EE 2720

Handout #13

• Combinational-Circuit Synthesis

Definition of synthesis

— Given a problem statement:
- Design the circuit.
- Obtain the logic diagrams

To become more specific.
— Given a word description of a problem:
- The number of input and output variables is determined and variable names are assigned to them.
- The truth table that defines the required relationship between inputs and outputs is derived.
- The simplified Boolean function for each output is obtained.
- Logic diagrams are drawn.

Example 1: Design a circuit to convert a 3-bit binary number into a 3-bit Gray code number.

Answer: We will use 3 inputs labeled A, B and C to represent the 3-bit binary number. We will use 3 outputs labeled X, Y and Z to represent the 3-bit Gray code number. The truth table that describes the relationship between inputs and outputs is shown on the next page.

A B C	X Y Z
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 1
0 1 1	0 1 0
1 0 0	1 1 0
1 0 1	1 1 1
1 1 0	1 0 1
1 1 1	1 0 0

Table 1; (truth table for problem of example 1).

Note: I showed you how to construct a 3-bit Gray code in one of my earlier handouts.

From the above truth table we can easily get the following logic equations for X, Y, Z

$X = A \cdot B' \cdot C' + A \cdot B' \cdot C + A \cdot B \cdot C' + A \cdot B \cdot C =$

$= A \cdot (B' \cdot C' + B' \cdot C + B \cdot C' + B \cdot C) =$

$= A \cdot [B' \cdot (C' + C) + B \cdot (C' + C)] =$

$= A \cdot (B' \cdot 1 + B \cdot 1) = A \cdot (B' + B) = A \cdot 1 = A$ or

$X = A$ (1); (Note: You can get $X = A$ just by inspection)

$Y = A' \cdot B \cdot C' + A' \cdot B \cdot C + A \cdot B' \cdot C' + A \cdot B' \cdot C =$

$= A' \cdot B \cdot (C' + C) + A \cdot B' \cdot (C' + C) = A' \cdot B \cdot 1 + A \cdot B' \cdot 1$

$= A' \cdot B + A \cdot B' = A \oplus B$ or

$Y = A \oplus B$ (2)

$Z = A' \cdot B' \cdot C + A' \cdot B \cdot C' + A \cdot B' \cdot C + A \cdot B \cdot C' =$

$= B' \cdot C \cdot (A' + A) + B \cdot C' \cdot (A' + A) = B' \cdot C \cdot 1 + B \cdot C' \cdot 1 =$

$= B' \cdot C + B \cdot C' = B \oplus C$ or

$Z = B \oplus C$ (3).

The above equations (1), (2), (3) suggest the logic circuit shown on next page.

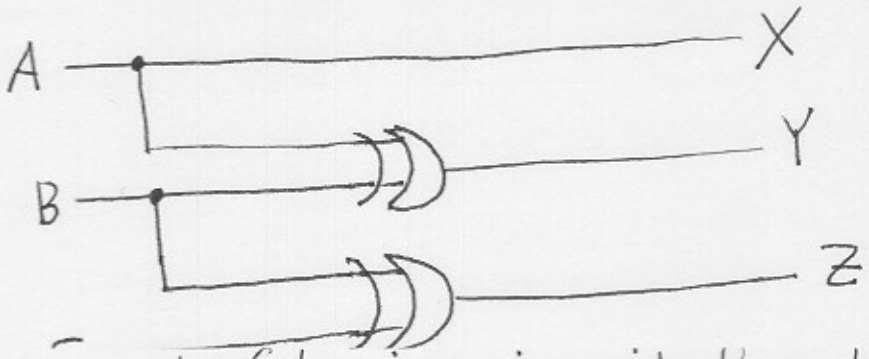

Figure 1; (logic circuit for the binary to Gray code converter).

- Description of a logic function using the English language:

Sometimes we describe a logic function using the English language and the "and", "or" and "not" statements. Here is an example of an alarm system.

Example 2: Provide the logic function and some logic circuits corresponding to the following English statement that describes an alarm system:

The ALARM output is 1 if the PANIC input is 1 or if (ENABLE input is 1 and the EXITING input is 0 and the house is not SECURE). The house is SECURE if the WINDOW and DOOR and GARAGE inputs are all 1.

Answer: we first provide the logic equation for the output ALARM. We have

$ALARM = PANIC + ENABLE \cdot EXITING' \cdot SECURE'$ (1).

$SECURE = WINDOW \cdot DOOR \cdot GARAGE$ (2)

From (1) and (2) we get

$$ALARM = PANIC + ENABLE \cdot EXITING' \cdot (WINDOW \cdot DOOR \cdot GARAGE)' =$$

$$= PANIC + ENABLE \cdot EXITING' \cdot (WINDOW' + DOOR' + GARAGE') =$$

$$= PANIC + ENABLE \cdot EXITING' \cdot WINDOW' + ENABLE \cdot EXITING' \cdot DOOR' + ENABLE \cdot EXITING' \cdot GARAGE' \quad (3).$$

Eq. (3) above suggests the AND-OR logic circuit shown in figure 2 below.

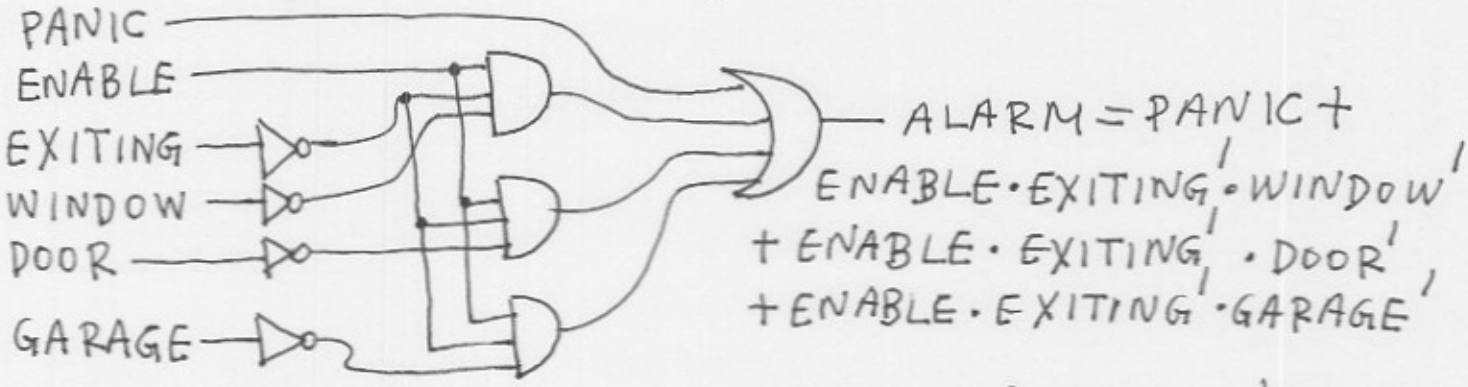

Figure 2; (AND-OR logic circuit for the alarm system).

Note: We can factor the expression of ALARM provided by eq. (3) to get an equivalent OR-AND circuit, equivalent to this of fig. 2; (doing this algebraically might not be that easy because there are 6 variables; you can do it graphically and it is trivial; (see example 1 of handout #11)). Besides the AND-OR and OR-AND realizations just mentioned, we can also get equivalent realizations

for ALARM function using only NAND gates, using only OR and NAND gates, using only NOR and OR gates, using only NOR gates, using only AND and NOR gates or using only NAND and AND gates; (this is true for any logic circuit; we have done this before; see handout # 10 for details).

Example 3: Design a logic circuit that accepts as input a 4-bit number $A=(a_3a_2a_1a_0)_2$ and produces an output F that is true (1) if the number A is prime. Otherwise F=0.

Reminder: A number is called prime if it is only divided by 1 and itself. If it is not prime, it is then called composite.

Answer: With 4 bits (we have a 4-bit input $A=a_3a_2a_1a_0$) we can represent numbers in the range 0, 1, 2, ..., 15. In this range, the prime numbers are 1, 2, 3, 5, 7, 11, 13. The truth table that provides F as a function of a_3, a_2, a_1, a_0 is given below; a_3 is MSB, a_0 is LSB:

a_3	a_2	a_1	a_0	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Table 2: (truth table for the prime number detector).

We now provide the canonical sum for F. We get it from the truth table. The canonical sum for F is

$$F = \Sigma_{a_3, a_2, a_1, a_0}(1, 2, 3, 5, 7, 11, 13) = a_3' \cdot a_2' \cdot a_1' \cdot a_0 + a_3' \cdot a_2' \cdot a_1 \cdot a_0' + a_3' \cdot a_2' \cdot a_1 \cdot a_0 + a_3' \cdot a_2 \cdot a_1' \cdot a_0 + a_3' \cdot a_2 \cdot a_1 \cdot a_0 + a_3 \cdot a_2' \cdot a_1 \cdot a_0 + a_3 \cdot a_2 \cdot a_1' \cdot a_0 \quad (1)$$

Eq. (1) above suggests the AND-OR logic circuit for the prime number detector shown in figure 3 below:

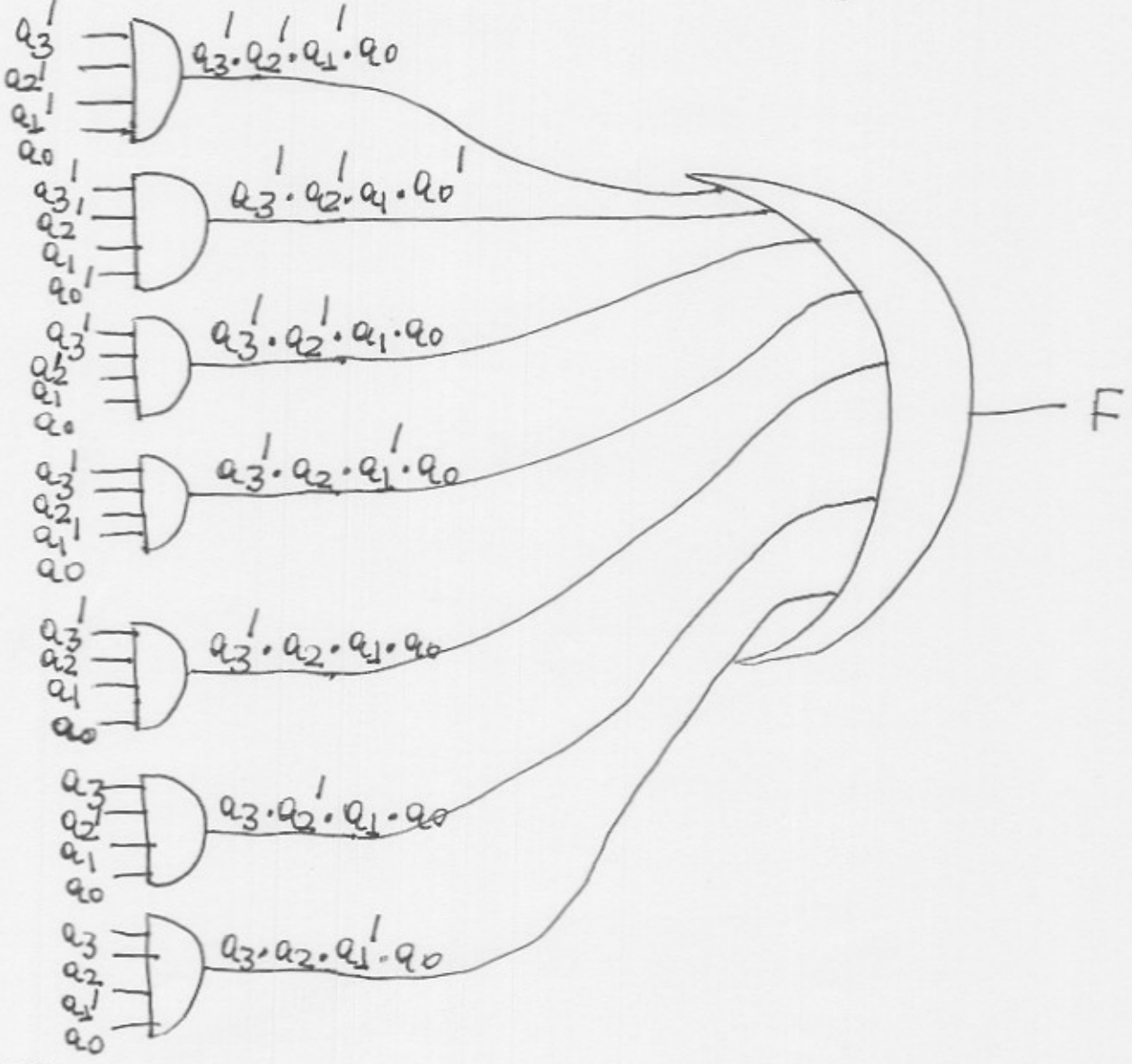

Figure 3; (AND-OR logic circuit for the prime number detector; I just don't show the NOT gates here; couldn't do it).

• Combinational-Circuit Minimization

Canonical sum and canonical product expressions are usually uneconomical to realize; (recall that the realization of fig. 3 of the previous page is based on a canonical sum (eq. (1) of previous page)). The canonical sum and product expressions are uneconomical to realize because the number of possible minterms or maxterms (and therefore gates) grows exponentially with the number of input variables. We minimize a combinational circuit by reducing the number and size of gates involved in the design.

The minimization methods used to reduce the cost of a two-level circuit (AND-OR, OR-AND, NAND-NAND, NOR-NOR etc...) are:

- Minimize the number of first-level gates.
- Minimize the number of inputs on each first-level gate.
- Minimize the number of inputs on the second-level gate. This is a side effect of the first reduction.

Note: The above minimization methods do not consider the cost of input inverters. They assume that both uncomplemented and complemented versions of all input variables are available. This is ok for PLD (Programmable Logic Devices) -based designs; PLDs have both uncomplemented and complemented versions of all input variables available for free.

Note: Most minimization methods are based on generalizations of theorems (T10) and (T10'). (T10), (T10') state:

$X \cdot Y + X \cdot Y' = X$ (T10).

$(X+Y) \cdot (X+Y') = X$ (T10').

The generalizations of (T10), (T10') are provided below by equations (2), (3) respectively:

given product term $\cdot Y$ + given product term $\cdot Y'$ = given product term. (2)

(given sum term $+ Y$) $\cdot$ (given sum term $+ Y'$) = given sum term. (3)

Proof of (2): It is trivial.

given product term $\cdot Y$ + given product term $\cdot Y'$ =
given product term $\cdot (Y+Y')$ = given product term $\cdot 1$ =
given product term.

Proof of (3):

Just apply principle of duality on eq. (2) above and you are done!!

Implications of eqs. (2), (3): If two product or sum terms differ only in the complementing or not of one variable, we can combine them into a single term with one less variable. So we save one gate and the remaining gate has one fewer input.

Let's now apply eq. (2) above to minimize F of eq (1) of page 6 (the output of the prime number detector).

We have: $F = \sum_{a_3, a_2, a_1, a_0} (1, 2, 3, 5, 7, 11, 13) =$

$$= \sum_{a_3,a_2,a_1,a_0}(1,3,5,7,2,11,13) = a_3'\cdot a_2'\cdot a_1'\cdot a_0 +$$
$$a_3'\cdot a_2'\cdot a_1\cdot a_0 + a_3'\cdot a_2\cdot a_1'\cdot a_0 + a_3'\cdot a_2\cdot a_1\cdot a_0 +$$
$$+ a_3'\cdot a_2'\cdot a_1\cdot a_0' + a_3\cdot a_2'\cdot a_1\cdot a_0 + a_3\cdot a_2\cdot a_1'\cdot a_0 =$$
$$= (a_3'\cdot a_2'\cdot a_0\cdot a_1' + a_3'\cdot a_2'\cdot a_0\cdot a_1) +$$
$$+ (a_3'\cdot a_2\cdot a_0\cdot a_1' + a_3'\cdot a_2\cdot a_0\cdot a_1) +$$
$$+ a_3'\cdot a_2'\cdot a_1\cdot a_0' + a_3\cdot a_2'\cdot a_1\cdot a_0 + a_3\cdot a_2\cdot a_1'\cdot a_0$$
$$= a_3'\cdot a_2'\cdot a_0 + a_3'\cdot a_2\cdot a_0 + a_3'\cdot a_2'\cdot a_1\cdot a_0' +$$
$$+ a_3\cdot a_2'\cdot a_1\cdot a_0 + a_3\cdot a_2\cdot a_1'\cdot a_0 =$$
$$= (a_3'\cdot a_0\cdot a_2' + a_3'\cdot a_0\cdot a_2) + a_3'\cdot a_2'\cdot a_1\cdot a_0' +$$
$$+ a_3\cdot a_2'\cdot a_1\cdot a_0 + a_3\cdot a_2\cdot a_1'\cdot a_0 =$$
$$= a_3'\cdot a_0 + a_3'\cdot a_2'\cdot a_1\cdot a_0' + a_3\cdot a_2'\cdot a_1\cdot a_0 +$$
$$+ a_3\cdot a_2\cdot a_1'\cdot a_0 \quad (4)$$

Equation (4) above suggests the simplified AND-OR logic circuit for the prime number detector shown in figure 4 below:

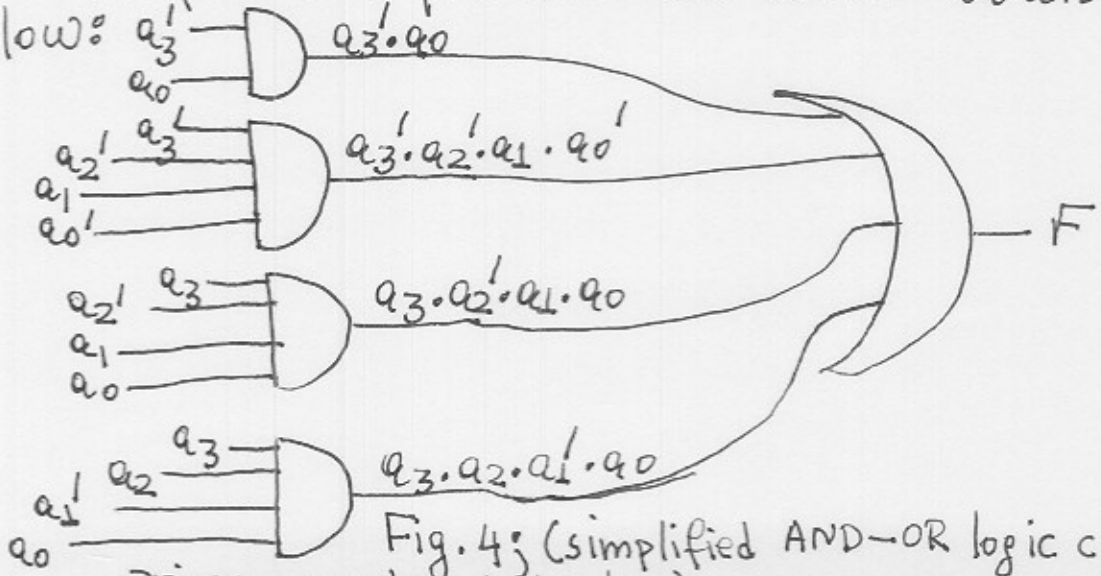

Fig. 4; (simplified AND-OR logic circuit for the prime number detector).

Implication of the previous minimization: Comparing the circuits of figures 3 and 4 we see that this of fig. 4 has three fewer gates in the first level (as compared to this of fig. 3) and one of the remaining gates has two fewer inputs (2 versus 4). Regarding the second level, the circuit of fig. 4 relies on a 4-input OR gate while this of fig. 3 relies on a 7-input OR gate. All this based on the simple eq. (2) of page 8!!

Let's now provide an OR-AND logic circuit for the prime number detector. We have to transform F of eq (1) of page 6 into product-of-sums form. The way I'll do this is by writing the canonical product for F. We have:

$$F = \sum_{a_3,a_2,a_1,a_0}(1,2,3,5,7,11,13) =$$

$$= \prod_{a_3,a_2,a_1,a_0}(0,4,6,8,9,10,12,14,15) =$$

$$= (a_3+a_2+a_1+a_0)\cdot(a_3+a_2'+a_1+a_0)\cdot(a_3+a_2'+a_1'+a_0)\cdot$$
$$(a_3'+a_2+a_1+a_0)\cdot(a_3'+a_2+a_1+a_0')\cdot(a_3'+a_2+a_1'+a_0)\cdot$$
$$(a_3'+a_2'+a_1+a_0)\cdot(a_3'+a_2'+a_1'+a_0)\cdot(a_3'+a_2'+a_1'+a_0') \quad (5)$$

Equation (5) above suggests the non-simplified OR-AND logic circuit for the prime number detector shown in figure 5 on the next page.

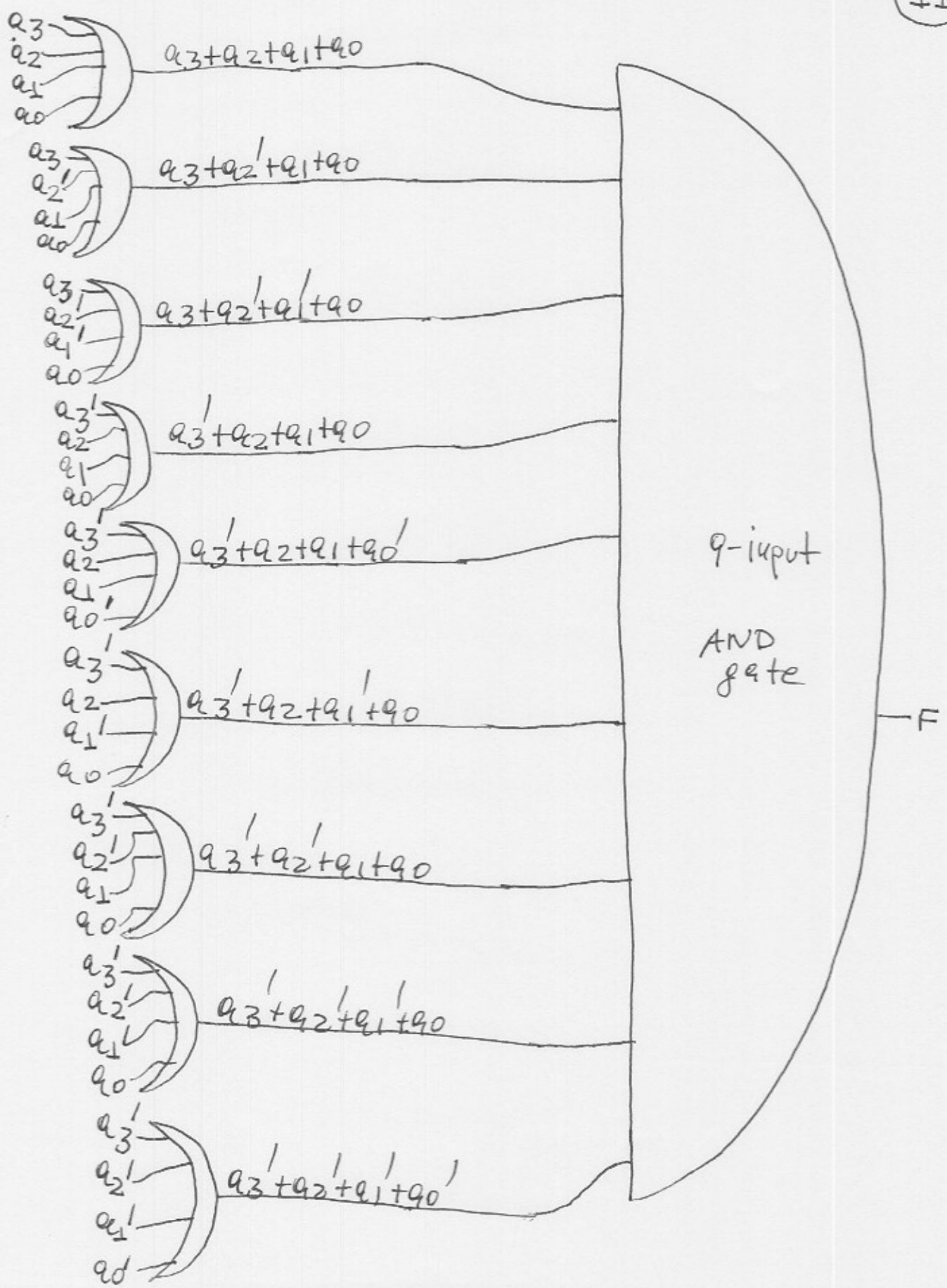

Figure 5: (non-simplified OR-AND logic circuit for the prime number detector).

Let's now apply eq. (3) (see page 8 for eq. (3)) to minimize F of eq (5) of page 10. We have:

$$F=\Pi_{a_3,a_2,a_1,a_0}(0,4,6,8,9,10,12,14,15)=$$
$$=\Pi_{a_3,a_2,a_1,a_0}(0,4,8,10,12,14,6,9,15)=$$
$$=(a_3+a_2+a_1+a_0)\cdot(a_3+a_2'+a_1+a_0)\cdot$$
$$(a_3'+a_2+a_1+a_0)\cdot(a_3'+a_2+a_1'+a_0)\cdot$$
$$(a_3'+a_2'+a_1+a_0)\cdot(a_3'+a_2'+a_1'+a_0)\cdot$$
$$(a_3+a_2'+a_1'+a_0)\cdot(a_3'+a_2+a_1+a_0')\cdot(a_3'+a_2'+a_1'+a_0')=$$
$$=[(a_3+a_1+a_0+a_2)\cdot(a_3+a_1+a_0+a_2')]\cdot$$
$$[(a_3'+a_2+a_0+a_1)\cdot(a_3'+a_2+a_0+a_1')]\cdot$$
$$[(a_3'+a_2'+a_0+a_1)\cdot(a_3'+a_2'+a_0+a_1')]\cdot$$
$$(a_3+a_2'+a_1'+a_0)\cdot(a_3'+a_2+a_1+a_0')\cdot(a_3'+a_2'+a_1'+a_0')=$$
$$=(a_3+a_1+a_0)\cdot(a_3'+a_2+a_0)\cdot(a_3'+a_2'+a_0)\cdot$$
$$(a_3+a_2'+a_1'+a_0)\cdot(a_3'+a_2+a_1+a_0')\cdot(a_3'+a_2'+a_1'+a_0')$$
$$=(a_3+a_1+a_0)\cdot[(a_3'+a_0+a_2)\cdot(a_3'+a_0+a_2')]\cdot$$
$$(a_3+a_2'+a_1'+a_0)\cdot(a_3'+a_2+a_1+a_0')\cdot(a_3'+a_2'+a_1'+a_0')$$
$$=(a_3+a_1+a_0)\cdot(a_3'+a_0)\cdot(a_3+a_2'+a_1'+a_0)\cdot$$
$$(a_3'+a_2+a_1+a_0')\cdot(a_3'+a_2'+a_1'+a_0') \quad (6).$$

Equation (6) above suggests the following simplified OR–AND logic circuit for the prime number detector shown in figure 6 on the next page.

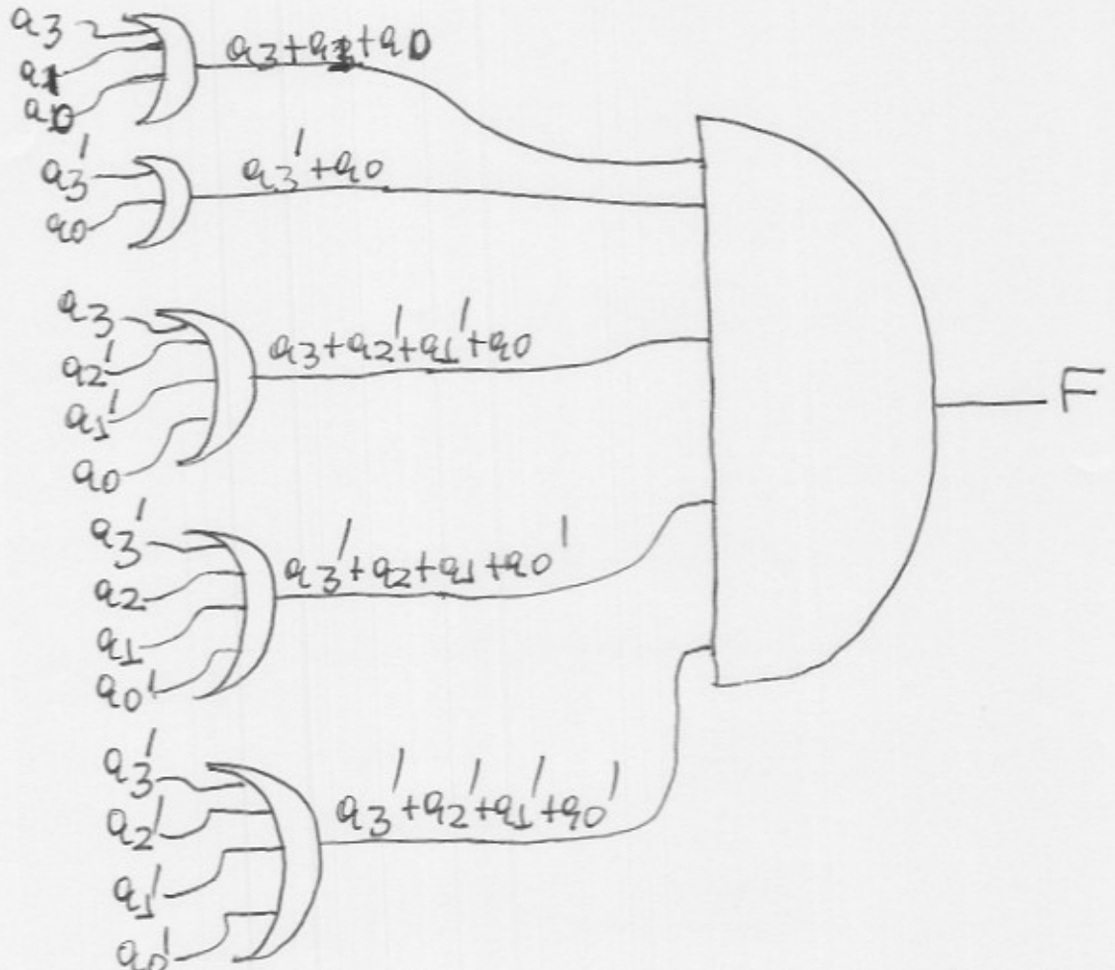

Figure 6; (simplified OR-AND logic circuit for the prime number detector).

• Enjoy the simplifications: Comparing the logic circuits of figures 5, 6, you can see that this of fig. 5 relies on nine 4-input OR gates in the first level and one 9-input AND gate in the second level, while the circuit of fig. 6 relies only on three 4-input OR gates, one 3-input OR gate and one 2-input OR gate in the first level and one 5-input AND gate in the second level. And all this based on a very simple theorem (this of eq. (3) on page 8).

If you don't believe me that theorem of eq. (3) is simple, here is a trivial proof of it. I told

you on page 8 that you can demonstrate the validity of this theorem by using the principle of duality applied on eq. (2) of page 8. If you want another proof here it is:

$(\text{given sum term} + Y) \cdot (\text{given sum term} + Y') =$

$= \text{given sum term} \cdot \text{given sum term} +$

$\text{given sum term} \cdot Y' + Y \cdot \text{given sum term} + \cancel{Y \cdot Y} \to 0$

$= \text{given sumterm} \cdot \text{given sumterm} + \text{given sumterm} \cdot Y'$

$+ \text{given sumterm} \cdot Y = \text{given sumterm} + \text{given sumterm} \cdot Y'$

$+ \text{given sumterm} \cdot Y = \text{given sumterm} \cdot 1 +$

$+ \text{given sumterm} \cdot Y' + \text{given sumterm} \cdot Y =$

$\text{given sumterm} \cdot (1 + Y' + Y) = \text{given sumterm} \cdot 1 =$

$\text{given sumterm}.$

Note: We already got simplified AND-OR and OR-AND logic circuits for the prime number detector; (see figures 4 and 6 on pages 9 and 13 respectively). Besides the simplified AND-OR and simplified OR-AND realizations, you can also get equivalent simplified realizations for the prime number detector using only NAND gates, using only OR and NAND gates, using only NOR and OR gates, using only NOR gates, using only AND and NOR gates or using only NAND and AND gates; (see handout #10 for details).

Note: For the simplified realizations using only NAND or only NOR gates, you can very easily get them by using the graphical approach. Just start from figures 4 or 6 respectively and insert bubbles.

Note: More examples on combinational circuit synthesis (design I mean) will most probably be provided in future handouts. I know you enjoy the subject!