5 Combinational logic circuits – optimization and synthesis

5.1 Objectives

The two classes characterizing completely, and incompletely specified functions are studied. Several strategies for function optimization are enumerated. Next, commonly used functions are exemplified, optimized and synthesized using logic gates. The following combinational circuits are implemented: the BCD–Excess 3 converter, the 2-bit arithmetic comparator and the summation units.

5.2 Theoretical considerations

According to their structure, the circuits can be *combinational* or *sequential*. The outputs of the *combinational* circuits represent functions which are strictly dependent on input values. When implemented with logic gates, they can be recognized for their lack of backward connectivity, from output pins to input pins. The functions implemented by the outputs can be *completely specified* or *incompletely specified*.

A function is *incompletely specified* when the output is unknown for a subset of input combinations of values. In such cases, the common notation used for the output is the symbol X or Ø. For instance, a circuit expecting 4-bit decimal digits on its input will have an undefined behavior for binary values in the range 1010÷1111, because they do not represent decimal digits. This is the case for the BCD–Excess 3 converter with the behavior defined in Table 5. 1.

Table 5. 1 Truth table of the BCD–Excess 3 converter

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

5.2.1 Techniques for optimizing logic circuits

The optimization process aims to generate a new circuit with similar functionality, but with a simpler structure. The goal is to decrease the number of operations and the number of variables, which will decrease the necessary logic gates, their interconnections and the length of the wires. Next, we enumerate several optimization strategies commonly used when synthesizing circuits:

a) **The use of additional variables**: For instance, the set of boolean expressions $\begin{cases} a = (c \odot d) \cdot (e \oplus f) \\ b = \overline{c \odot d} + (e \oplus f) \end{cases}$ can be rewritten using the auxiliary variables $m = c \odot d$ and $n = e \oplus f$. The result will be a new set of expressions: $\begin{cases} a = m \cdot n \\ b = \overline{m} + n \end{cases}$.

b) **Applying the laws from boolean algebra**: The laws of boolean algebra are largely used in circuit optimization. Some of the most common are: $\overline{\overline{x}} = x$, $x + 1 = 1$, $x \cdot 1 = x$, $x + \overline{x} = 1$ or $x \cdot \overline{x} = 0$, etc. Considering the expression $x \cdot y \cdot z + x \cdot y \cdot \overline{z}$ and applying the distributive law we obtain the following equivalent formulations: $x \cdot y \cdot z + x \cdot y \cdot \overline{z} = x \cdot y \cdot (z + \overline{z}) = x \cdot y \cdot 1 = x \cdot y$.

c) **Adopting general scale techniques**, such as minimization based on Karnaugh maps or the Quine-McCluskey algorithm.

5.2.2 Boolean functions synthesis

The synthesis starts with a graphical representation of the implemented functions using the truth tables or the Karnaugh maps. The representation is converted to optimized boolean expressions, which are implemented using physical circuits. The following sections will describe several synthesis examples.

5.2.2.1 The BCD–Excess 3 converter

The set of outputs W, X, Y, Z in the truth Table 5. 1 describe the conversion from BCD (Binary Coded Decimal) code to Excess 3 code. Their corresponding Karnaugh maps are presented in Figure 5. 1.

Function optimization to Minimal Disjunctive Normal Form (MDNF): Looking at the Karnaugh map representation the goal is to identify groups of neighboring 1s and X cells. The groups must have rectangular shapes, and their number of cells must be a power of 2. It should be considered that, for a Karnaugh map, the first and last rows as well as the first and last columns are adjacent (neighbors). While all 1s should be part of a group, the size of the identified groups must be maximized, and their number minimized. Groups solely based on X cells must be avoided. A 1 or an X cell can be part of several groups, if needed to increase their size.

A term will result for each group by applying the operator AND over the variables constant within the group. The variables will be inverted if their value is 0, or not, otherwise. The Minimal Disjunctive Normal Form is obtained by ORing the terms.

Function optimization to Minimal Conjunctive Normal Form (MCNF): The technique is symmetrical to Minimal Disjunctive Normal Form, because it aims to identify groups of 0s, based on the same principles. All 0s must be grouped. The groups can contain X cells, as well. The term resulted from a group apply an OR operation solely on the variables constant within the group. The variables are inverted if their value is 1, and not inverted, otherwise. The Minimal Conjunctive Normal Form is obtained by applying AND over these terms.

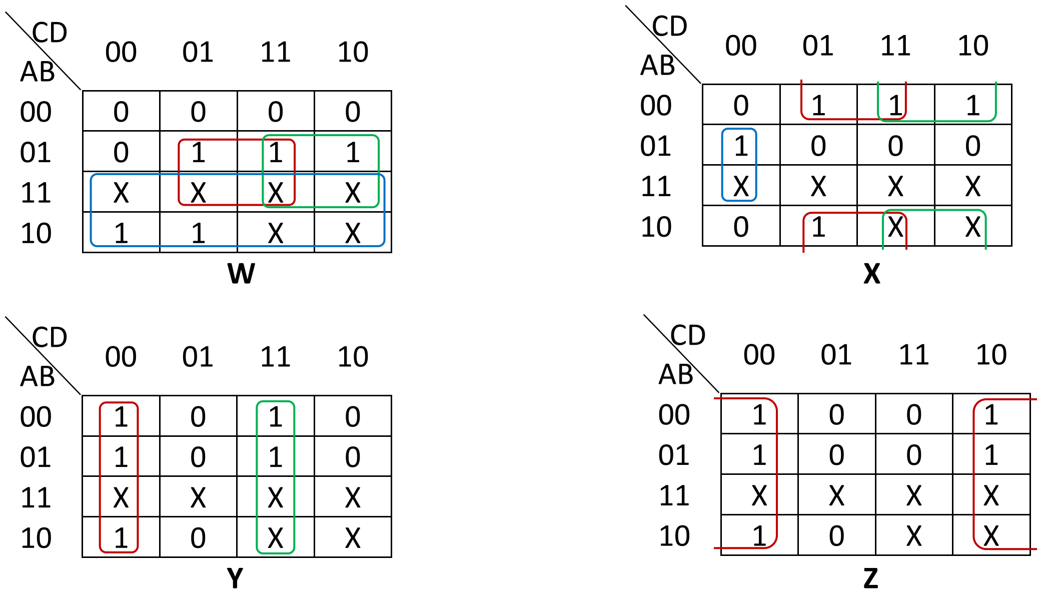


Figure 5. 1 Karnaugh maps for the BCD–Excess 3 converter outputs

The Minimal Disjunctive Normal Forms resulted from groups identified in Figure 5. 1 will reveal the following expressions:

$$\begin{aligned} W &= B \cdot D + B \cdot C + A \\ X &= \overline{B} \cdot D + \overline{B} \cdot C + B \cdot \overline{C} \cdot \overline{D} \\ Y &= \overline{C} \cdot \overline{D} + C \cdot D \\ Z &= \overline{D} \end{aligned} \qquad (5.\ 1)$$

5.2.2.2 The 2-bit unsigned comparator

The 2-bit unsigned comparator have the inputs $N_1=A_1A_0$ and $N_2=B_1B_0$. The outputs are F_1, F_2 and F_3. Their functionality respects the following rules:

- $F_1=1$, if $N_1>N_2$ and $F_1=0$, otherwise;
- $F_2=1$, if $N_1=N_2$ and $F_2=0$, otherwise;
- $F_3=1$, if $N_1<N_2$ and $F_3=1$, otherwise.

The block diagram of the 2-bit unsigned comparator is presented in the following figure.

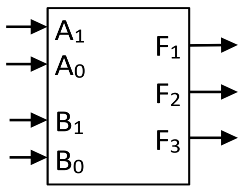
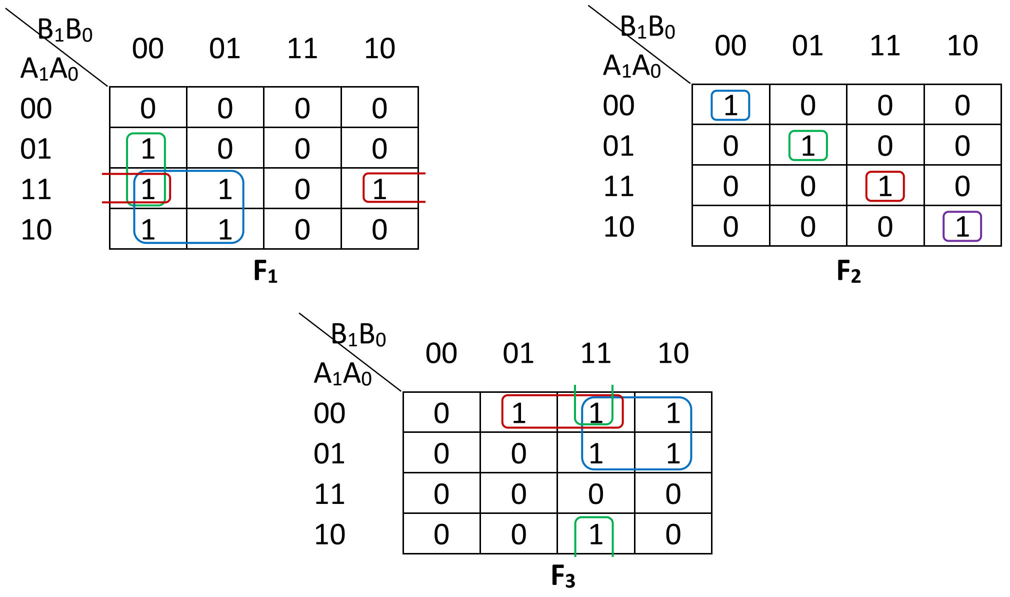


Figure 5. 2 Block diagram of the 2-bit unsigned comparator

As noticed in the previous lab work, the 2-bit unsigned comparator can be implemented using 1-bit unsigned comparators and additional logic gates. An alternative solution is fully based on logic gates. It uses the Minimal Disjunctive Normal Forms of the outputs, as functions of the 4 variables: A_1, A_0, and B_1, B_0, respectively. The Karnaugh maps can be filled according to simplified rules. For F_1, the 1s in the Karnaugh map are matched where $A_1A_0>B_1B_0$. Similarly, in the Karnaugh map for F_3, the 1s indicate $A_1A_0<B_1B_0$, while for F_2, they indicate $A_1A_0=B_1B_0$. The rest of cells are 0ed. The diagrams are highlighted below:



A_1A_0 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

F_1

A_1A_0 \ B_1B_0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

F_2

A_1A_0 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

F_3

Figure 5. 3 Karnaugh maps for the 2-bit unsigned comparator outputs

Based on groups identified in the Karnaugh maps, the resulting Minimal Disjunctive Normal Forms for F_1, F_2 and F_3 are:

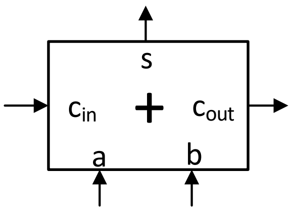
$$\begin{gathered} F_1 = A_1 \cdot \overline{B_1} + A_0 \cdot \overline{B_1} \cdot \overline{B_0} + A_1 \cdot A_0 \cdot \overline{B_0} \\ F_2 = \overline{A_1} \cdot \overline{A_0} \cdot \overline{B_1} \cdot \overline{B_0} + \overline{A_1} \cdot A_0 \cdot \overline{B_1} \cdot B_0 + A_1 \cdot A_0 \cdot B_1 \cdot B_0 + A_1 \cdot \overline{A_0} \cdot B_1 \cdot \overline{B_0} \\ F_3 = \overline{A_1} \cdot B_1 + \overline{A_0} \cdot B_1 \cdot B_0 + \overline{A_1} \cdot \overline{A_0} \cdot B_0 \end{gathered} \qquad (5.\,2)$$

Using the boolean algebra laws, F_2 can be further simplified as:

$$\begin{aligned} F_2 &= \overline{A_1} \cdot \overline{B_1} \cdot (\overline{A_0} \cdot \overline{B_0} + A_0 \cdot B_0) + A_1 \cdot B_1 \cdot (\overline{A_0} \cdot \overline{B_0} + A_0 \cdot B_0) = \\ &= (\overline{A_1} \cdot \overline{B_1} + A_1 \cdot B_1) \cdot (A_0 \odot B_0) = (\mathbf{A_1} \odot \mathbf{B_1}) \cdot (\mathbf{A_0} \odot \mathbf{B_0}) \end{aligned} \qquad (5.\,3)$$

5.2.2.3 The 1-bit full adder

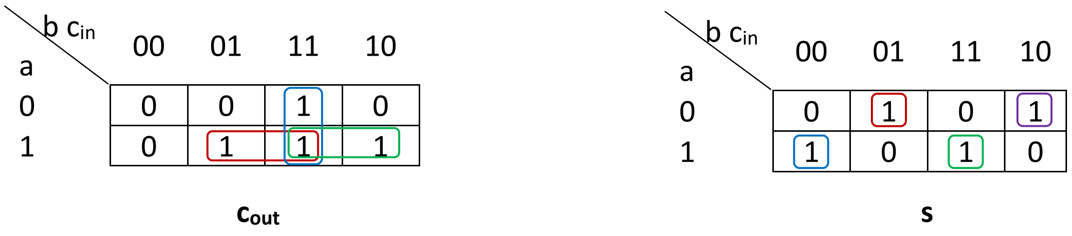
The 1-bit full adder is the basic element when implementing summation on inputs with *n* bits. For an *n*-bit adder, the 1-bit full adder is cascaded *n* times. The 1-bit full adder unit implements the summations of 2 input bits plus a carry input bit c_{in} (Carry In), that can be generated from inferior ranks. On the output, the 1-bit full adder unit generates the result bit s (Sum) and the carry bit c_{out} (Carry Out) that remains to be forwarded at superior ranks. The block diagram and the truth tables are highlighted in Figure 5. 4.



a	b	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Figure 5. 4 Block diagram and the truth table for a 1-bit full adder

From the truth table above the Karnaugh maps of the outputs can be generated:



a \ b c_{in}	00	01	11	10
0	0	0	1	0
1	0	1	1	1

c_{out}

a \ b c_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

s

Figure 5. 5 Karnaugh maps for the 1-bit adder outputs

The resulting Minimal Disjunctive Normal Forms for c_{out} and s are:

$$c_{out} = a \cdot c_{in} + b \cdot c_{in} + a \cdot b$$
$$s = a \cdot \bar{b} \cdot \overline{c_{in}} + \bar{a} \cdot \bar{b} \cdot c_{in} + a \cdot b \cdot c_{in} + \bar{a} \cdot b \cdot \overline{c_{in}} = \ldots = a \oplus b \oplus c_{in} \qquad (5.\ 4)$$

5.2.2.4 The 4-bit (half-byte) adder designed by cascading 1-bit full adders

A 4-bit adder, for inputs $A_{3:0}$ and $B_{3:0}$, can be implemented with four 1-bit full adders connected to allow the carry bit propagation from lower ranks to higher ranks (cascading). The carry input at the lowest rank will be connected to 0 (GND) – the result will be a 4-bit half-adder. The highest carry output bit will indicate an overflow. The logic diagram is presented next:

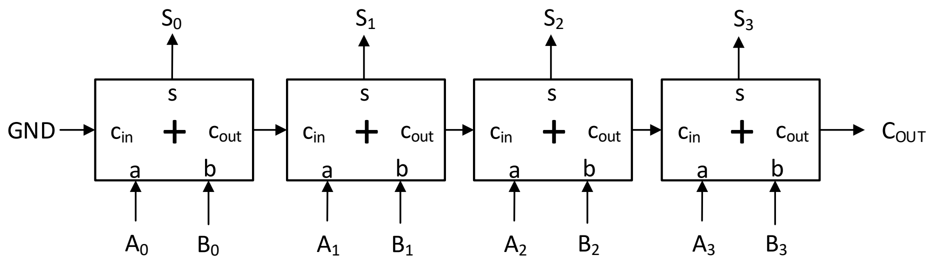


Figure 5. 6 Design of a 4-bit half-adder by cascading 1-bit full adders

Note: Similarly, 4-bit adders can be cascaded to implement summation units on multiple of 4 number of inputs bits: the carry input at the lowest rank will be connected to 0, while the rest of the carry lines will be interconnected to forward the carry bits to superior ranks.

5.2.2.5 Designing adders using minimization techniques

An alternative solution is to express the outputs of the adder as functions of input bits. From the truth tables, it is possible to generate the Karnaugh maps and extract the Minimal Disjunctive Normal Forms, which can be synthesized using logic gates. For instance, a 2-bit adder implementing the sum $A_1A_0+B_1B_0$, will output the sum on S_1S_0, plus a carry out C_{OUT} bit. Its functionality is described in Table 5. 2. Functions S_1, S_0, and C_{OUT} can be minimized using the Karnaugh maps and the boolean algebra laws, leading to the following expressions:

Table 5. 2 A 2-bit half-adder

A_1	A_0	B_1	B_0	C_{OUT}	S_1	S_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

$$\begin{aligned} C_{OUT} &= A_1 \cdot B_1 + A_0 \cdot B_1 \cdot B_0 + A_1 \cdot A_0 \cdot B_0 \\ S_1 &= A_1 \oplus B_1 \oplus (A_0 \cdot B_0) \\ S_0 &= A_0 \oplus B_0 \end{aligned} \tag{5. 5}$$

Note: The pros with minimization techniques (vs. cascading) is the reduced number of logic gates. The cons is the exponential number of combinations involved in the minimization process, as the number of variables increases. Since the minimization based on Karnaugh maps is limited to 4 inputs, alternative solutions should be adopted, such as the Quine-McCluskey algorithm.

5.3 Assignments

1. Calculate the Minimal Disjunctive Normal Form of $f = \sum(0,4,7,10,12,13) + \sum_{\Phi}(1,8,15)$.
2. Implement and test on the board the output W of the BCD–Excess 3 converter.
3. Implement and test on the board the output F_2 of the 2-bit unsigned comparator, using XNOR gates.
4. Implement and test on the board the 1-bit full adder. For the s (Sum) output use XOR gates.
5. Implement and test in Logisim a 4-bit half-adder using 1-bit adders (with attribute `Data Bits = 1`) from the *Arithmetic* library.
6. Implement and test in Logisim an 8421(BCD)–2421(Aiken, https://en.wikipedia.org/wiki/Aiken_code) converter. For implementation, design the truth table and generate the Minimal Disjunctive Normal Forms.