6 Combinational circuits

6.1 Introductory concepts

- A combinational circuit (block, component) consists of logic gates and processes \( n \) input signals \( x(n-1), \ldots, x(0) \) into \( m \) output signals \( y(m-1), \ldots, y(0) \) using a function \( y = f(x) \), in such a way that output signals depend only on the current input signals.

\[
y_i = f_i(x_{n-1}, \ldots, x_0) \quad \text{for} \quad i = 0, \ldots, n-1 \quad \text{or} \quad y = f(x)
\]

where each \( f_i \) is a logic function.

- Past values of the input signals do not have any influence on the current values of the output signals.

- Description of any combinational circuit with \( n \) inputs and \( m \) outputs can be ultimately reduced to a truth table with \( 2^n \) rows and \( m \) column.

- From the point of view of their internal structures combinational blocks can be classified into two groups:
  - un-structured circuits, that is a “random” collection of gates,
  - structured circuits forming 1-D and 2-D arrays of components.

The simplest combinational blocks are collections of gates. For example, an n-bit NAND gate:

- A slightly more complicated example includes the collection of gates driven by a common control signal, say \( s \), to perform two operations:

- We are already familiar with a \( n \)-to-\( 2^n \) decoder as a minterm/maxterm generator. Typically the decoder has an additional enable signal \( s \) such that the minterms are generated only for \( s = 1 \), whereas for \( s = 0 \) outputs are in an inactive state, typically 0.

- The decoder can be also used as a Demultiplexer. The 1-bit demultiplexer receives 1-bit data on a single input and re-directs it into one-out-of-\( m = 2^n - 1 \) outputs selected by an n-bit number \( x \).
6.2 Example of a VHDL code for a 2-to-4 decoder

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY dec2to4 IS
  PORT(
    x : IN std_logic_vector (1 DOWNTO 0);
    y : OUT std_logic_vector (3 DOWNTO 0)
  );
END dec2to4 ;

ARCHITECTURE struct OF dec2to4 IS
  SIGNAL xb : std_logic_vector(1 DOWNTO 0);
BEGIN
  y(0) <= xb(1) AND xb(0);
  y(1) <= xb(1) AND x(0);
  y(2) <= xb(0) AND x(1);
  y(3) <= x(1) AND x(0);
  xb <= NOT(x);
END struct;

• A VHDL program represents a digital circuit.
• It can represent: signal flow, behaviour or structure of the circuit.
• The program can be used to simulation/test, or to synthesize the digital circuit.
• The above program describes signal flow.
• All assignment statements are interpreted/executed concurrently, therefore can be written in any order.

6.3 Multiplexers

• A n-to-1 multiplexer connects its output y to one of n = 2^m inputs a_0, a_1, ..., a_n selected by an m-bit control/select signal s.
• In other words the multiplexer output is a sum of products of input signals with respective minterms:

\[ y = a_0 \cdot m_0 + a_1 \cdot m_1 + \ldots + a_{n-1} \cdot m_{n-1} = \sum_{i=0}^{n-1} a_i \cdot m_i \]

• A 4-to-1 multiplexer can be implemented in the following way:

Function Table:
6.4 Describing a multiplexer in VHDL

There are a number of ways to describe a multiplexer in VHDL. The following two methods use various concurrent assignment statements. We use a 4-to-1 multiplexer as an example.

\[
\begin{array}{cccc}
0 & a(0) \\
1 & a(1) \\
2 & a(2) \\
3 & a(3) \\
\end{array}
\]

\[
\begin{array}{cccc}
s(1:0) & \text{MUX} \\
0 & 1 \\
1 & 0 \\
\end{array}
\]

\[
y = f(x_2, x_1, x_0) = \sum (1, 3, 5, 6)
\]

6.4.1 Conditional Signal Assignment Statement

\[
<= \ldots \text{when} \ldots \text{else}
\]

ARCHITECTURE condSA OF mux4to1A IS
BEGIN
y <= a(0) WHEN s = "00" ELSE
a(1) WHEN s = "01" ELSE
a(2) WHEN s = "10" ELSE
a(3) ;
END condSA ;

6.4.2 Selected Signal Assignment Statement

\[
<= \text{with} \ldots \text{select} \ldots \text{when} \ldots , \ldots ;
\]

ARCHITECTURE selSA OF mux4to1A IS
BEGIN
WITH s SELECT
y <= a(0) WHEN "00" , -- comma
a(1) WHEN "01" ,
a(2) WHEN "10" ,
a(3) WHEN OTHERS ; -- semicolon
END selSA ;

Multiplexer as a universal logic element

- Typically multiplexers are used to re-direct signals from different sources onto a common output.
- However, when we compare expression for canonical implementation of a logic function with expression for a multiplexer we note that they are structurally identical.
- It means that a multiplexer with \( m \) select signals can be used as a universal logic block implementing any logic function of \( m \) variables specified by constants (from a truth table) at the multiplexer inputs.
- As an example consider implementation of a 3-variable function using an 8-to-1 multiplexer:

\[
y = f(x_2, x_1, x_0) = \sum (1, 3, 5, 6)
\]

- If we allow inputs to the multiplexer to be not only constants \((0, 1)\), but also variable(s) (or their complements), then, in particular, using a \( 2^m \)-to-1 multiplexer, we can implement any logic function of \( m + 1 \) variables.
- In such a case \( m \) variables are applied to the select inputs of the multiplexer, whereas the remaining variable, its complement and constants \((0, 1)\) are applied to the multiplexed inputs.
Example

Implement a 3-variable function
\[ y = f(x_2, x_1, x_0) = \sum (1, 3, 4, 5) \]
using a \(2^2\)-to-1 multiplexer

- Variables \((x_1, x_0)\) are used as the select variables in a 4-to-1 multiplexer
- The remaining variable \(x_2\) will be used at the multiplexer inputs.

- To do this we modify the truth table comparing values of the output signal \(y\) for two values of the variable \(x_2\)

6.5 Unstructured combinational circuits

- The name “unstructured” refers to implementations of a \(n\)-input \(m\)-output combinational circuit build from simple gates which are not grouped into any sub-blocks.
- To illustrate the concept let us consider the following implementation of a 2-bit multiplier.
- It is a 2-bit by 2-bit multiplication circuit that forms a 4-bit product:

\[ a \cdot b = \underbrace{a(a, a_0) \cdot b(b, b_0)}_{2 \times 1} = \underbrace{(c, c_2, c_1, c_0)}_{2 \times 2} \]

such that \( c = a \cdot b = f(a, b) \)

4 Boolean functions of 4 variables must be derived:

\[ f_3 = f_3(a, a_0, b, b_0) \]
\[ f_2 = f_2(a, a_0, b, b_0) \]
\[ f_1 = f_1(a, a_0, b, b_0) \]
\[ f_0 = f_0(a, a_0, b, b_0) \]
The next step is to convert the truth table into the Karnaugh maps:

**Multiplication table:**

From the Karnaugh maps we can obtain the following SoP expressions:

\[
\begin{align*}
C_3 &= a_1a_0b_1b_0 \\
C_2 &= a_1b_1\overline{b_0} + b_1a_0 = a_1b_1(\overline{a_0} + \overline{b_0}) \\
C_1 &= a_1\overline{b_1}b_0 + a_0\overline{b_1}b_0 + b_1\overline{a_1}a_0 + b_1\overline{a_0}a_0 \\
&= a_1b_0(\overline{a_1} + \overline{b_0}) + b_1a_0(\overline{a_1} + \overline{b_0}) \\
C_0 &= b_0a_0 
\end{align*}
\]

A possible implementation with “mixed” gates: AND, OR NAND

Note that equation for \(c_1\) has been simplified so that it is no longer a standard form but a 3-level implementation:

\[
\begin{align*}
C_3 &= a_1a_0b_1b_0 \\
C_2 &= a_1b_1\overline{b_0} + b_1a_0 = a_1b_1(\overline{a_0} + \overline{b_0}) \\
C_1 &= a_1\overline{b_1}b_0 + a_0\overline{b_1}b_0 + b_1\overline{a_1}a_0 + b_1\overline{a_0}a_0 \\
&= a_1b_0(\overline{a_1} + \overline{b_0}) + b_1a_0(\overline{a_1} + \overline{b_0}) \\
C_0 &= b_0a_0 
\end{align*}
\]

The above implementation is an example of an unstructured combinational circuit.