50.002 COMPUTATIONAL STRUCTURES

INFORMATION SYSTEMS TECHNOLOGY AND DESIGN

Logic Synthesis

Natalie Agus (Fall 2018)

1 Sum of Products

A combinational device has functional specifications. Functional specifications are represented with **truth tables**. So for example, the following is the truth table of an NAND gate (input: A and B, output: Y):

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Table 1

We can also have functional specifications in terms of boolean expression. To convert truth tables into boolean expressions, we take the following steps:

1. Look **only for the rows with Y= 1**. That means we only look at rows 1 to 3, and ignore the fourth row for the boolean expression.
2. For each row, If the value of the input is a 0, then express it with a NOT.
3. So for row 1, we have $\overline{A}\ \overline{B}$. For row 2, we have $\overline{A}B$. For row 3 we have $A\overline{B}$.
4. Sum all the expressions from the rows with Y=1:

$$Y = \overline{A}\ \overline{B} + \overline{A}B + A\overline{B} \tag{1}$$

5. The expression above is called the ***sum of products***.

Sometimes in textbooks, it is called as canonical sum of products. They mean the same thing as just "sum of products".

2 Straightforward Logic Synthesis

Given a sum-of-products boolean expression, we can make a combinational device that has that boolean expression as functional specification using three level of logics: INV, AND, OR with arbitrary number of inputs. For example, given the following sum of products expression,

$$Y = \overline{C}\ \overline{B}A + \overline{C}BA + CB\overline{A} + CBA$$

We can make a combinational device as such that it adheres to the expression above:

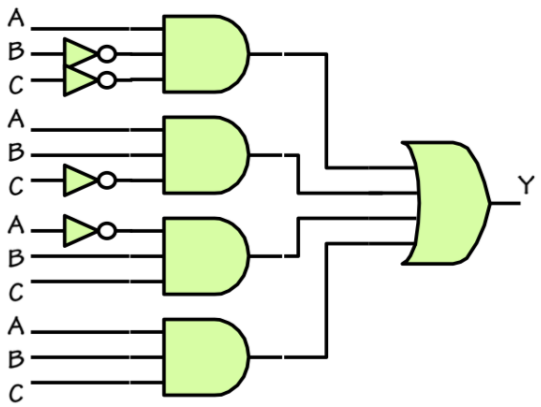


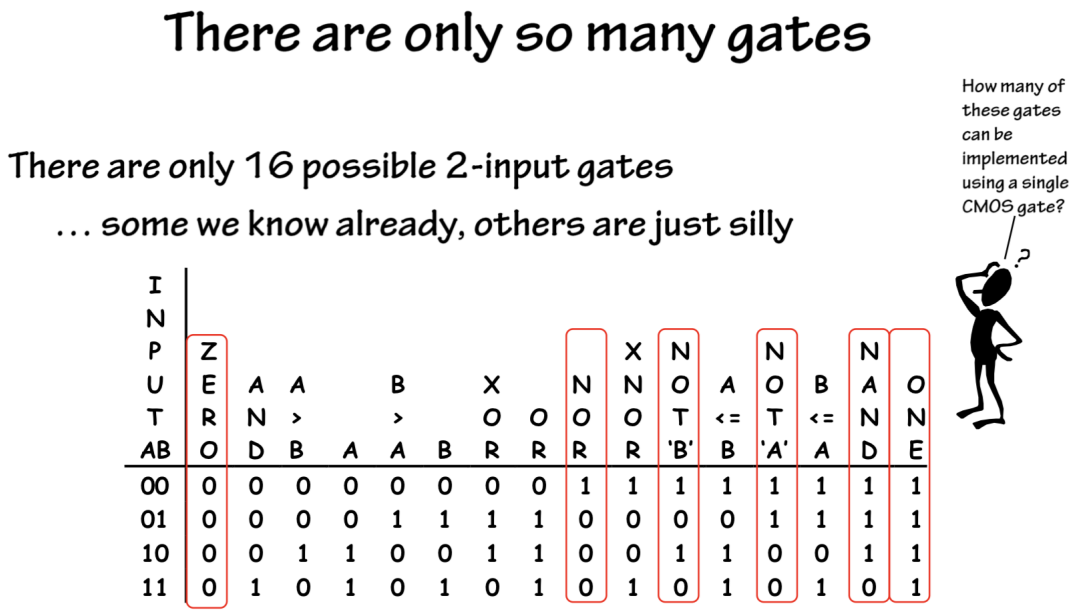
Figure 1

Explanation:

1. The boolean expression of Y contains 4 terms summed together
2. The OR gate at the output Y represents the summation of the four terms.
3. The AND gates in the second ”column” of the figure represents the combination of each of the input terms
4. The INV at the input represents the NOT inputs (0 inputs).

3 N-input Gates

There are 16-possible 2-input gates. See the image below. In short, **there are 2^{2^x} possible x-input gates**.



INPUT AB	ZERO	AND	A > B	A	B > A	B	XOR	OR	NOR	XNOR	NOT 'B'	A <= B	NOT 'A'	B <= A	NAND	ONE
00	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
01	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
10	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
11	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Figure 2

4 DeMorgan's Theorem

$$\overline{A+B} = \overline{A}\,\overline{B} \tag{2}$$

$$\overline{AB} = \overline{A}+\overline{B} \tag{3}$$

5 NAND and NOR gates

NANDs and NORs are universal, meaning that they can implement **any** boolean function. AND, OR, and INV aren't sufficient. We can use NANDs and NORs to make AND, OR and INV:

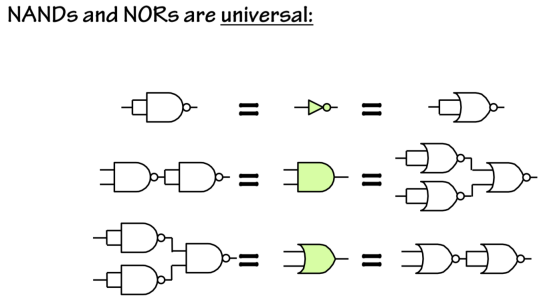


Figure 3

6 Boolean Algebra

The boolean algebra is useful to manipulate boolean equations.

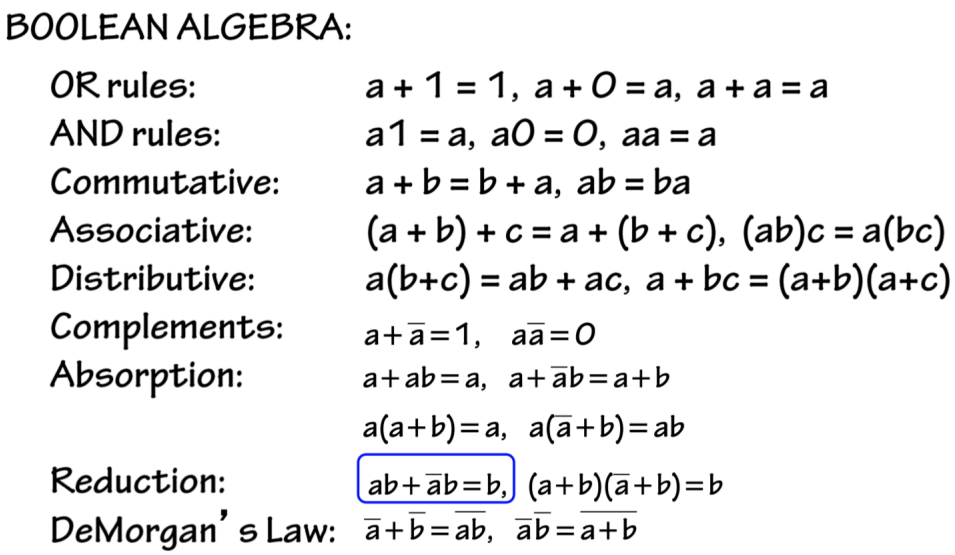


Figure 4

7 Boolean Minimization

We can use the *reduction* rule to perform boolean minimization:

$$Y = \overline{C}\,\overline{B}A + \overline{C}BA + CB\overline{A} + CBA \tag{4}$$

$$= \overline{C}\,\overline{B}A + \overline{C}BA + CB \text{ (reduce the last two terms)} \tag{5}$$

$$= \overline{C}A + CB \text{ (reduce the first two terms)} \tag{6}$$

8 Karnaugh Map

Note: Please read this method on your own. It is a straightforward *alternative* method to perform boolean minimization. This is a method to easily perform boolean minimization, and **ultimately the end goal is to reduce the digital circuit to its minimum number of gates** (save cost, save time).

The following figure in the next page shows a 2-input (by input it just basically means how many input boolean variables), 3-input, and 4-input Karnaugh maps.

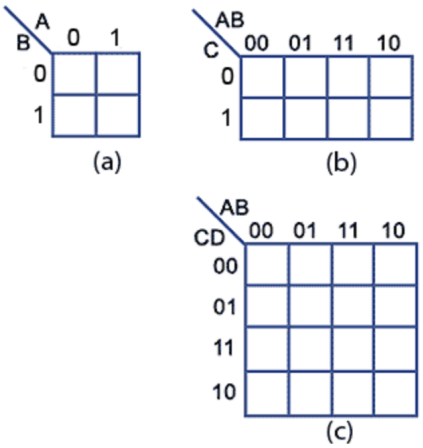


Fig.2.4.1 Karnaugh Maps

Figure 5

The number of cells of Karnaugh maps with x inputs is 2^x cells. Then, fill in '1' to all the cells that represent logic '1' on the boolean expression. This is illustrated in the truth table and its corresponding Karnaugh map below:

Table 2.4.1

A	M	C	X	Boolean
0	0	0	0	
0	0	1	0	
0	1	0	1	M
0	1	1	1	M•C
1	0	0	0	
1	0	1	1	A•C
1	1	0	1	A•M
1	1	1	1	A•M•C

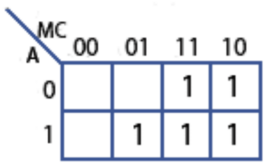


Figure 6

Then you can simplify the Karnaugh Map using these 6 ground rules:

1. Groups should contain as many "1" cells (i.e. cells containing a logic 1) as possible and no blank cells.
2. Groups can only contain 1, 2, 4, 8, 16 or 32... etc. cells (powers of 2).
3. A "1" cell can only be grouped with adjacent "1" cells that are immediately above, below, left or right of that cell; no diagonal grouping.
4. Groups of "1" cells can overlap. This helps make smaller groups as large as possible, which is an advantage in finding the simplest solution.

5. The top/bottom and left/right edges of the map are considered to be continuous, as shown in Fig. 2.4.3, so larger groups can be made by grouping cells across the top and bottom or left and right edges of the map.

6. There should be as few groups as possible.

Following the rules above, the simplified example Karnaugh map is:

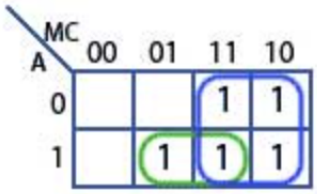


Figure 7

To convert this Map back into boolean expression, we need to look at each group:

1. In the blue group, the output is 1 regardless of A, and regardless of C. Hence, the boolean expression for the blue group is just M.

2. In the green group, the output is 1 regardless of M. Therefore, the boolean expression for the green group is AC.

3. The complete simplified boolean expression is: $X = M + AC$.

9 Multiplexer

The Multiplexer (Mux) is implemented using logic gates. It is expensive, but universal, meaning that it can implement any boolean function because essentially it ”hardcodes” the truth table. A mux is made up of logic gates (INV, AND, and OR, or NANDs). The symbol for the mux is as shown in the image below. The truth table is written at the side. A mux always has three components: the inputs, the selector signal(s), and the output. It basically ”allow” either of the input signal to pass through when selected. For example in the case of 2-input mux below, when S=0, it will pass through signal from D_0 as output.

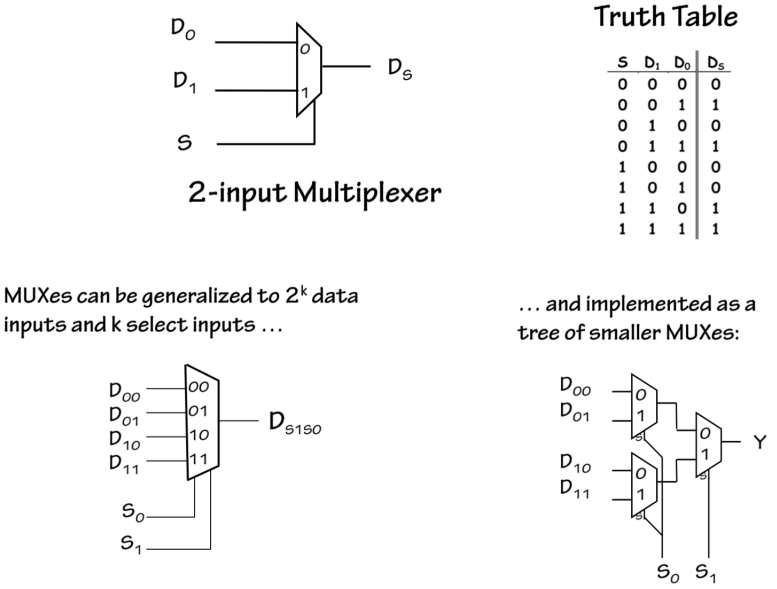


Figure 8

Some properties about multiplexers:

1. Muxes are universal, meaning that it can implement any boolean functions
2. A Mux can have 2^k data inputs, and k bits select inputs, and **only can have 1 output**

Below is an example of how Mux can be used to implement a more complex combinational device, the full adder:

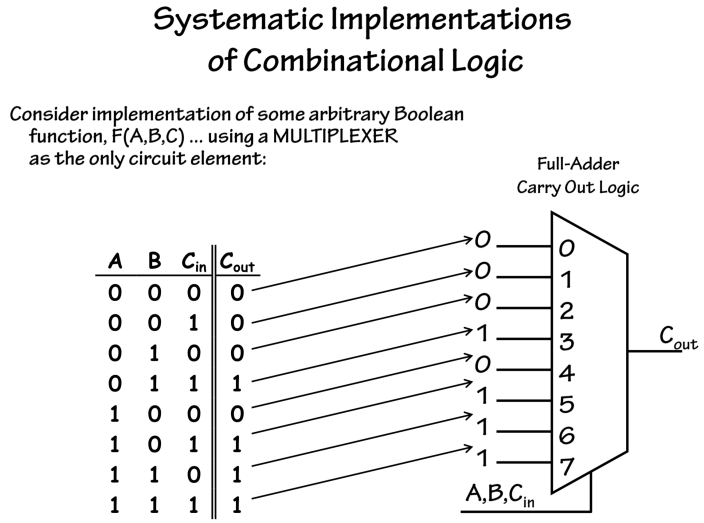


Figure 9

10 Decoder

The schematic below represents the schematic and truth table of a decoder with 2 select inputs A_0 and A_1:

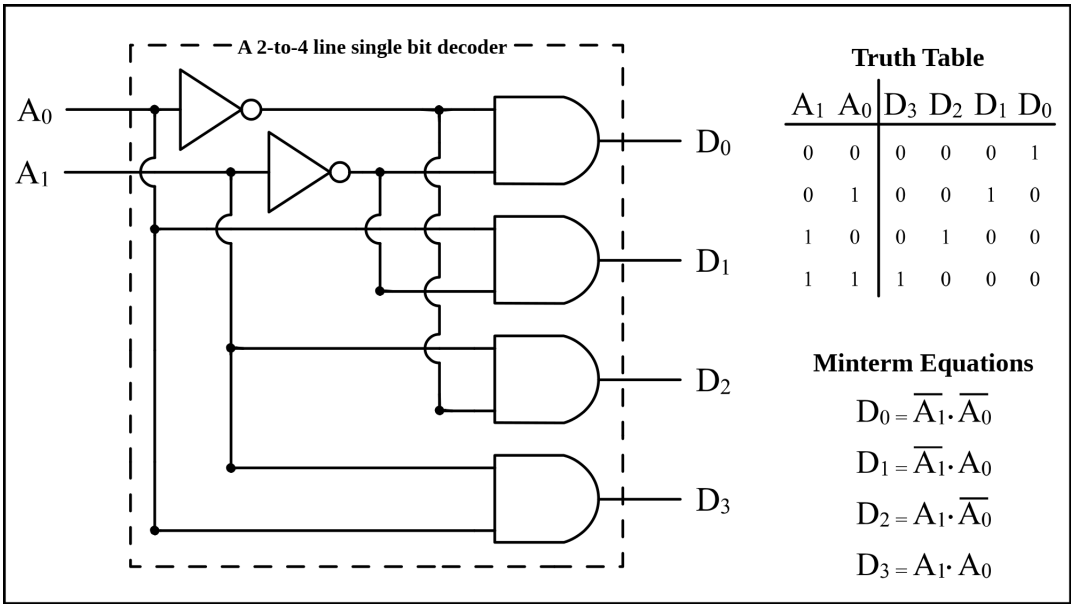


Figure 10

Note: do not worry about the logic gate schematics of a decoder. It is only there to show you that a decoder is made up of the normal logic gates like inverters and AND gates. In practice, use the symbol of decoder as shown in Figure 11.

Some properties about decoders:

1. A Decoder is the opposite of Mux
2. It has k select inputs, and 2^k **possible data outputs**
3. The selected output i is HIGH (1), and the rest of the $2^k - 1$ data output is LOW (0).

11 Read-Only-Memories (ROM)

A decoder's function is to create a read-only-memories. For example, if we sort of "hard-code" the Full-Adder, we end up with the following schematic in the next page. Explanation for the schematic:

1. At the output of the decoder, the little circuit with inverted triangle symbol signifies a pulldown circuit (connected to ground), which will "drain" a signal into 0.

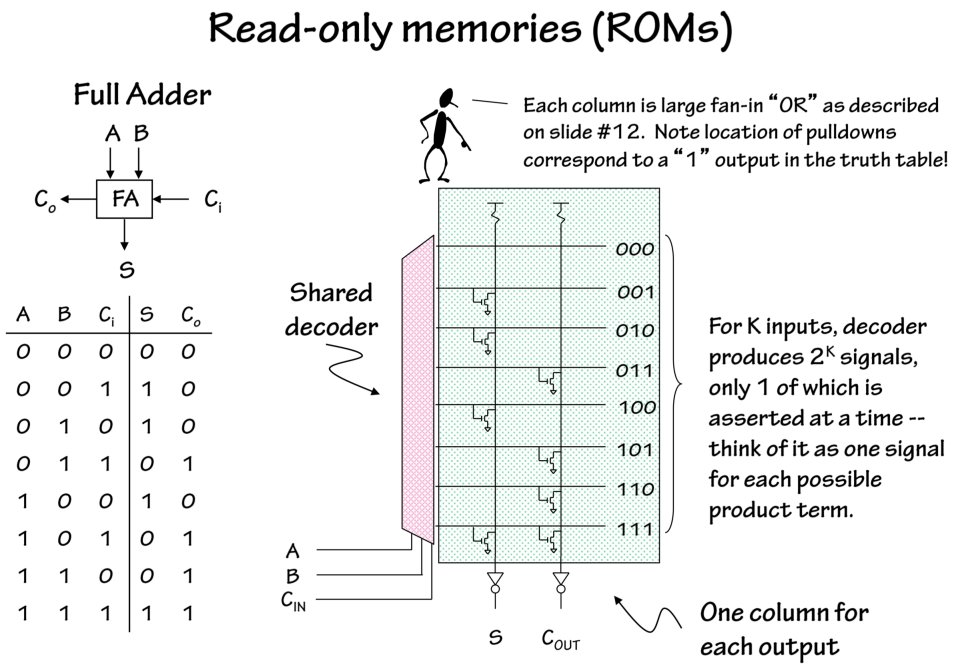


Figure 11

2. At each combination of A, B, and C_i, only one of the 8 outputs of the decoder will be 1. For example, when $A = 0, B = 0, C_i = 1$, the second output from the top is 1. There's a pulldown for S (it is connected to the ground), which makes it 0, and no pulldown for the C_{out}, which makes it 1.

3. Note the **presence of inverters by invention** at the end of the two vertical output lines for S and C_{out}, so the output is inverted to be 1 for S and 0 for C_{out}.

4. By **invention**, the location of the "pulldown" circuits correspond to a 1 in the truth table for that particular output (S or C_{out}).

Properties of ROM:

1. ROMs ignore the structure of combinational functions (hardcoded)

2. The selectors are like "addresses" of an entry

3. For an N-input boolean function, the size of ROM is roughly $2^N \times$ #outputs. For example, the Full Adder has 3 inputs (A, B, C_{in}), and 2 outputs (S and C_{out}). Hence the size of the ROM is $2^3 * 2 = 16$.