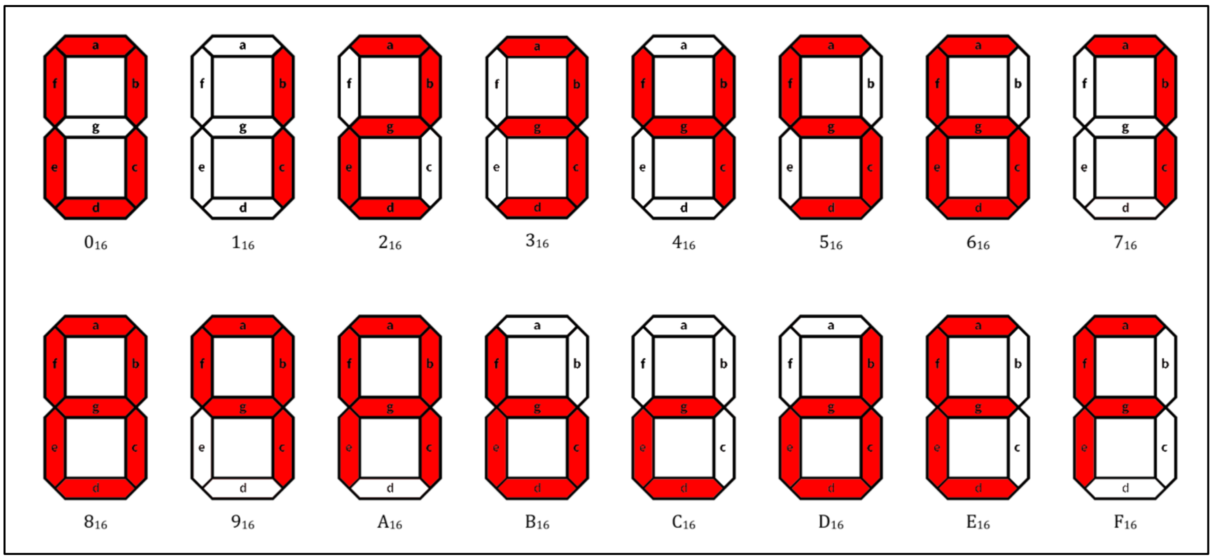


Fig. 2. Hexadecimals 0_{16} to F_{16} represented on a 7-segment display

Figure 2 illustrates various 7-segment displays with unique illumination patterns (indicated by segments with a red background) for the hexadecimal characters 0_{16} to F_{16}. The description under each pattern corresponds to the hexadecimal value represented by the 7-segment display. To ensure the uniqueness of each character representation, the hexadecimals B_{16}, C_{16}, and D_{16} were displayed as lowercase letters. Accordingly, these specifications were used as the guide in generating the truth table below.

TABLE 1
THE TRUTH TABLE FOR THE CORRESPONDING BITS (A TO G) IN THE SEVENSEG BIT VECTOR

Hex	SW	Prime	a	b	c	d	e	f	g
0	0000	0	1	1	1	1	1	1	0
1	0001	0	0	1	1	0	0	0	0
2	0010	1	1	1	0	1	1	0	1
3	0011	1	1	1	1	1	0	0	1
4	0100	0	0	1	1	0	0	1	1
5	0101	1	1	0	1	1	0	1	1
6	0110	0	1	0	1	1	1	1	1
7	0111	1	1	1	1	0	0	0	0
8	1000	0	1	1	1	1	1	1	1
9	1001	0	1	1	1	1	0	1	1
A	1010	0	1	1	1	0	1	1	1
b	1011	1	0	0	1	1	1	1	1
c	1100	0	0	0	0	1	1	0	1
d	1101	1	0	1	1	1	1	0	1
E	1110	0	1	0	0	1	1	1	1
F	1111	0	1	0	0	0	1	1	1

Table 1 shows a truth table for the desired patterns of each 7-Segment Display. The leftmost column has hexadecimal numbers 0_{16} to F_{16}, with each character display represented by the values in columns (or segments) *a*, *b*, *c*, *d*, *e*, *f*, and *g*. A value of 0 indicates that the segment is turned off, while a value of 1 indicates that the segment is illuminated.

Karnaugh Maps:

By observing Table 1, we can deduce that the corresponding Boolean function for each segment will be extensive. Thus, a trial-and-error simplification process is impractical. Consequently, as the optimal method, Karnaugh Maps (k-maps) were used to to graph and visually illustrate the simplification of each function.

TABLE 2
THE K-MAP VARIABLES AND THEIR EQUIVALENT INPUT NOTATIONS

K-map Variables	**Input Equivalent**
w	sw(3)
x	sw(2)
y	sw(1)
z	sw(0)

To note, the bits in bit vector *sw* is indexed 3 down to 0. Thus, w represents the first bit, x for the second, y for the third, and lastly, z represents the final bit.

To start, given a(w, x, y, z) = Σ(0, 2, 3, 5, 6, 7, 8, 9, 10, 14, 15) from Table 1, a k-map can be generated for *a* as shown in Figure 3.

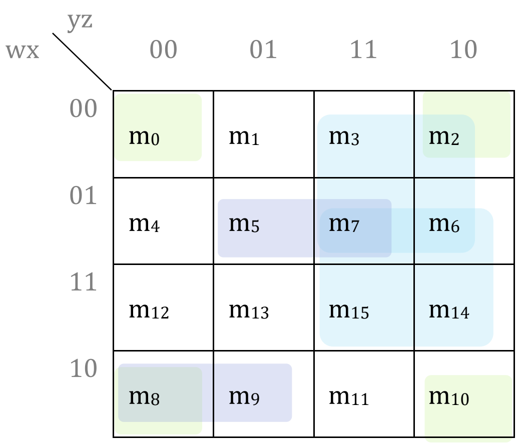


Fig. 3. The K-map representation and groupings for segment *a* where the minterms are grouped into corresponding essential implicants.

Following these essential implicants, we can simplify the Boolean function for a(w, x, y, z) through:

(0, 2, 8, 10)
= (w’x’y’z’) + (w’x’yz’) + (wx’y’z') + (wx’yz’)
= (w’x’z’) + (wx’z’)
= x’z’

(2, 3, 6, 7)
= (w’x’yz’) + (w’x’yz) + (w’xyz') + (w’xyz)
= (w’x’y) + (w’xy)
= w’y

(6, 7, 14,15)
= (w’xyz’) + (w’xyz) + (wxyz’) + (wxyz)
= (w’xy) + (wxy)
= xy

(5, 7)
= (w’xy’z) + (w’xyz)
= w’xz

(8, 9)
= (wx’y’z’) + (wx’y’z)
= wx’y’

$\boxed{\text{a(w, x, y, z) = x’z’ + w’y + xy + w’xz + wx’y’}}$

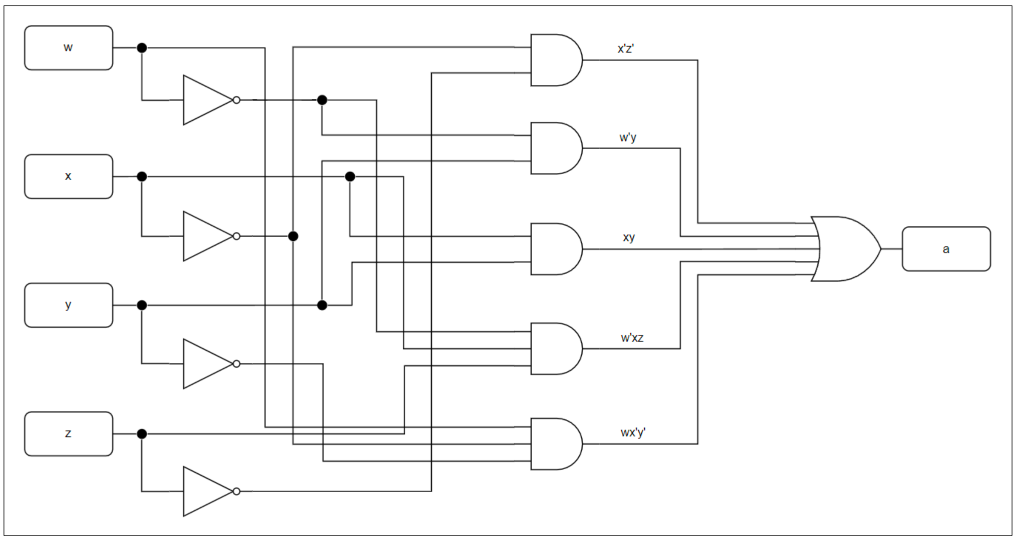


Fig. 3.1. A logic gate representation of the Boolean function
a(w, x, y, z) = x’z’ + w’y + xy + w’xz + wx’y’

Then, given $b(w, x, y, z) = \Sigma(0, 1, 2, 3, 4, 7, 8, 9, 10, 13)$ from Table 1, a k-map can be generated for *b* as shown in Figure 4.

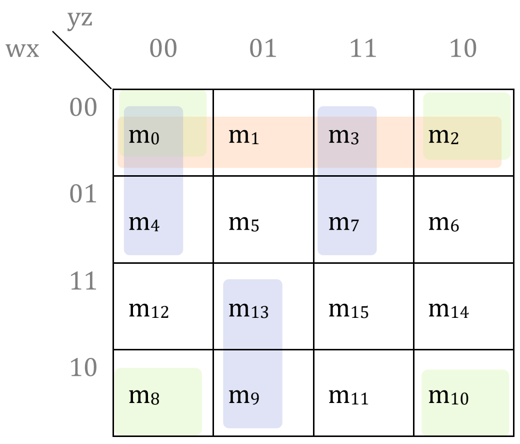


Fig. 4. The K-map representation and groupings for segment *b* where the minterms are grouped into corresponding essential implicants.

Following these essential implicants, we can simplify the Boolean function for $b(w, x, y, z)$ through:

$(0, 1, 2, 3)$
$= (w'x'y'z') + (w'x'y'z) + (w'x'yz') + (w'x'yz)$
$= (w'x'y') + (w'x'y)$
$= w'x'$

$(0, 2, 8, 10)$
$= (w'x'y'z') + (w'x'yz') + (wx'y'z') + (wx'yz')$
$= (w'x'z') + (wx'z')$
$= x'z'$

$(0, 4)$
$= (w'x'y'z') + (w'xy'z')$
$= w'y'z'$

$(3, 7)$
$= (w'x'yz) + (w'xyz)$
$= w'yz$

$(9, 13)$
$= (wx'y'z) + (wxy'z)$
$= wy'z$

$$\boxed{b(w, x, y, z) = w'x' + x'z' + w'y'z' + w'yz + wy'z}$$

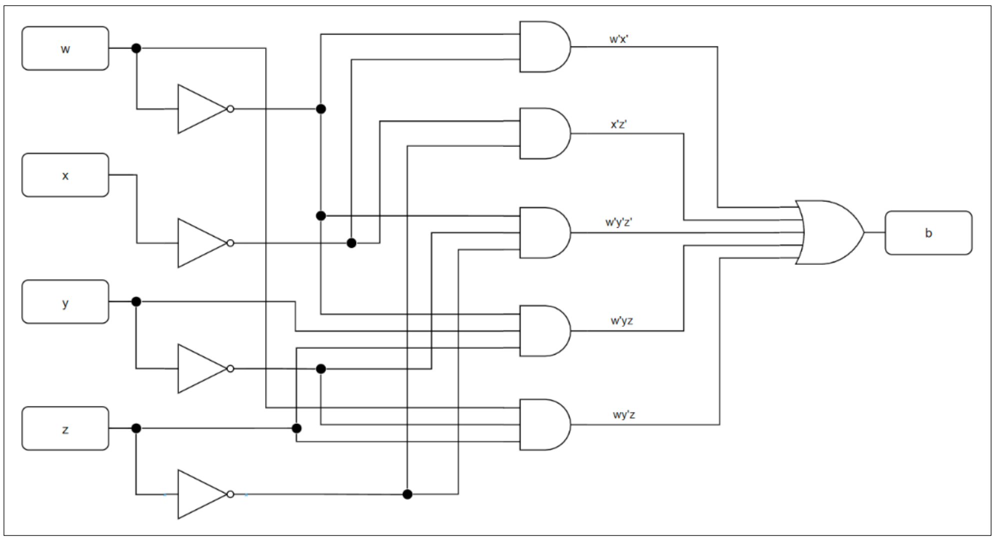


Fig. 4.1. A logic gate representation of the Boolean function
b(w, x, y, z) = w'x' + x'z' + w'y'z' + w'yz + wy'z

Next, given c(w, x, y, z) = Σ(0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13) from Table 1, a a k-map can be generated for *c* as shown in Figure 5.

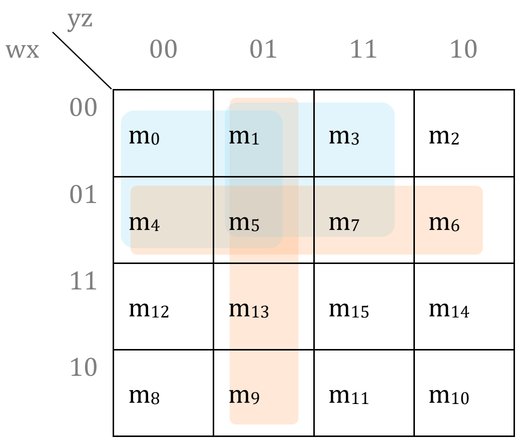


Fig. 5. The K-map representation and groupings for segment *c* where the minterms are grouped into corresponding essential implicants.

Following these essential implicants, we can simplify the Boolean function for c(w, x, y, z) through:

(0, 1, 4, 5)
= (w'x'y'z') + (w'x'y'z) + (w'xy'z') + (w'xy'z)
= (w'x'y') + (w'xy')
= w'y'

(1, 3, 5, 7)
= (w'x'y'z) + (w'x'yz) + (w'xy'z) + (w'xyz)
= (w'x'z) + (w'xz)
= w'z

(1, 5, 9, 13)
= (w'x'y'z) + (w'xy'z) + (wx'y'z) + (wxy'z)
= (w'y'z) + (wy'z)
= y'z

(4, 5, 6, 7)
= (w'xy'z') + (w'xy'z) + (w'xyz') + (w'xyz)
= (w'xy') + (w'xy)
= w'x

(8, 9, 10, 11)
= (wx'y'z') + (wx'y'z) + (wx'yz') + (wx'yz)
= (wx'y') + (wx'y)
= wx'

$\boxed{c(w, x, y, z) = w'y' + w'z + y'z + w'x + wx'}$

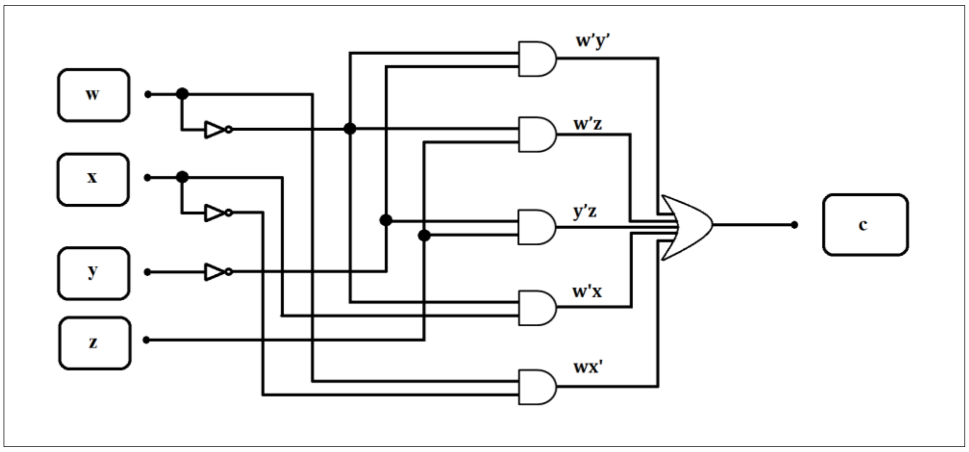


Fig. 5.1. A logic gate representation of the Boolean function
c(w, x, y, z) = w'y' + w'z + y'z + w'x + wx'

Meanwhile, given d(w, x, y, z) = Σ(0, 2, 3, 5, 6, 8, 9, 11, 12, 13, 14) from Table 1, a k-map can be generated for *d* as shown in Figure 6.

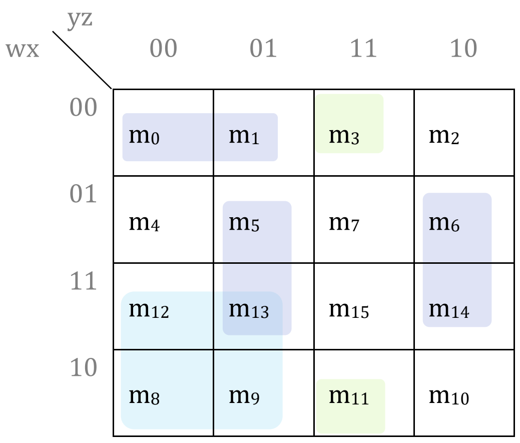


Fig. 6. The K-map representation and groupings for segment *d* where the minterms are grouped into corresponding essential implicants.

Following these essential implicants, we can simplify the Boolean function for d(w, x, y, z) through:

(8, 9, 12, 13)

$$= (wx'y'z') + (wx'y'z) + (wxy'z') + (wxy'z)$$
$$= (wx'y) + (wxy')$$
$$= wy'$$

(0, 1)

$$= (w'x'y'z') + (w'x'yz')$$
$$= w'x'z'$$

(3, 11)

$$= (w'x'yz) + (wx'yz)$$
$$= x'yz$$

(5, 13)

$$= (w'xy'z) + (wxy'z)$$
$$= xy'z$$

(6, 14)

$$= (w'xyz') + (wxyz')$$
$$= xyz'$$

$$\boxed{d(w, x, y, z) = wy' + w'x'z' + x'yz + xy'z + xyz}$$

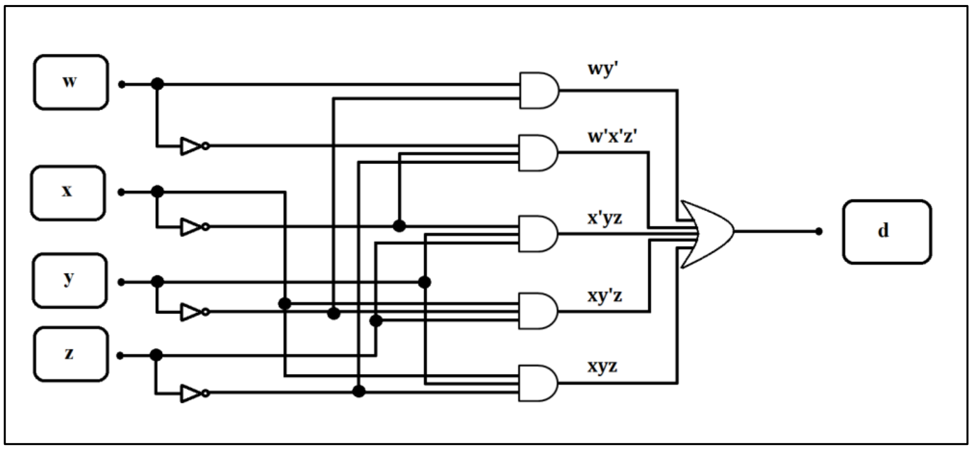


Fig. 6.1. A logic gate representation of the Boolean function
d(w, x, y, z) = wy' + w'x'z' + x'yz + xy'z + xyz

Followed by e(w, x, y, z) = Σ(0, 2, 6, 8, 10, 11, 12, 13, 14, 15) from Table 1, a k-map can be generated for *e* as shown in Figure 7.

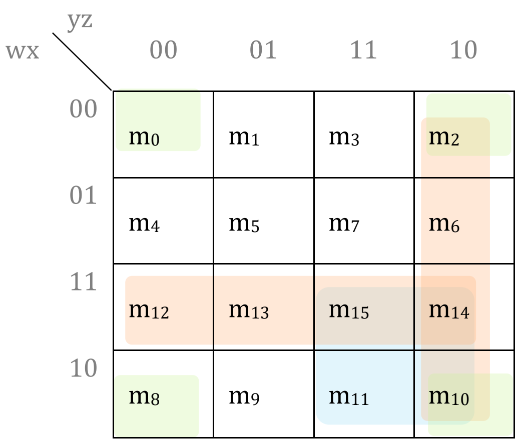


Fig. 7. The K-map representation and groupings for segment *e* where the minterms are grouped into corresponding essential implicants.

Following these essential implicants, we can simplify the Boolean function for e(w, x, y, z) through:

(0, 2, 8, 10)
$$= (w'x'y'z') + (w'x'yz') + (wx'y'z') + (wx'yz')$$
$$= (w'x'z') + (wx'z')$$
$$= x'z'$$

(2, 6, 10, 14)
$$= (w'x'yz') + (w'xyz') + (wxyz') + (wx'yz')$$
$$= (w'yz') + (wyz')$$
$$= yz'$$

$$\begin{aligned}(10, 11, 14, 15) \\ &= (wx'yz) + (wx'yz') + (wxyz) + (wxyz') \\ &= (wx'y) + (wxy) \\ &= wy\end{aligned}$$

$$\begin{aligned}(12, 13, 14, 15) \\ &= (wxy'z') + (wxy'z) + (wxyz) + (wxyz') \\ &= (wxy') + (wxy) \\ &= wx\end{aligned}$$

$$\boxed{e(w, x, y, z) = x'z' + yz' + wy + wx}$$

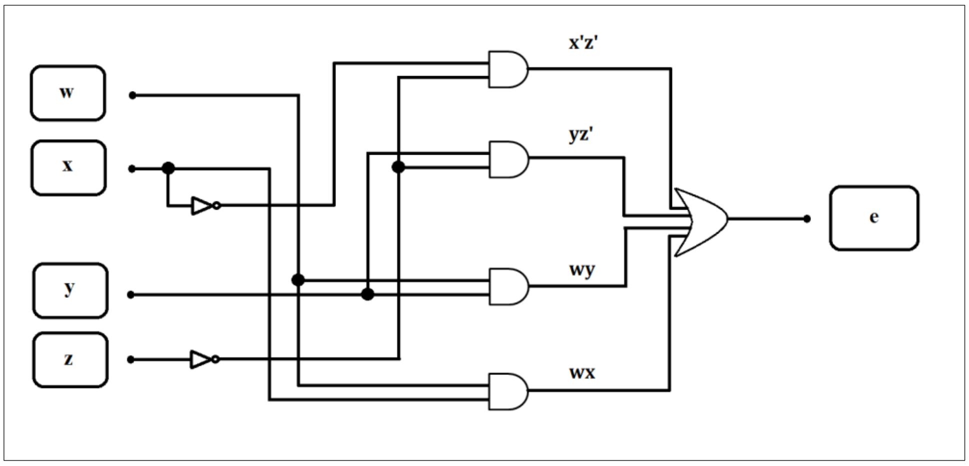
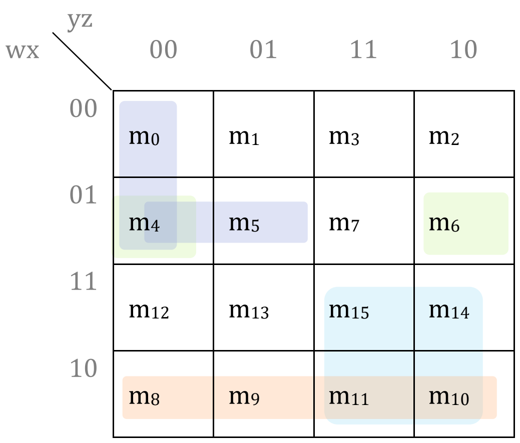


Fig. 7.1. A logic gate representation of the Boolean function
$e(w, x, y, z) = x'z' + yz' + wy + wx$

Then, for $f(w, x, y, z) = \Sigma(0, 2, 3, 5, 6, 8, 9, 11, 12, 13, 14)$ from Table 1, a k-map can be generated for *f* as shown in Figure 8.



wx \ yz	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

Fig. 8. The k-map representation and groupings for segment *f* where the minterms are grouped into corresponding essential implicants.

Following these essential implicants, we can simplify the Boolean function for f(w, x, y, z) through:

$$\begin{aligned}&(8, 9, 10, 11)\\ &\quad = (wx'y'z') + (wx'y'z) + (wx'yz) + (wx'yz')\\ &\quad = (wx'y') + (wx'y)\\ &\quad = wx'\end{aligned}$$

$$\begin{aligned}&(10, 11, 14, 15)\\ &\quad = (wx'yz) + (wx'yz') + (wxyz) + (wxyz')\\ &\quad = (wxy) + (wx'y)\\ &\quad = wy\end{aligned}$$

$$\begin{aligned}&(0, 4)\\ &\quad = (w'x'y'z') + (w'xy'z)'\\ &\quad = w'y'z'\end{aligned}$$

$$\begin{aligned}&(4, 5)\\ &\quad = (w'xy'z') + (w'xy'z)\\ &\quad = w'xy'\end{aligned}$$

$$\begin{aligned}&(4, 6)\\ &\quad = (w'xy'z') + (w'xyz')\\ &\quad = w'xz'\end{aligned}$$

$$\boxed{f(w, x, y, z) = wx' + wy + w'y'z' + w'xy' + w'xz'}$$

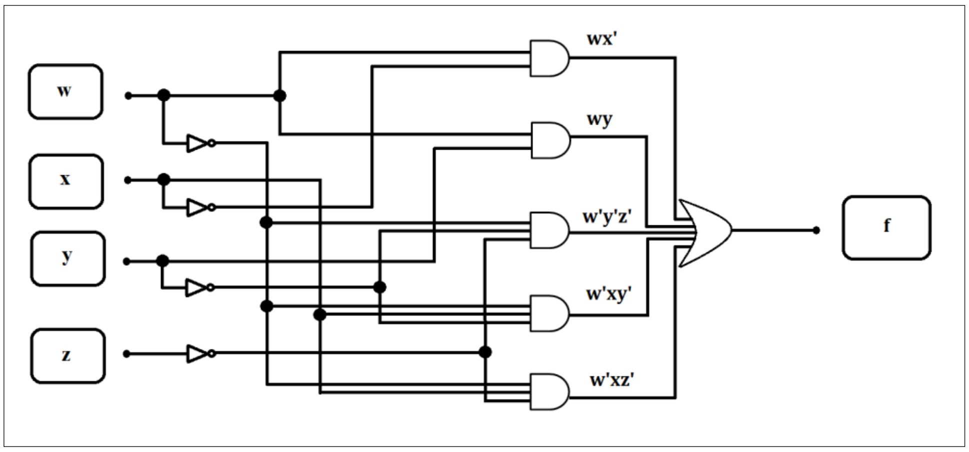


Fig. 8.1. A logic gate representation of the Boolean function
f(w, x, y, z) = wx' + wy + w'y'z' + w'xy' + w'xz'

Finally, we have g(w, x, y, z) = Σ(0, 2 ,3, 5, 6 ,8, 9, 11, 12, 13, 14) from Table 1, a k-map can be generated for *g* as shown in Figure 9.

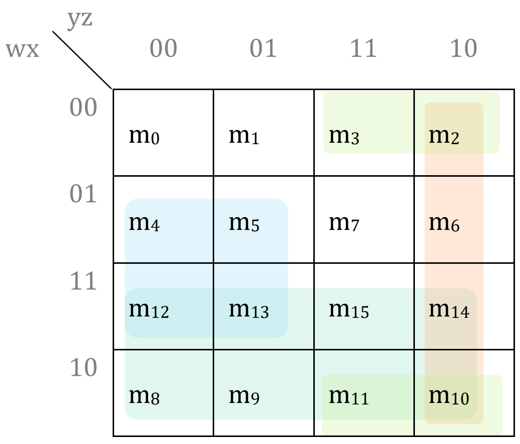


Fig. 9. The k-map representation and groupings for segment *g* where the minterms are grouped into corresponding essential implicants.

Following these essential implicants, we can simplify the Boolean function for g(w, x, y, z) through:

$$\begin{aligned}
&(8, 9, 10, 11, 12, 13, 14, 15)\\
&\qquad = (wx'y'z') + (wx'y'z) + (wx'yz) + (wx'yz')\\
&\qquad\quad + (wxy'z') + (wxy'z) + (wxyz) + (wxyz')\\
&\qquad = (wx'y') + (wx'y) + (wxy') + (wxy)\\
&\qquad = (wx) + (wx')\\
&\qquad = w
\end{aligned}$$

$$\begin{aligned}
&(2, 3, 10, 11)\\
&\qquad = (w'x'yz) + (w'x'yz') + (wx'yz) + (wx'yz')\\
&\qquad = (w'x'y) + (wx'y)\\
&\qquad = x'y
\end{aligned}$$

$$\begin{aligned}
&(2, 6, 10, 14)\\
&\qquad = (w'x'yz') + (w'xyz') + (wxyz') + (wx'yz')\\
&\qquad = (w'yz') + (wyz')\\
&\qquad = yz'
\end{aligned}$$

$$\begin{aligned}
&(4, 5, 12, 13)\\
&\qquad = (w'xy'z') + (w'xy'z) + (wxy'z') + (wxy'z)\\
&\qquad = (w'xy') + (wxy')\\
&\qquad = xy'
\end{aligned}$$

$$\boxed{g(w, x, y, z) = w + x'y + yz' + xy'}$$

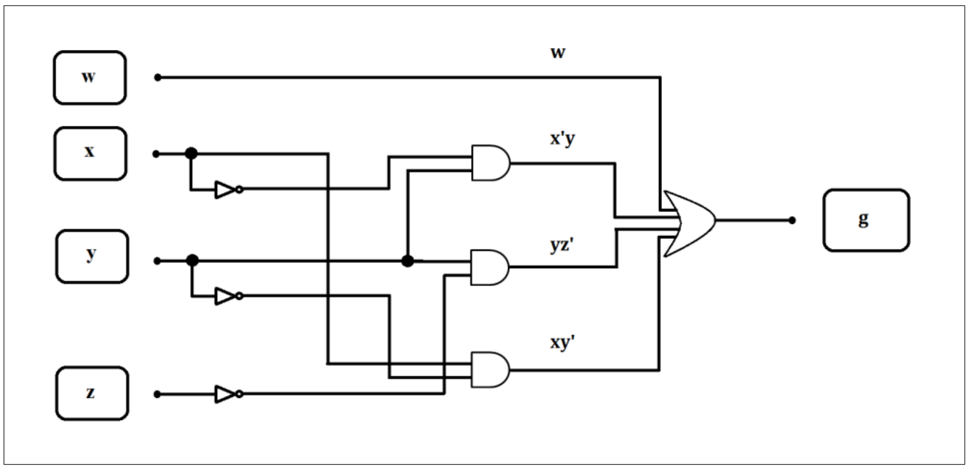


Fig. 9.1. A logic gate representation of the Boolean function
g(w, x, y, z) = w + x'y + yz' + xy'

For improved readability, the simplified Boolean functions for all seven segments are compiled and summarized in Table 3.

TABLE 3
THE SIMPLIFIED BOOLEAN FUNCTIONS OF SEGMENTS A TO G

Segment	**Boolean Function**
a	x’z’ + w’y + xy + w’xz + wx’y’
b	w’x’ + x’z’ + w’y’z’ + w’yz + wy’z
c	w’y’ + w’z + y’z + w’x + wx’
d	wy' + w'x'z' + x'yz + xy'z + xyz
e	x'z' + yz' + wy + wx
f	wx' + wy + w'y'z' + w'xy' + w'xz'
g	w + x'y + yz' + xy'

Finally, we can revert these functions into the input notation of *sw* to develop our solution in VHDL.

TABLE 4
THE INPUT LOGIC FUNCTION FOR SEGMENTS A TO G

Segment	Boolean Function
a	sw(2)’sw(0)’ + sw(3)’sw(1) + sw(2)sw(1) + sw(3)’sw(2)sw(0) + sw(3)sw(2)’sw(1)’
b	sw(3)’sw(2)’ + sw(2)’sw(0)’ + sw(3)’sw(1)’sw(0)’ + sw(3)’sw(1)sw(0) + sw(3)sw(1)’sw(0)
c	sw(3)’sw(1)’ + sw(3)’sw(0) + sw(1)’sw(0) + sw(3)’sw(2) + sw(3)sw(2)’
d	sw(3)sw(1)' + sw(3)'sw(2)'sw(0)' + sw(2)'sw(1)sw(0) + sw(2)sw(1)'sw(0) + sw(2)sw(1)sw(0)
e	sw(2)'sw(0)' + sw(1)sw(0)' + sw(3)sw(1) + sw(3)sw(2)
f	sw(3)sw(2)' + sw(3)sw(1) + sw(3)'sw(1)'sw(0)' + sw(3)'sw(2)sw(1)' + sw(3)'sw(2)sw(0)'
g	sw(3) + sw(2)'sw(1) + sw(1)sw(0)' + sw(2)sw(1)'

Testing and Debugging

Since it was mathematically solved, no major error was encountered. However, one logic error was detected on the third bit or segment *c* of the representation for F_{16} (see *Fig 10*). Due to the specific nature of the error, it was quickly traced back in line 47 of the main program (see *Fig 11*).

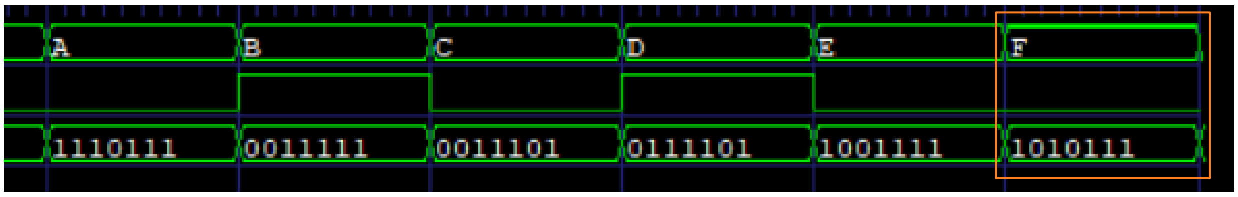
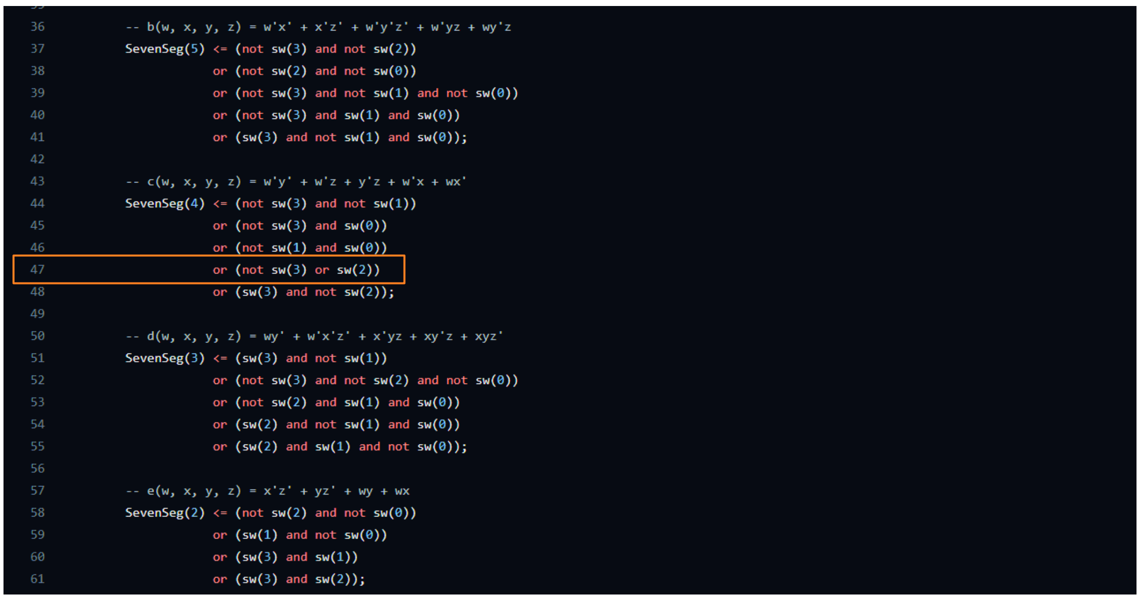


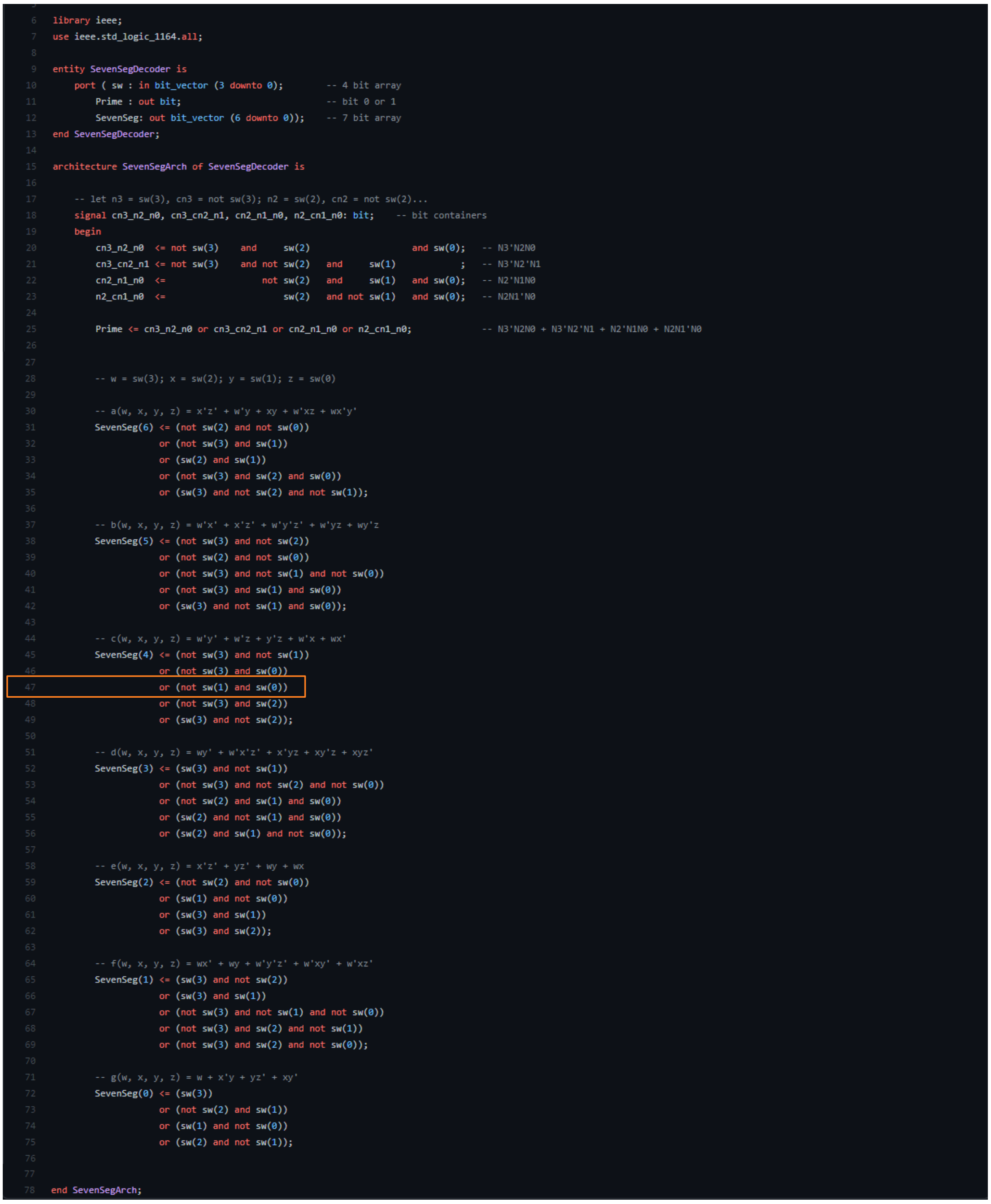
Fig. 10. The observed incorrect wave output in GTKWave



```
36          -- b(w, x, y, z) = w'x' + x'z' + w'y'z' + w'yz + wy'z
37          SevenSeg(5) <= (not sw(3) and not sw(2))
38                      or (not sw(2) and not sw(0))
39                      or (not sw(3) and not sw(1) and not sw(0))
40                      or (not sw(3) and sw(1) and sw(0))
41                      or (sw(3) and not sw(1) and sw(0));
42
43          -- c(w, x, y, z) = w'y' + w'z + y'z + w'x + wx'
44          SevenSeg(4) <= (not sw(3) and not sw(1))
45                      or (not sw(3) and sw(0))
46                      or (not sw(1) and sw(0))
47                      or (not sw(3) or sw(2))
48                      or (sw(3) and not sw(2));
49
50          -- d(w, x, y, z) = wy' + w'x'z' + x'yz + xy'z + xyz'
51          SevenSeg(3) <= (sw(3) and not sw(1))
52                      or (not sw(3) and not sw(2) and not sw(0))
53                      or (not sw(2) and sw(1) and sw(0))
54                      or (sw(2) and not sw(1) and sw(0))
55                      or (sw(2) and sw(1) and not sw(0));
56
57          -- e(w, x, y, z) = x'z' + yz' + wy + wx
58          SevenSeg(2) <= (not sw(2) and not sw(0))
59                      or (sw(1) and not sw(0))
60                      or (sw(3) and sw(1))
61                      or (sw(3) and sw(2));
```

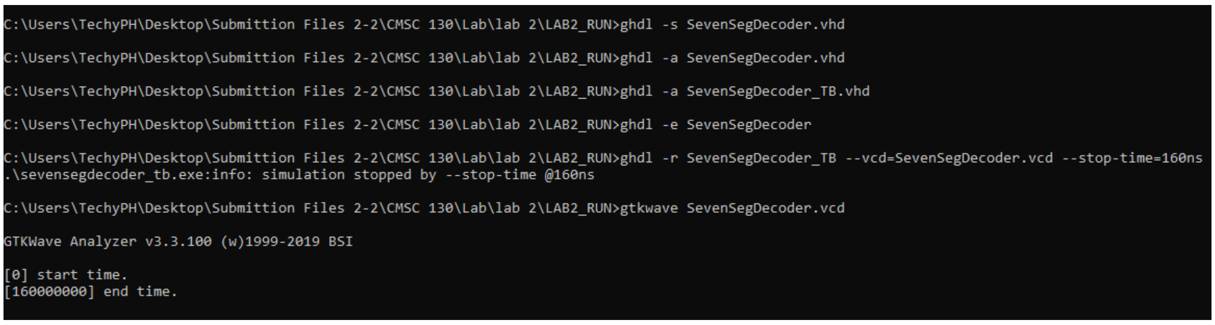
Fig. 11. The source of conflict traced in line 47 SevenSegDecoder

As observed, the Boolean functions were supposedly expressed in Sum of Products (SOP) forms and the identified dot product was incorrectly translated using an *or* operator. Accordingly, this was converted into an *and* operation as seen in the working code in Figure 12.



```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity SevenSegDecoder is
    port ( sw : in bit_vector (3 downto 0);        -- 4 bit array
        Prime : out bit;                           -- bit 0 or 1
        SevenSeg: out bit_vector (6 downto 0));    -- 7 bit array
end SevenSegDecoder;

architecture SevenSegArch of SevenSegDecoder is

    -- let n3 = sw(3), cn3 = not sw(3); n2 = sw(2), cn2 = not sw(2)...
    signal cn3_n2_n0, cn3_cn2_n1, cn2_n1_n0, n2_cn1_n0: bit;    -- bit containers
    begin
        cn3_n2_n0  <= not sw(3)    and     sw(2)                   and sw(0);   -- N3'N2N0
        cn3_cn2_n1 <= not sw(3)    and not sw(2)   and     sw(1)              ;   -- N3'N2'N1
        cn2_n1_n0  <=                  not sw(2)   and     sw(1)   and sw(0);   -- N2'N1N0
        n2_cn1_n0  <=                      sw(2)   and not sw(1)   and sw(0);   -- N2N1'N0

        Prime <= cn3_n2_n0 or cn3_cn2_n1 or cn2_n1_n0 or n2_cn1_n0;             -- N3'N2N0 + N3'N2'N1 + N2'N1N0 + N2N1'N0


        -- w = sw(3); x = sw(2); y = sw(1); z = sw(0)

        -- a(w, x, y, z) = x'z' + w'y + xy + w'xz + wx'y'
        SevenSeg(6) <= (not sw(2) and not sw(0))
                    or (not sw(3) and sw(1))
                    or (sw(2) and sw(1))
                    or (not sw(3) and sw(2) and sw(0))
                    or (sw(3) and not sw(2) and not sw(1));

        -- b(w, x, y, z) = w'x' + x'z' + w'y'z' + w'yz + wy'z
        SevenSeg(5) <= (not sw(3) and not sw(2))
                    or (not sw(2) and not sw(0))
                    or (not sw(3) and not sw(1) and not sw(0))
                    or (not sw(3) and sw(1) and sw(0))
                    or (sw(3) and not sw(1) and sw(0));

        -- c(w, x, y, z) = w'y' + w'z + y'z + w'x + wx'
        SevenSeg(4) <= (not sw(3) and not sw(1))
                    or (not sw(3) and sw(0))
                    or (not sw(1) and sw(0))
                    or (not sw(3) and sw(2))
                    or (sw(3) and not sw(2));

        -- d(w, x, y, z) = wy' + w'x'z' + x'yz + xy'z + xyz'
        SevenSeg(3) <= (sw(3) and not sw(1))
                    or (not sw(3) and not sw(2) and not sw(0))
                    or (not sw(2) and sw(1) and sw(0))
                    or (sw(2) and not sw(1) and sw(0))
                    or (sw(2) and sw(1) and not sw(0));

        -- e(w, x, y, z) = x'z' + yz' + wy + wx
        SevenSeg(2) <= (not sw(2) and not sw(0))
                    or (sw(1) and not sw(0))
                    or (sw(3) and sw(1))
                    or (sw(3) and sw(2));

        -- f(w, x, y, z) = wx' + wy + w'y'z' + w'xy' + w'xz'
        SevenSeg(1) <= (sw(3) and not sw(2))
                    or (sw(3) and sw(1))
                    or (not sw(3) and not sw(1) and not sw(0))
                    or (not sw(3) and sw(2) and not sw(1))
                    or (not sw(3) and sw(2) and not sw(0));

        -- g(w, x, y, z) = w + x'y + yz' + xy'
        SevenSeg(0) <= (sw(3))
                    or (not sw(2) and sw(1))
                    or (sw(1) and not sw(0))
                    or (sw(2) and not sw(1));


end SevenSegArch;
```

Fig. 12. The working code for SevenSegDecoder



```
C:\Users\TechyPH\Desktop\Submittion Files 2-2\CMSC 130\Lab\lab 2\LAB2_RUN>ghdl -s SevenSegDecoder.vhd

C:\Users\TechyPH\Desktop\Submittion Files 2-2\CMSC 130\Lab\lab 2\LAB2_RUN>ghdl -a SevenSegDecoder.vhd

C:\Users\TechyPH\Desktop\Submittion Files 2-2\CMSC 130\Lab\lab 2\LAB2_RUN>ghdl -a SevenSegDecoder_TB.vhd

C:\Users\TechyPH\Desktop\Submittion Files 2-2\CMSC 130\Lab\lab 2\LAB2_RUN>ghdl -e SevenSegDecoder

C:\Users\TechyPH\Desktop\Submittion Files 2-2\CMSC 130\Lab\lab 2\LAB2_RUN>ghdl -r SevenSegDecoder_TB --vcd=SevenSegDecoder.vcd --stop-time=160ns
.\sevensegdecoder_tb.exe:info: simulation stopped by --stop-time @160ns

C:\Users\TechyPH\Desktop\Submittion Files 2-2\CMSC 130\Lab\lab 2\LAB2_RUN>gtkwave SevenSegDecoder.vcd

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

[0] start time.
[160000000] end time.
```

Fig. 13. The GHDL and GTKWave commands to build and run SevenSegDecoder.

Finally, after another test run, the program worked correctly and produced the expected results where it would generate 1 for the segments that needs to be lit, 0 otherwise. As seen in Figure 14, the signals were simulated and displayed using GTKWave. For one full cycle, the simulation time was set at 160 nanoseconds. It is worth noting that GTKWave's default viewing format is hexadecimal, which was changed to binary to match the intended outcomes of this laboratory exercise.

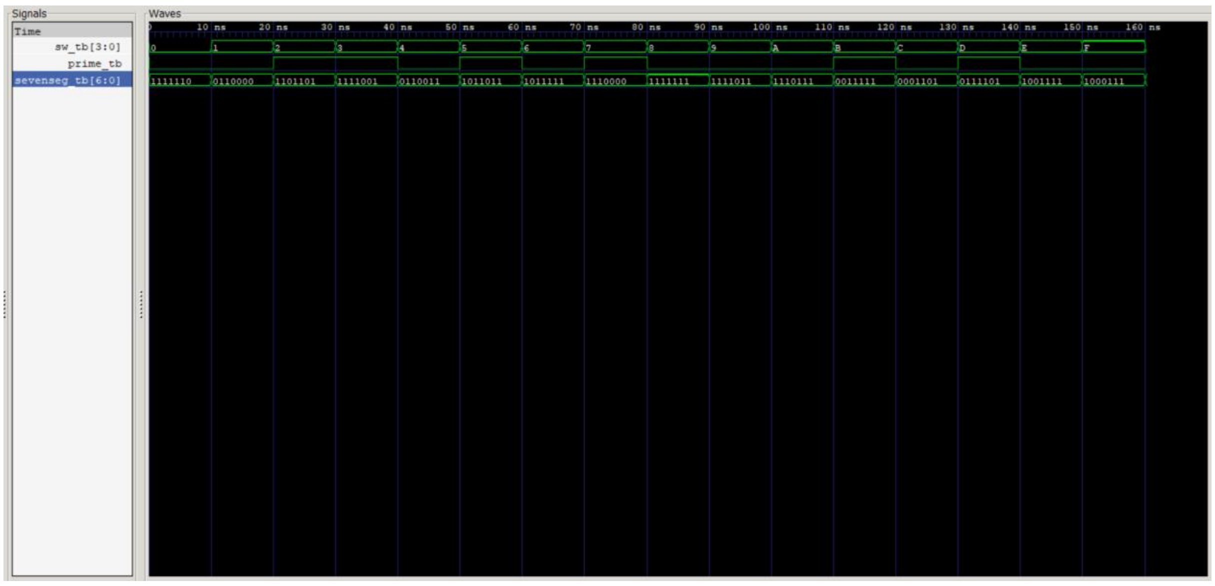


Fig. 14. The expected correct wave output of SevenSegDecoder

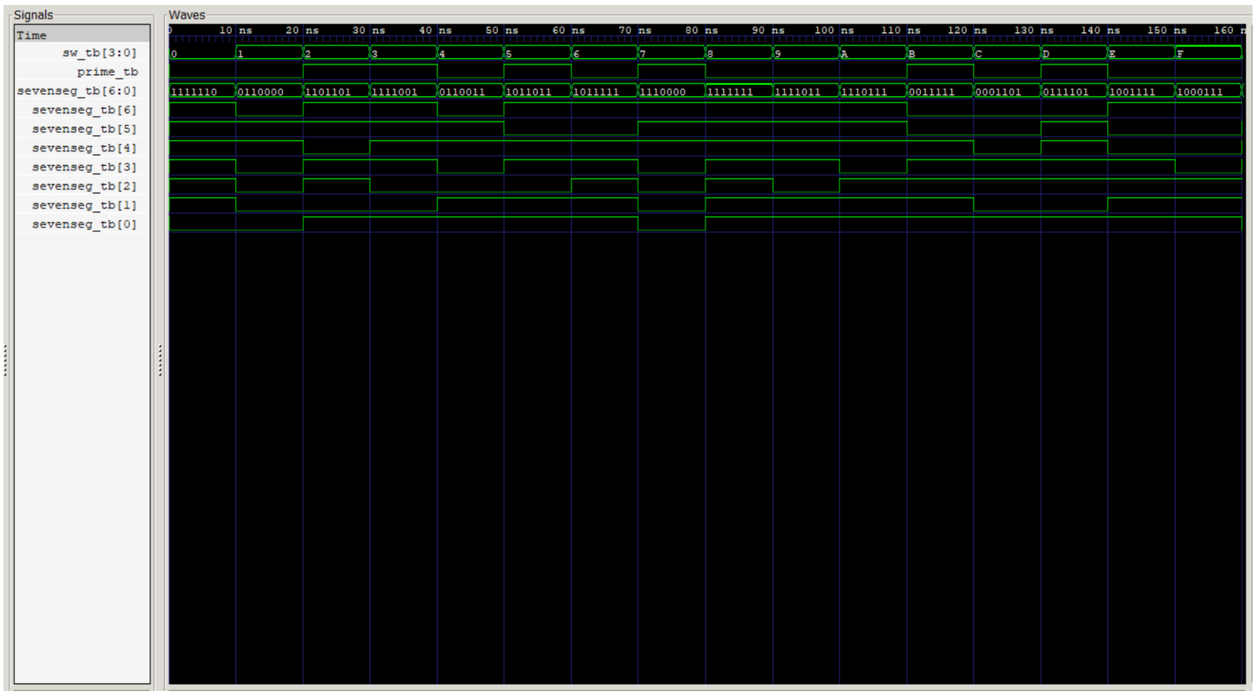


Fig. 15. The expanded wave output in GTKWave representing
the signal for each bit in bit vector SevenSeg

Conclusion

The purpose of this laboratory assignment was to design a Seven-Segment Decoder program that takes a 4-bit input and outputs numbers between 0_{16} and F_{16}, while reusing the codes from the previous laboratory. A truth table was generated as the comprehensive basis for the Boolean functions of every hexadecimal character display representation. In connection, Karnaugh Maps were created to simplify and minimize the terms of the Boolean functions extracted from the table. Then, the simplified Boolean functions were translated in code using VHDL. Upon testing, only a minor error was encountered in the segment representation and was immediately resolved by tracing it back in the main program. Finally, the program run successfully and produced the expected results. The architecture was able to determine all prime numbers and showed the seven segments for hexadecimals 0_{16} to F_{16}, as shown in GTKWave, thus, verifying the functionality and validity of the solutions. It is recommended for future trials to regularly cross-check the code to avoid critical logic errors.

References

Electronics Tutorials, “7-segment display and driving a 7-segment display,” *Basic Electronics Tutorials*, 07-Oct-2021. [Online]. Available: https://www.electronics-tutorials.ws/blog/7-segment-display-tutorial.html.

M. M. Mano and M. D. Ciletti, “Digital Design: with an introduction to the Verilog HDL (6th ed.)”, Boston, Massachusetts: Pearson Education Limited, 2018.