

Solution to **Problem 1: No-NAND Workaround**

We are asked to check the equality between the circuit in Figure 1-1 and the *NAND* gate.

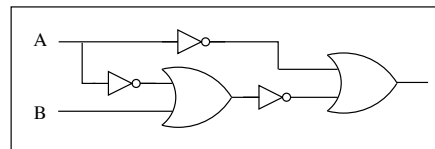


Figure 1-1: Logic circuit

Solution to Problem 1, part a.

Here we present three ways to test this identity: direct algebraic manipulation, hand writing the truth tables, and a MATLAB calculation.

Algebraic Manipulation

First we notice that the circuit in Figure 1-1 can be written as an algebraic expression:

$$\bar{A} + (\overline{A + B}). \quad (1-1)$$

Using de Morgan's Law on the parenthesized expression:

$$\bar{A} + (A \cdot \bar{B}). \quad (1-2)$$

Applying the distributive property for the *OR*:

$$(\bar{A} + A) \cdot (\bar{A} + \bar{B}). \quad (1-3)$$

The first parenthesized expression is always true, and can thus be ignored. We recognize de Morgan's Law again in the second parenthesized expression:

$$(\bar{A} + \bar{B}) = \overline{A \cdot B} \quad (1-4)$$

and the right hand side is simply the *NAND* gate. Other proofs exist that use the unnamed theorem.

Truth Table Analysis

We can also prove this identity by writing out the truth tables. By hand it looks like this, with all intermediate calculations:

A	B	\overline{A}	$\overline{A+B}$	$\overline{\overline{A+B}}$	$\overline{A+\overline{A+B}}$	$NAND$
0	0	1	1	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	1
1	1	0	1	0	0	0

Table 1-2: Truth table analysis of Problem 1

MATLAB Program

Using the following code, we can solve the problem in MATLAB:

```
% Start of file, filename testequality.m
%

A = [0 0 1 1];
B = [0 1 0 1];
Right = ~A | ~(~A | B);
Left = ~ A & B;
if Left == Right
    fprintf('The equality is true.\n');
else
    fprintf('The equality is false.\n');
end

%% End of file
```

This code gives the following output:

```
>> testequality
The equality is true.
```

Solution to Problem 1, part b.

We are interested in reducing the number of *OR* gates in the control circuit from Figure 1-1. We can, and have already shown so in the last step from our proof, see equation (1-4). The left hand side of (1-4) translates to the circuit in figure 1-2. We note that this is the same conclusion we would have reached by simple application of De Morgan's law to the *NAND* gate.

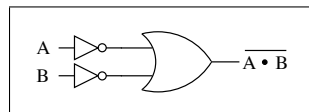


Figure 1-2: Logic circuit

Solution to Problem 2: Morse Encoder

We are asked to write a MATLAB function to encode text strings containing any of the characters *EITHMOC* in Morse code. The following function follows the procedure outlined in the problem statement.

Beginning of file morse.m

```
function [coded]=morse(tocode)
tocode=upper(tocode);
\% Define the table of codes

codes=char('.','. .','. . .','. . . .','-','-','-','-','-','-','-','-');

\% define the table of symbols
symbols='EISHTMOC';

\% replace word separations in the text by 4 spaces (note that word
\% separations are supposed to be 7 spaces, these will be provided by the
\% three spaces we add to separate characters)
tocode=strrep(tocode,' ',' ');
\% carry on replacement of each character by its code.
for i=1:length(symbols)
    \% note that we pad the code of each character with the three spaces for
    \% character separation.
    tocode=strrep(tocode,symbols(i),[deblank(codes(i,:)),' ']);
end
\% now construct the output string (tocode has already been modified,
\% we just need to copy it to the output variable)
coded=tocode;
disp(['Playing:',coded,'']);
\% now we can construct the sound vector. We shall go over each element
\% in coded, then replace dots by a unit waveform and dashes by a three time
\% unit waveform. In this case we take the waveform to be a a 1.3KHz
\% armonic with a unit duration of .17 seconds.
\% The choice of unit time corresponds to trying to fit an average of
\% 5 words per minute. This implies roughly 385 time units per minute,
\% and around 6 units per second, or .17 seconds per unit.
soundvector=[];
AT=.17; \% unit time in seconds
w=2*pi*1300; \% frequency.(in rads/second)
fs=8192;\% sampling rate (Matlab default is 8192)
for i=1:length(coded)
    if coded(i)=='.'
        soundvector= [soundvector,w*(0:(1/fs):(AT))];
    elseif coded(i)=='-'
        soundvector= [soundvector,w*(0:(1/fs):(3*AT))];
    else
        soundvector= [soundvector,0*(0:(1/fs):(AT))];
    end
end
end
sound(sin(soundvector),fs);

End of file morse.m
```

Alternatively, if you chose to use the function `unitsound`, you would just have to replace from the `disp` command onwards by the following piece of code:

