

ECE212/212s Lecture Notes

Lecture: Chapter 3

Dr. Clay Gloster, Jr.

Chapter 3 - Multilevel Combinational Logic

In this chapter we will cover:

- The conversion of AND/OR and OR/AND networks to NAND-NAND and NOR-NOR networks.
- Design strategies for implementing logic in more than two levels.
- Computer-aided design tools for multilevel combinational circuits.
- Time response in digital networks (Hazard-Free Design).

Design strategies for implementing logic in more than two levels.

Factoring of canonical SOP or POS expressions.

$$\begin{aligned} f(A, B, C, D, E, F, G) &= A D F + A E F + B D F + B E F + C D F + C E F + G \\ &= (A D + A E + B D + B E + C D + C E) F + G \\ &= (A D + B D + C D + (A + B + C) E) F + G \\ &= ((A + B + C) D + (A + B + C) E) F + G \\ &= ((A + B + C)(D + E)) F + G \end{aligned}$$

Then:

$$f(A, B, C, D, E, F, G) = X Y F + G$$

where,

$$\begin{aligned} X &= (A + B + C) \\ Y &= (D + E) \end{aligned}$$

Draw the multilevel circuit.

Compare with two-level circuit in the text.

Two level circuit for f contains six, (3-input AND gates); one, (7-input OR gates); a total of 7 gates; 19 literals; (See Figure 3.1a in text.)

Multilevel circuit for f contains one, (3-input OR gate); two, (2-input OR gates); and one, (3-input AND gate); a total of 4 gates; 9 literals; (See Figure 3.1b in text.)

Multilevel circuits typically require less gates/wires, but usually have worse delay because of increased levels of logic. But, technologies typically limit the maximum fanin(number of inputs) and maximum fanout(number of wires connected to a gate output) so most circuits are implemented using multilevel logic.

Most multilevel tools apply the following steps:

- **Decomposition:** Takes one Boolean function and replaces it with a collection of new subexpressions.
- **Extraction:** Similar to decomposition, but for multiple output functions. (Finds common subexpressions amongst different outputs.)
- **Factoring:** We factor out common subexpressions to attempt to reduce required gate count.
- **Substitution:** We substitute function G into a function F.
- **Collapsing:** The opposite of substitution. If

$$F = A + BC \quad \text{and} \quad G = A + B$$

then

$$F = G(A + C)$$

We Collapse G back into F.

- **Polynomial Division:** If

$$X = AC + AD + BC + BD + E \quad \text{and} \quad Y = A + B$$

X divided by Y

$$X = Y(C + D) + E$$

The conversion of AND/OR and OR/AND networks to NAND-NAND and NOR-NOR networks. Figure 1.

Demorgan's Law using Logic Gates

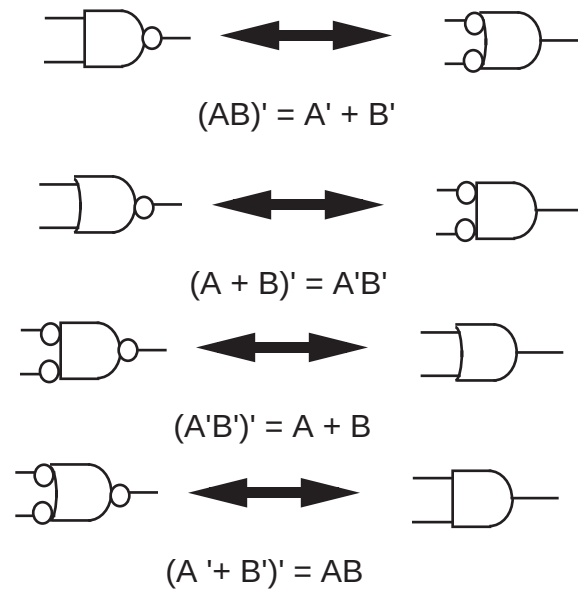


Figure 1: Demorgans Law for gate conversion

1. (*AND-OR/NAND-NAND Mappings*) Draw schematics for the following expressions, mapped into NAND-only networks. You may assume that literals and their complements are available.

a.

$$F(A, B, C) = \overline{A} \overline{B} + A \overline{C} \quad (1)$$

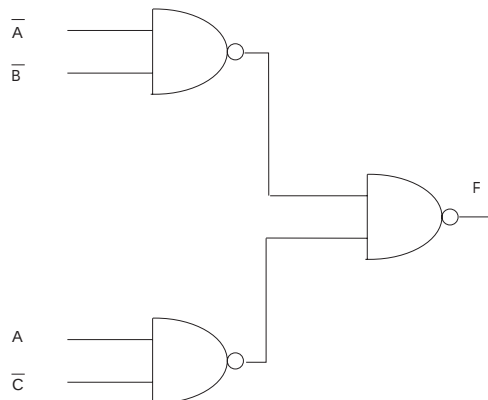


Figure 2: Schematic for Problem 1a.

b.

$$F(A, B, C) = \overline{\overline{A} \overline{B} + A \overline{C}} \quad (2)$$

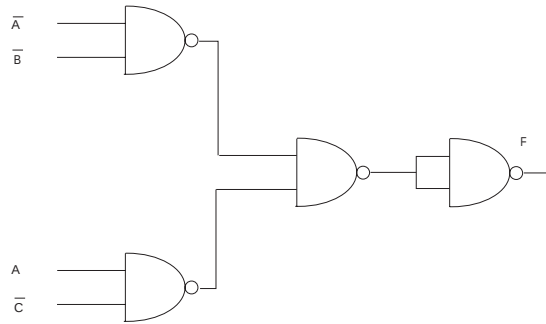


Figure 3: Schematic for Problem 1b.

2. (*OR-AND/NOR-NOR Mappings*) Draw schematics for the following expressions, mapped into NOR-only networks. You may assume that literals and their complements are available.

a.

$$F(A, B, C) = (\overline{A} + B + C)(A + \overline{C})(A + \overline{B}) \quad (3)$$

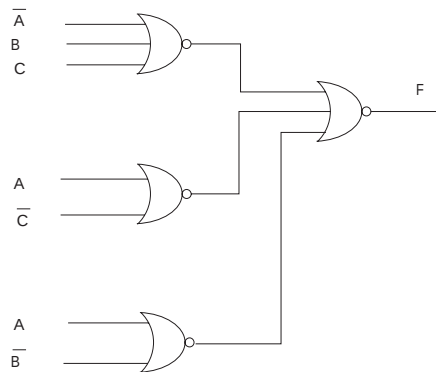


Figure 4: Schematic for Problem 2a.

b.

$$F(A, B, C) = A B C + A B + A C \quad (4)$$

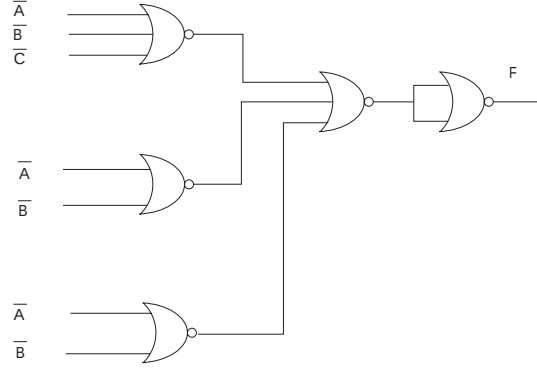


Figure 5: Schematic for Problem 2b.

3. (*Hazard-Free Design*) Implement the following Boolean functions using **hazard-free sum of products** circuits:

a.

$$F(A, B, C) = \overline{A} B + A \overline{C} \quad (5)$$

We first expand $F(A, B, C)$ into canonical form

$$\begin{aligned} F(A, B, C) &= \overline{A} B \overline{C} + \overline{A} B C + A \overline{B} \overline{C} + A B \overline{C} \\ &= \sum m(2, 3, 4, 6) \end{aligned} \quad (6)$$

Using a Karnaugh map (K-map), we group (2,3) and (4,6) into groups of 2. We must then add (2,6) to eliminate the static-1 hazard. The figure 6 shows the resulting K-map.

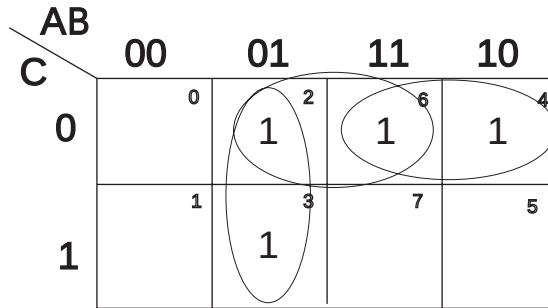


Figure 6: Problem 3a (Static-1 hazards).

It follows that we must add the product term $B \overline{C}$ to eliminate static-1 hazards.

$$F_{HF}(A, B, C) = \overline{A} B + A \overline{C} + B \overline{C} \quad (7)$$

We can check for static-0 hazards by verifying from the K-map that the above function covers the static-0 hazards.

$$F = (A + B)(\overline{A} + \overline{C}) \quad (8)$$

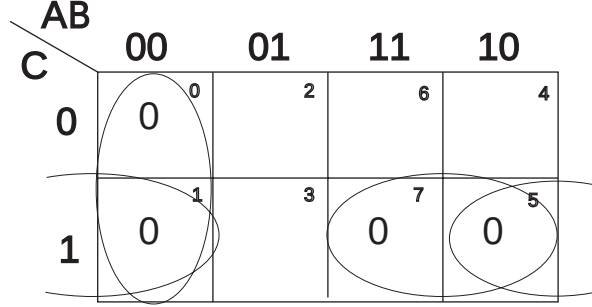


Figure 7: Karnaugh map for Problem 3a (static-0 hazards).

Using the K-map in figure 7 we group (0,1), (5,7) into groups of 2. We must then add (1,5).

$$F_{HF} = (A + B)(\overline{A} + \overline{C})(B + \overline{C}) \quad (9)$$

b.

$$F(A, B, C, D) = \sum m(0, 5, 7, 8, 10, 11, 12, 14) \quad (10)$$

The figure 8 gives a K-map for this function.

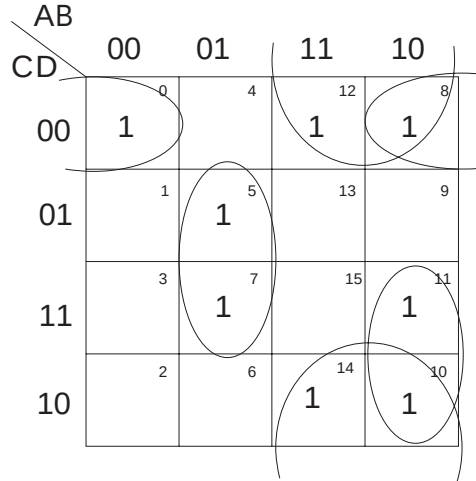


Figure 8: Karnaugh map for Problem 3b (static-1 hazards).

We obtain a group of 4 as (8,10,12,14) and groups of 2 as (0,8), (5,7) and (10,11). This is a minimized expression and there are no static-1 hazards. Thus, the static-1 hazard free expression is given by

$$F(A, B, C, D) = A \overline{D} + \overline{B} \overline{C} \overline{D} + \overline{A} B D + A \overline{B} C \quad (11)$$

As we did in problem 1a above, we can verify that this also covers the static-0 hazards by using $F(A, B, C, D)$.

$$F(A, B, C, D) = (\overline{A} + D)(B + C + D)(A + \overline{B} + \overline{D})(\overline{A} + B + \overline{C}) \quad (12)$$

Expanding, we obtain

$$\begin{aligned} F(A, B, C, D) &= \overline{A} B \overline{D} + \overline{A} \overline{B} C + \overline{A} C \overline{D} + \overline{A} \overline{B} D \\ &+ A B D + A \overline{C} D + \overline{B} \overline{C} D \end{aligned} \quad (13)$$

Table 1 shows the minterms covered by each product term in the above expression.

product term	covers
01-0	(4,6)
001-	(2,3)
0-10	(2,6)
00-1	(1,3)
11-1	(13,15)
1-01	(9,13)
-001	(1,9)

Table 1: Coverage of product terms for $\overline{F}$ in Problem 1b.

The figure 9 gives the corresponding K-map and shows that there are no static-0 hazards.

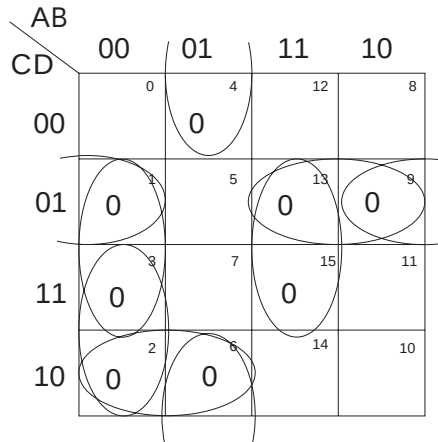


Figure 9: Problem 3b (Static-0 hazards).

Thus, the static hazard free sum of products expression is given by

$$F(A, B, C, D) = A \overline{D} + \overline{B} \overline{C} \overline{D} + \overline{A} B D + A \overline{B} C \quad (14)$$

Static-0 Hazard free product of sums expression is given by

$$F(A, B, C, D) = (A + \overline{B} + D)(\overline{A} + \overline{B} + \overline{D})(B + C + \overline{D})(A + B + \overline{C}) \quad (15)$$

$$F_{HF}(A, B, C, D) = (A + \overline{B} + D)(\overline{A} + \overline{B} + \overline{D})(B + C + \overline{D})(A + B + \overline{C})(A + B + \overline{D})(\overline{A} + C + \overline{D})(A + \overline{C} + D)$$

1. (*Multilevel Synthesis using misII*) Use the program espresso followed by the program misII found in the octtools locker (add octtools) to solve the following multilevel combinational circuit design problem. For assistance, please see the file misII_intro.ps found in the ece212_info locker. Use misII in the noninteractive mode and use the ttl library found in the script file /ncsu/ece212_info/lib/script.ttl. You must turn in

- The input file to espresso in pla/espresso format. (file.esp)
- The input file to misII(also the output file from espresso) in pla/espresso format. (file.min)
- The output file created using misII. (file.mis)
- A drawing of the multilevel circuit that was produced by misII.

Remember to run espresso first to minimize the two level truth table.

To run espresso use the command:

espresso file.esp > file.min

To run misII with the appropriate library use the command:

misII -f /ncsu/ece212_info/lib/script.ttl -t pla file.min > file.mis

where file.min is the name of the input file for misII.

and file.mis is the name of the output file for misII.

- a. Use the tools espresso and misII to find the multilevel circuit description for the full adder found in the text.

The input file for espresso, *full.esp* is shown below.

```
.i 3
.o 2
.p 8
.ilb a b ci
.ob sum co
000 00
001 10
```

```

010 10
011 01
100 10
101 01
110 01
111 11
.e

```

The minimized input file for misII *full.min* is shown below.

```

.i 3
.o 2
.ilb a b ci
.ob sum co
.p 7
100 10
010 10
001 10
111 10
-11 01
1-1 01
11- 01
.e

```

The output file created after running misII, *full.mis* is shown below.

```

.model full.min
.inputs a b ci
.outputs sum co
.default_input_arrival 0.00000 0.00000
.default_output_required 0.00000 0.00000
.default_input_drive 0.20000 0.20000
.default_output_load 1.00000
.names ci a b [144]
0-- 1
-0- 1
--0 1
.names a b [120]
1- 1
-1 1
.names ci [120] [88]
00 1
.names a b [142]
1- 1
-1 1
.names [142] ci [92]
11 1
.names a b [93]

```

```

11 1
.names [92] [93] co
1- 1
-1 1
.names [88] co [146]
1- 1
-1 1
.names [144] [146] sum
0- 1
-0 1
.end

```

The schematic for the circuit is shown in the figure 10.

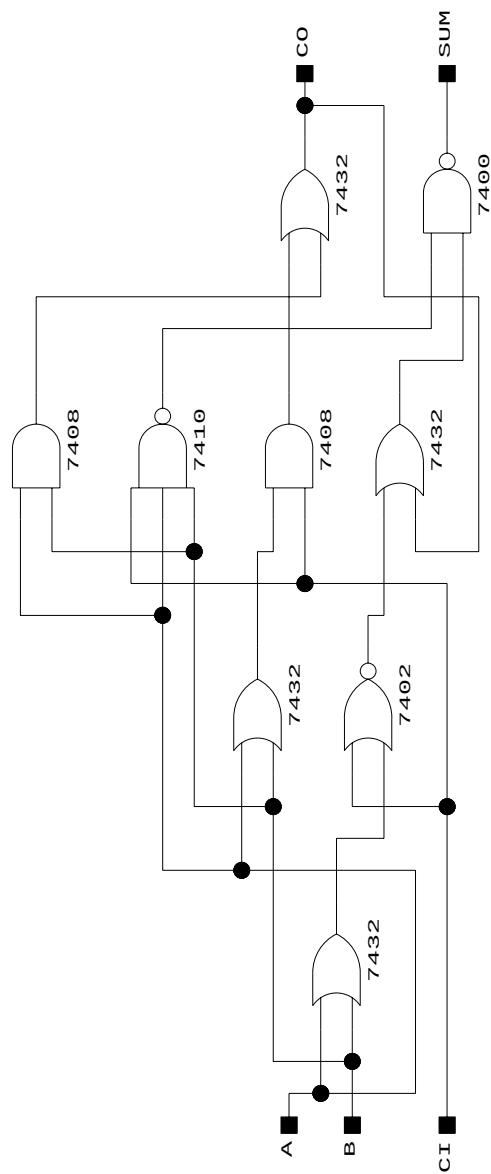


Figure 10: Multilevel Implementation of the full adder.

- b. You are to design an 8 input priority encoder. There are 8 inputs ($P_0 - P_7$) which correspond to requests from 8 different units. (A request is a logic high.) There are 3 outputs (S_2, S_1, S_0) which form the binary value of the unit which is to be serviced. (S_2 is the MSB) The unit connected to P_0 has lowest priority and the unit connected to P_7 has highest priority. If P_7 is asserted, then it is serviced regardless of the other requests. If P_7 is not asserted, and P_6 is asserted, then P_6 should get serviced regardless of the other requests. Similarly, P_i gets serviced only if units $P_{i+1} - P_7$ are not currently asserted (logic low) and P_i is asserted (logic high).

The input file for espresso, *pe.esp* is shown below.

```
.i 8
.o 3
.ilb P0 P1 P2 P3 P4 P5 P6 P7
.ob S2 S1 S0
.p 8
10000000 000
-1000000 001
--100000 010
---10000 011
----1000 100
-----100 101
-----10 110
-----1 111
.e
```

The minimized input file for misII *pe.min* is shown below.

```
.i 8
.o 3
.ilb P0 P1 P2 P3 P4 P5 P6 P7
.ob S2 S1 S0
.p 7
---1000- 011
-10-0-0- 001
--1-00-- 010
----1--- 100
-----10- 101
-----1- 110
-----1 111
.e
```

The output file created after running misII, *pe.mis* is shown below.

```
.model pe.min
.inputs P0 P1 P2 P3 P4 P5 P6 P7
.outputs S2 S1 S0
.default_input_arrival 0.00000 0.00000
.default_output_required 0.00000 0.00000
```

```

.default_input_drive 0.20000 0.20000
.default_output_load 1.00000
.names P4 P5 [127]
00 1
.names P6 [147]
0 1
.names P7 [150]
0 1
.names [127] [147] [150] S2
0-- 1
-0- 1
--0 1
.names P2 [139]
0 1
.names S2 [139] [96]
00 1
.names P3 [141]
0 1
.names S2 [141] [99]
00 1
.names P7 [99] [131]
1- 1
-1 1
.names [96] [131] [133]
1- 1
-1 1
.names P6 [133] S1
1- 1
-1 1
.names P1 [137]
0 1
.names S2 [137] [94]
00 1
.names P5 [94] [91]
00 1
.names [91] S1 [92]
00 1
.names [92] [131] S0
1- 1
-1 1
.end

```

The schematic for the circuit is shown in figure 11.

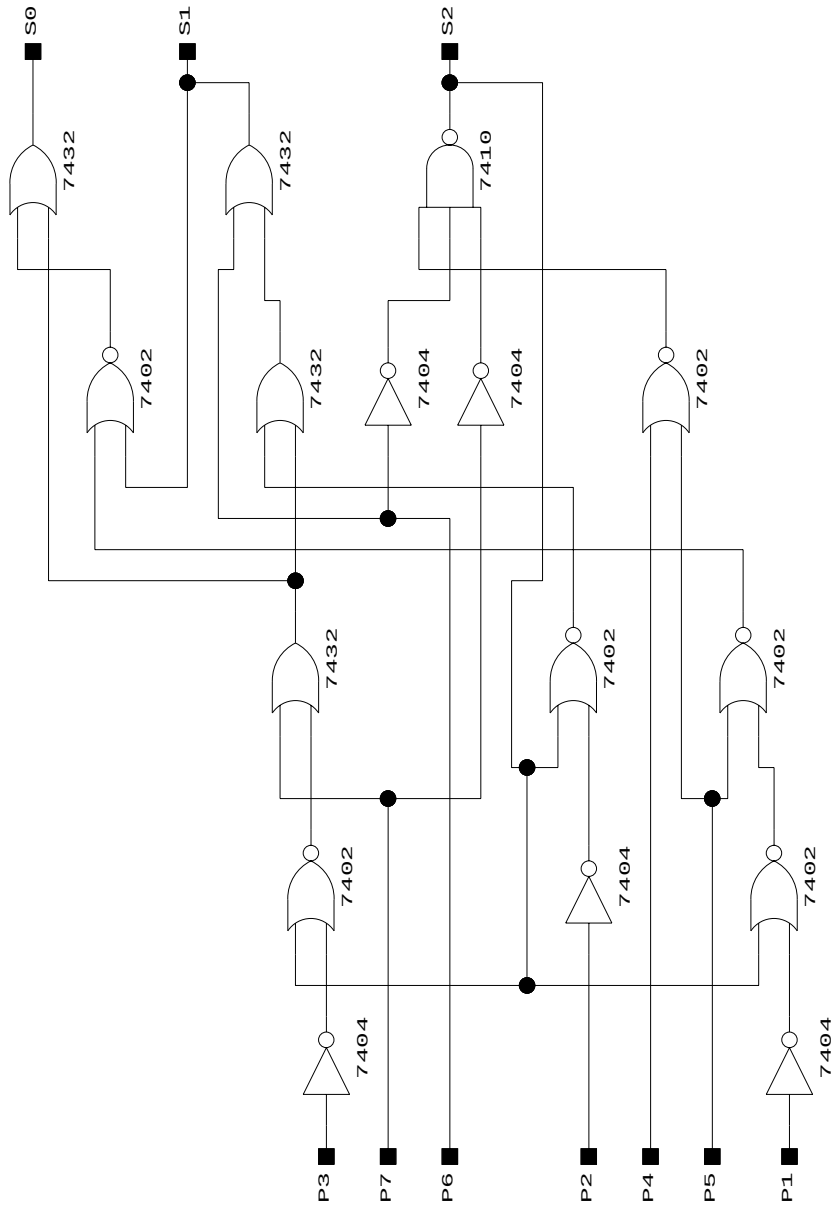


Figure 11: Multilevel Implementation of the priority encoder.