DIGITAL DESIGN

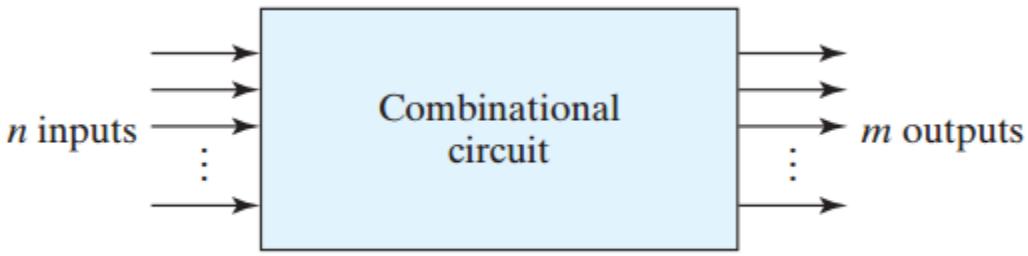
Combinational Logic Blocks

In digital circuit theory, **combinational logic**(sometimes also referred to as **time-independent logic**) is a type of digital logic which is implemented by Boolean circuits, where the output is a pure function of the present input only. This is in contrast to sequential logic, in which the output depends not only on the present input but also on the history of the input. In other words, sequential logic has *memory* while combinational logic does not.

Combinational logic is used in computer circuits to perform Boolean algebra on input signals and on stored data. Practical computer circuits normally contain a mixture of combinational and sequential logic. For example, the part of an arithmetic logic unit, or ALU, that does mathematical calculations is constructed using combinational logic. Other circuits used in computers, such as half adders, full adders, half subtractors, full subtractors,multiplexers, demultiplexers, encoders and decoders are also made by using combinational logic.

An alternate term is **combinatorial logic**, though this usage may be considered controversial.

A combinational circuit consists of an interconnection of logic gates. Combinational logic gates react to the values of the signals at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data. A block diagram of a combinational circuit is shown



Block Diagram of a Combinational Circuit

The n input binary variables come from an external source; the m output variables are produced by the internal combinational logic circuit and go to an external destination. Each input and output variable exists physically as an analog signal whose values are interpreted to be a binary signal that represents logic 1 and logic 0. (Note: Logic simulators show only 0's and 1's, not the actual analog signals.) In many applications, the source and destination are storage registers. If the registers are included with the combinational gates, then the total circuit must be considered to be a sequential circuit.

For n input variables, there are 2^n possible combinations of the binary inputs. For each possible input combination, there is one possible value for each output variable. Thus, a combinational circuit can be specified with a truth table that lists the output values for each combination of input variables. A combinational circuit also can be described by m Boolean functions, one for each output variable. Each output function is expressed in terms of the n input variables.

There are several combinational circuits that are employed extensively in the design of digital systems. These circuits are available in integrated circuits and are classified as standard components. They perform specific digital functions commonly needed in the design of digital systems. In this chapter, we introduce the most important standard combinational circuits, such as adders, subtractors, comparators, decoders, encoders, and multiplexers. These components are available in integrated circuits as medium-scale integration (MSI) circuits. They are also used as standard cells in complex very largescale integrated (VLSI) circuits such as application-specific integrated circuits (ASICs). The standard cell functions are interconnected within the VLSI circuit in the same way that they are used in multiple-IC MSI design.

Adder Circuit

We would start with a truth-table, then write the canonical Boolean equation, then simplify. Unfortunately, that technique is only effective for very narrow adders, because the size of the truth-table is too large for wider adders. Therefore, we need to find a way to break the design up into smaller more manageable pieces. We will design the smaller pieces individually, then wire them together to create the entire wide adder.

Long-hand addition gives us a good guide on how to break up the adder into pieces. When we add by hand, we begin by adding together the two least significant bits of A and B, a0 and b0, respectively. From that addition, we generate a result bit, s0, and possibly a carry bit. Then we move over to the next more significant column, adding the carry from the previous stage along with the next two bits of A and B, a1 and b1, respectively. We then continue the process by moving left one column at a time until we have finished all n columns.

The truth-table representing the value of least significant sum bit, s0, and the carry out of the least significant stage, c1, is shown below. (We name a carry bit according to the stage to which it is an input. Therefore, the carry into stage 1 is called c1.) By inspection of the true-table we can simply write the logic equations for this stage.

$$\begin{array}{cccc|c} & a_3 & a_2 & a_1 & a_0 \\ + & b_3 & b_2 & b_1 & b_0 \\ \hline & s_3 & s_2 & s_1 & s_0 \end{array}$$

a_0	b_0	s_0	c_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$s_0 = a_0 \text{ XOR } b_0$$
$$c_1 = a_0 \text{ AND } b_0$$

Here are the details for the next significant stage and all subsequent stages. The truth-table now has 8 rows, because there are three

inputs into these stages, a bit from each of A and B, along with the carryout from the previous stage. Even though the true-table is larger, the logic equations are still simple to write. By carefully inspecting the truth-table you will notice that the sum output is the exclusive-or of the three inputs—it is a 1 when the number of 1's in the input is odd. The carry-out function is the majority function—it is a 1 when the number of 1's in its input is greater than the number of 0's.

$$\begin{array}{cccc|c} & a_3 & a_2 & a_1 & a_0 \\ + & b_3 & b_2 & b_1 & b_0 \\ \hline & s_3 & s_2 & s_1 & s_0 \end{array}$$

a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$s_0 = a_0$ XOR b_0

$c_1 = a_0$ AND b_0

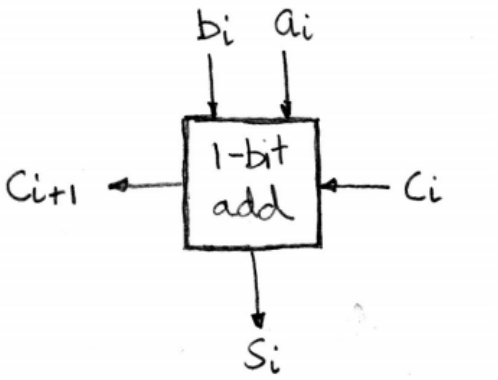
Here are the details for the next significant stage and all subsequent stages. The truth-table now has 8 rows, because there are three inputs into these stages, a bit from each of A and B, along with the carryout from the previous stage. Even though the true-table is larger, the logic equations are still simple to write. By carefully inspecting the truth-table you will notice that the sum output is the exclusive-or of the three inputs—it is a 1 when the number of 1's in the input is odd. The carry-out function is the majority function—it is a 1 when the number of 1's in its input is greater than the number of 0's.

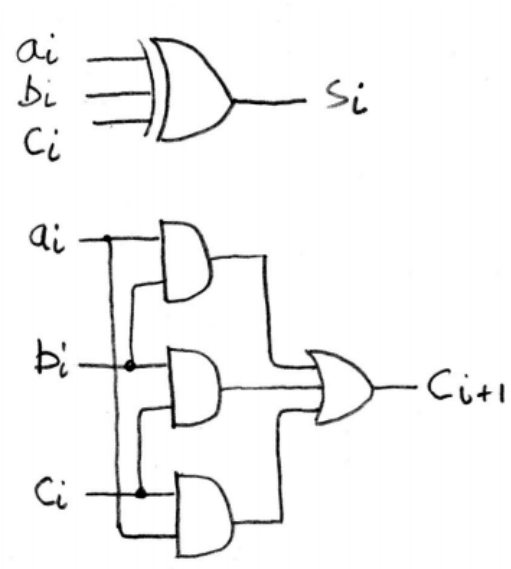
$$
\begin{array}{cccc|c}
 & a_3 & a_2 & a_1 & a_0 \\
+ & b_3 & b_2 & b_1 & b_0 \\
\hline
 & s_3 & s_2 & s_1 & s_0
\end{array}
$$

a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$
\begin{aligned}
s_i &= \text{XOR}(a_i, b_i, c_i) \\
c_{i+1} &= \text{MAJ}(a_i, b_i, c_i) = a_i b_i + a_i c_i + b_i c_i
\end{aligned}
$$

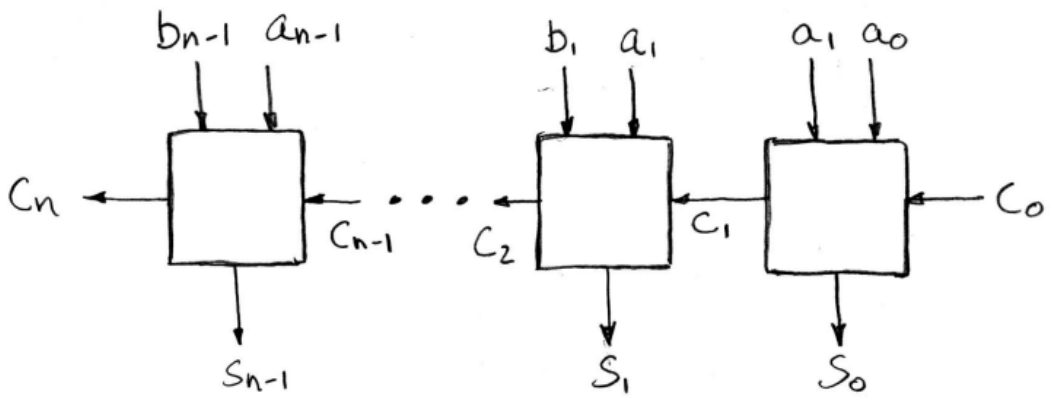
We will encapsulate the operations of one stage, or column, of the add operation into a small block. You can think of this block as a 1-bit adder. Its official name is a full-adder cell, but most people confuse this with an n-bit adder, which it certainly is not. The symbol we will use for this 1-bit adder, along with the gate-level circuit diagrams for its internals are shown below:





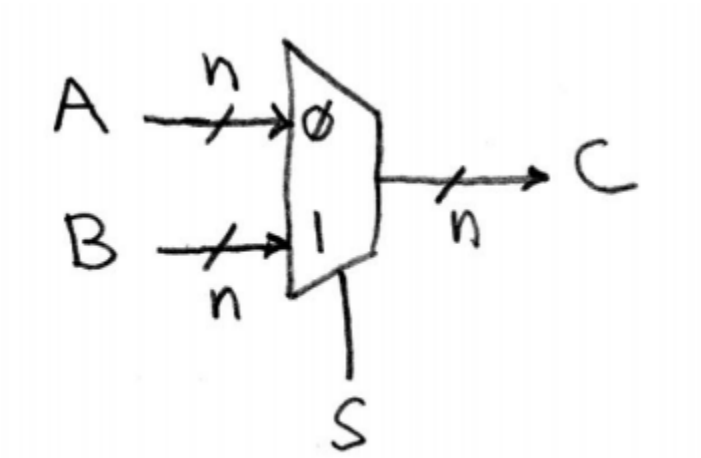
We will not bother creating a special block for the first column of the adder, because anything it can do the 1-bit adder block can do as well. We just need to figure out where to attach the carry-in to the first column.

The next step in the design of our adder circuit is to wire together a collection of our 1-bit adders to create an n-bit adder. All we need to do is to wire the carry-out output of one stage into the carry-in input of the next, from least significant to most, as shown below:



Data Multiplexors

A data multiplexor, commonly called a mux, is a circuit that selects its output value from a set of input values. It is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2^k input lines and k selection lines whose bit combinations determine which input is selected. For instance, consider the mux circuit shown below:



This mux has two n-bit inputs, A and B, and an n-bit output, C. Additionally, it has a special control signal labeled s, for select. The s signal is used to control which of the two input values is directed to the output. Specifically, the function of this mux can be described with these two rules:

when s=0, C=A

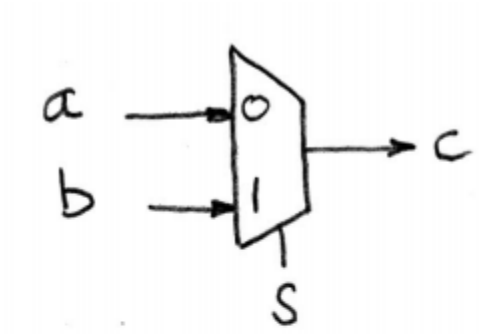
when s=1, C=B

To remind us of which value of s corresponds to which input, within the mux symbol we commonly label each input with its corresponding s value.

This particular mux example is a called a 2-to-1, n-bit wide multiplexor. It is 2-to-1 because it takes two data inputs and outputs one of them. It is n-bit wide because all data signals are n-bits in width. Notice, however, that the s signal is a single bit wide. Whenever a circuit must choose data from multiple sources, a mux is used.

Let's take a look inside the n-bit wide 2-to-1 mux. The simplest way to understand it is to consider it to be a collection of n instances of 1-bit wide 2-to-1 muxes. Each instance of the 1-bit wide mux is responsible for generating one bit of C from one bit of A and one bit of B. All n instances share the same control signal, s.

A 1-bit wide 2-to-1 mux is shown below along with its truth-table

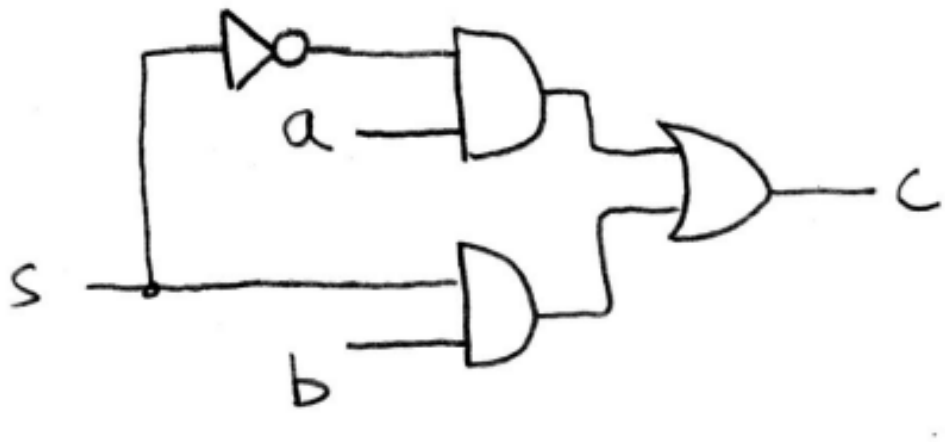


s	ab	c
0	00	0
0	01	0
0	10	1
0	11	1
1	00	0
1	01	1
1	10	0
1	11	1

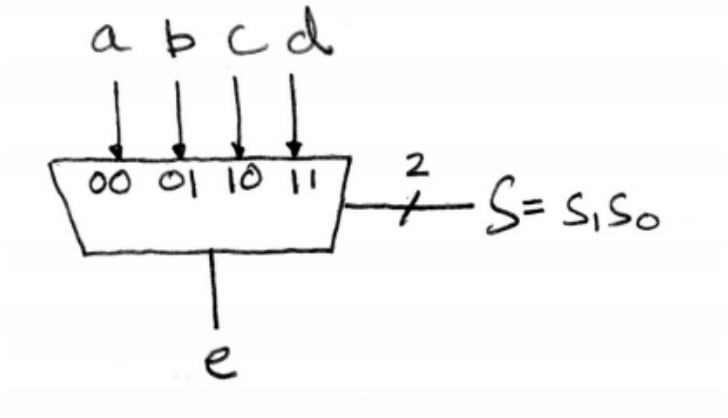
To come up with the logic equation and the associated gate-level circuit diagram we can apply the technique that we studied last lecture. We write the sum-of-products canonical form and simplify through algebraic manipulation. The algebraic steps and final result is

shown below. Intuitively this result makes sense; When the control input, s, is a 0, the right hand side of the equation reduces to a, and when it is a 1, the expression reduces to b.

$$\begin{aligned} c &= \overline{s}a\overline{b} + \overline{s}ab + s\overline{a}b + sab \\ &= \overline{s}(a\overline{b} + ab) + s(\overline{a}b + ab) \\ &= \overline{s}(a(\overline{b} + b)) + s((\overline{a} + a)b) \\ &= \overline{s}(a(1) + s((1)b) \\ &= \overline{s}a + sb \end{aligned}$$



Often times we find the need to extend the number of data inputs of a multiplexor. For instance, consider a 4-to-1 multiplexor:

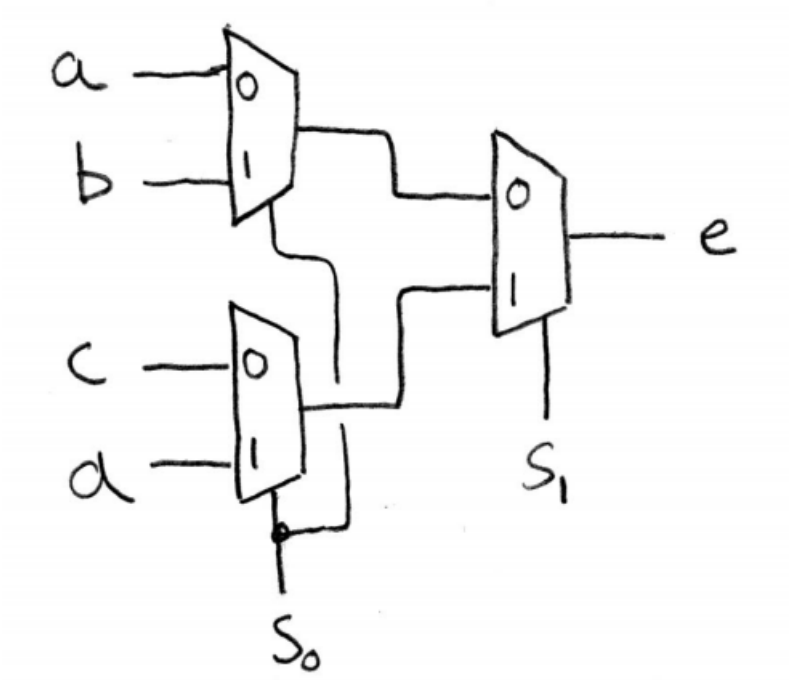


when S=00, e=a
when S=01, e=b
when S=10, e=c
when S=11, e=d

How would we come up with the circuit for this mux? We could start by enumerating the truth table—in this case the function has 4 single bit data inputs and one 2-bit wide control input, for a total of 6 single bit inputs. The truth-table would have 2^6, or 64 rows. Certainly, a feasible approach. If we were to do this, we would end up with the following logic equation:

$$e = \overline{s_1} \cdot \overline{s_0}a + \overline{s_1}s_0b + s_1\overline{s_0}c + s_1s_0d$$

Another way to design the circuit is to base it on the hierarchical nature of multiplexing. We can build a 4-to-1 mux from three 2-to-1 muxes as shown below:

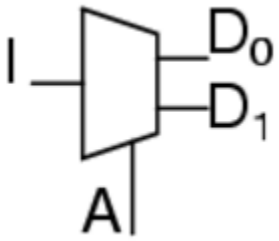


The first layer of muxes uses the s0 input to narrow the four inputs down to two, then the second layer uses s1 to choose the final output.

Demultiplexer

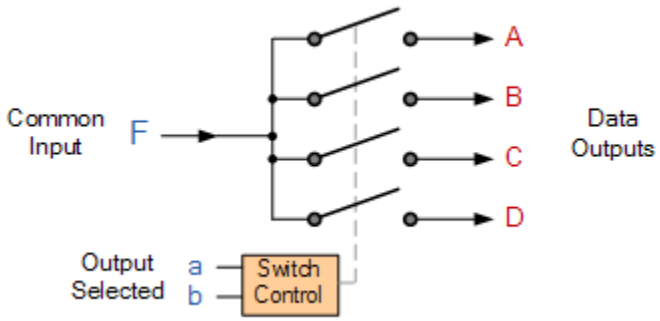
A demultiplexer, sometimes abbreviated dmux, is a circuit that has one input and more than one output. It is used when a circuit wishes to send a signal to one of many devices. This description sounds similar to the description given for a decoder, but a decoder is used to select among many devices while a demultiplexer is used to send a signal among many devices.

A demultiplexer is used often enough that it has its own schematic symbol.



The demultiplexer takes one single input data line and then switches it to any one of a number of individual output lines one at a time. The **demultiplexer** converts a serial data signal at the input to a parallel data at its output lines as shown below.

1-to-4 Channel De-multiplexer



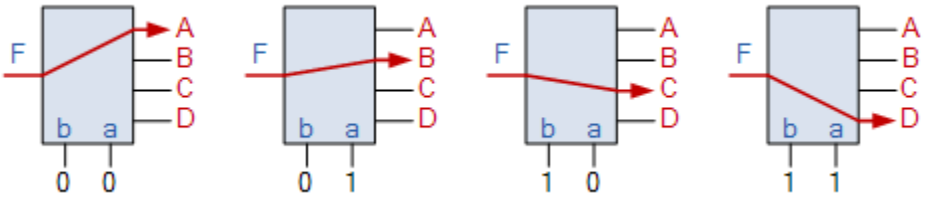
Output Select		Data Output Selected
b	a	
0	0	A
0	1	B
1	0	C
1	1	D

The Boolean expression for this 1-to-4 **Demultiplexer** above with outputs A to D and data select lines a, b is given as:

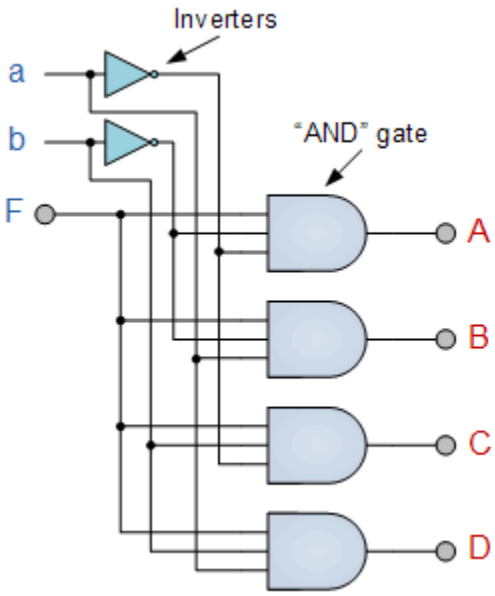
$$F = \overline{ab}A + \bar{a}bB + a\bar{b}C + abD$$

The function of the **Demultiplexer** is to switch one common data input line to any one of the 4 output data lines A to D in our example above. As with the multiplexer the individual solid state switches are selected by the binary input address code on the output select pins “a” and “b” as shown.

Demultiplexer output line selector



As with the previous multiplexer circuit, adding more address line inputs it is possible to switch more outputs giving a 1-to-2^n data line outputs. The implementation of the Boolean expression above using individual logic gates would require the use of six individual gates consisting of AND and NOT gates as shown.


Inverters
a
b
F
"AND" gate
A
B
C
D

Comparator

The comparison of two numbers is an operation that determines whether one number is greater than, less than, or equal to the other number. A magnitude comparator is a combinational circuit that compares two numbers A and B and determines their relative magnitudes. The outcome of the comparison is specified by three binary variables that indicate whether A > B, A = B, or A < B.

The algorithm is a direct application of the procedure a person uses to compare the relative magnitudes of two numbers. Consider two numbers, A and B , with four digits each. Write the coefficients of the numbers in descending order of significance:

$$A = A_3\ A_2\ A_1\ A_0$$

$$B = B_3\ B_2\ B_1\ B_0$$

Each subscripted letter represents one of the digits in the number. The two numbers are equal if all pairs of significant digits are equal: $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$, and $A_0 = B_0$. When the numbers are binary, the digits are either 1 or 0, and the equality of each pair of bits can be expressed logically with an exclusive-NOR function as

$$x_i = A_i\ B_i + \overline{A_i}\ \overline{B_i} \quad \text{for } i = 0, 1, 2, 3$$

where $x_i = 1$ only if the pair of bits in position i are equal (i.e., if both are 1 or both are 0).

The equality of the two numbers A and B is displayed in a combinational circuit by an output binary variable that we designate by the symbol (A = B). This binary variable is equal to 1 if the input numbers, A and B, are equal, and is equal to 0 otherwise. For equality to exist, all x_i variables must be equal to 1, a condition that dictates an AND operation of all variables:

$$(A = B) = x_3 x_2 x_1 x_0$$

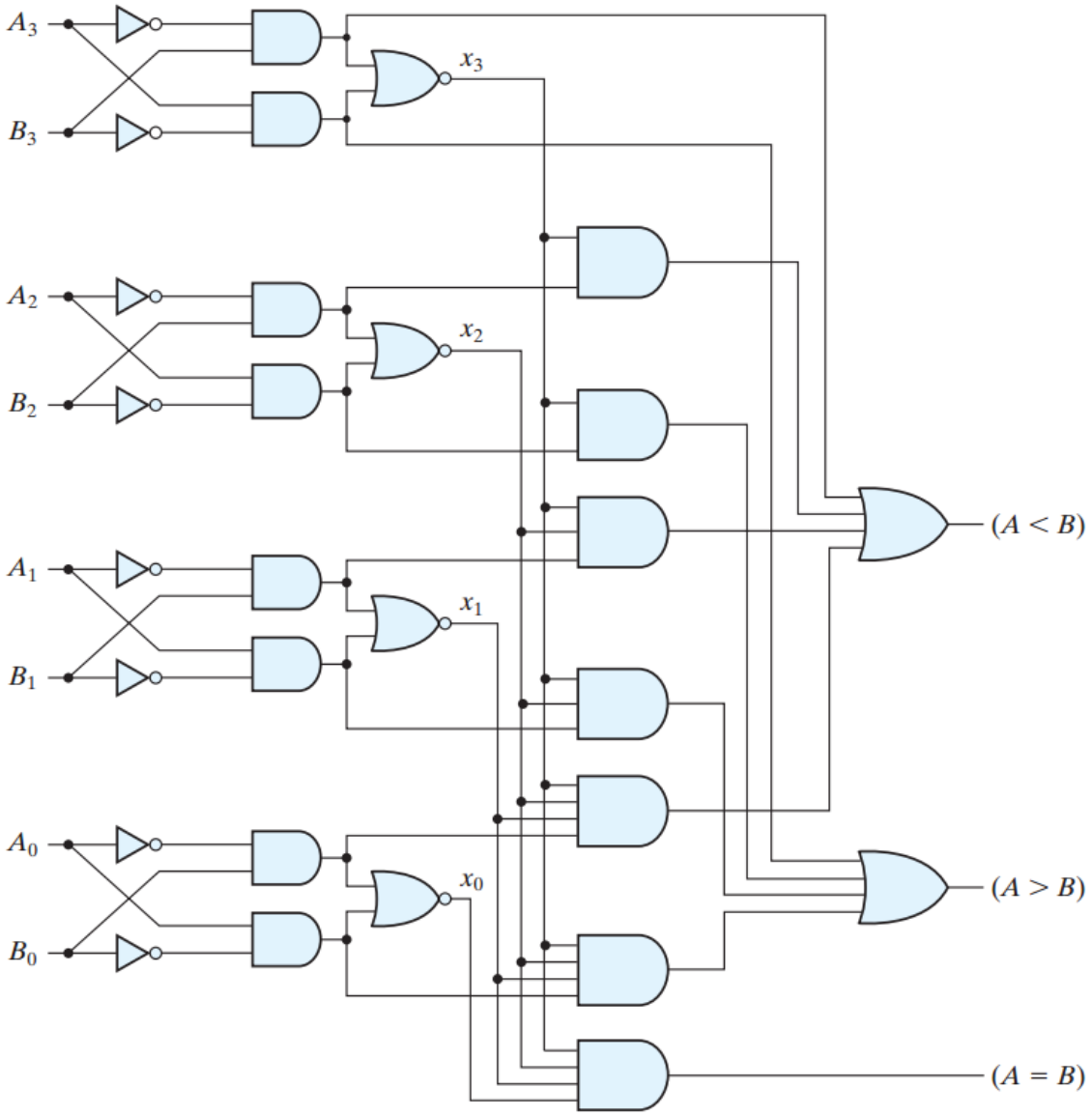
The binary variable (A = B) is equal to 1 only if all pairs of digits of the two numbers are equal.

To determine whether A is greater or less than B, we inspect the relative magnitudes of pairs of significant digits, starting from the most significant position. If the two digits of a pair are equal, we compare the next lower significant pair of digits. The comparison continues until a pair of unequal digits is reached. If the corresponding digit of A is 1 and that of B is 0, we conclude that A > B. If the corresponding digit of A is 0 and that of B is 1, we have A < B. The sequential comparison can be expressed logically by the two Boolean functions

$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$

$$(A < B) = A_3'B_3 + x3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$$

The symbols (A > B) and (A < B) are binary output variables that are equal to 1 when A > B and A < B, respectively. The logic diagram of the four-bit magnitude comparator is shown below.


A_3
B_3
x_3
A_2
B_2
x_2
A_1
B_1
x_1
A_0
B_0
x_0
$(A < B)$
$(A > B)$
$(A = B)$

Seven Segment Display

An ABCD-to-seven-segment decoder is a combinational circuit that converts a decimal digit in BCD to an appropriate code for the selection of segments in an indicator used to display the decimal digit in a familiar form. The seven outputs of the decoder (a, b, c, d, e, f, g) select the corresponding segments in the display. However, to display the characters and numbers (in order to produce the decimal readout), seven-segment displays are most commonly used. Mostly these displays are driven by the output stages of digital ICs (to which the visual indication of the output stages has to be performed) such as latches and decade counters, etc.

Principle of Display Decoder Circuit

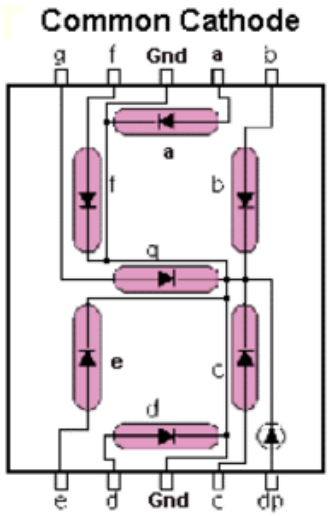
The basic idea involves driving a common cathode 7-segment LED display using combinational logic circuit. The logic circuit is designed with 4 inputs and 7 outputs, each representing an input to the display IC. Using Karnough's map, logic circuitry for each input to the display is designed.

Theory Behind the Circuit:

The first and foremost aspect of this circuit is decoder. A decoder is a combinational circuit which is used to convert a binary or BCD (Binary Coded Decimal) number to the corresponding decimal number. It can be a simple binary to decimal decoder or a BCD to 7 segment decoder.

Seven Segment Display Decoder Circuit Design

Step 1: The first step of the design involves analysis of the common cathode 7-segment display. A 7-segment display consists of an arrangement of LEDs in an 'H' form. A truth table is constructed with the combination of inputs for each decimal number. For example, decimal number 1 would command a combination of b and c (refer the diagram given below).



7 Segment LED

Step 2: The second step involves constructing the truth table listing the 7 display input signals, decimal number and corresponding 4 digit binary numbers.The figure below shows the truth table of a BCD to seven-segment decoder. In the truth table, there are 7 different output columns corresponding to each of the 7 segments.

Suppose the column for segment a shows the different combinations for which it is to be illuminated. So ‘a’ is active for the digits 0, 2, 3, 5, 6, 7, 8 and 9

Digit	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0

From the above truth table, the Boolean expressions of each output functions can be written as

$$a = F1 (A, B, C, D) = \sum m (0, 2, 3, 5, 6,7, 8, 9)$$

$$b = F2 (A, B, C, D) = \sum m (0, 1, 2, 3, 4, 7, 8, 9)$$

$$c = F3 (A, B, C, D) = \sum m (0, 1, 3, 4, 5, 6, 7, 8, 9)$$

$$d = F4 (A, B, C, D) = \sum m (0, 2, 3, 5, 6, 8)$$

$$e = F5 (A, B, C, D) = \sum m (0, 2, 6, 8)$$

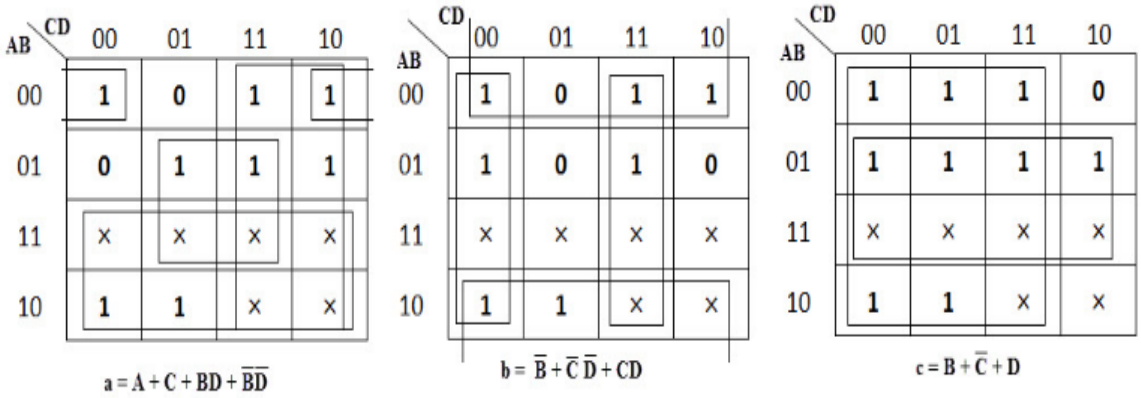
$$f = F6 (A, B, C, D) = \sum m (0, 4, 5, 6, 8, 9)$$

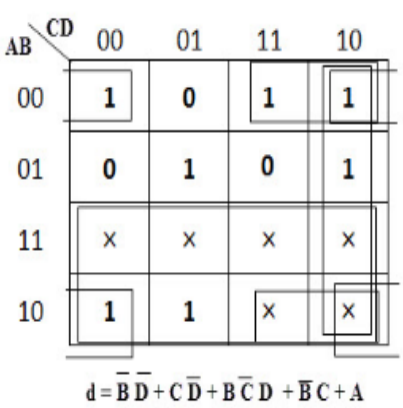
$$g = F7 (A, B, C, D) = \sum m (2, 3, 4, 5, 6, 8, 9)$$

Step 3: The third step involves constructing the Karnough's map for each output term and then simplifying them to obtain a logic combination of inputs for each output.

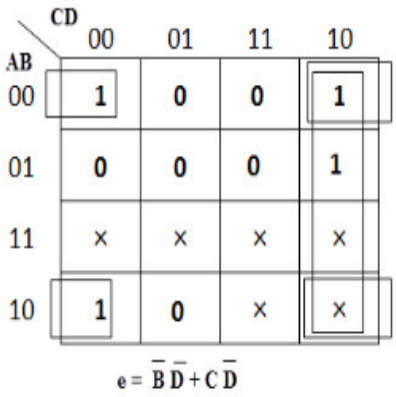
K-map Simplification

The below figures shows the k-map simplification for the common cathode seven-segment decoder in order to design the combinational circuit.

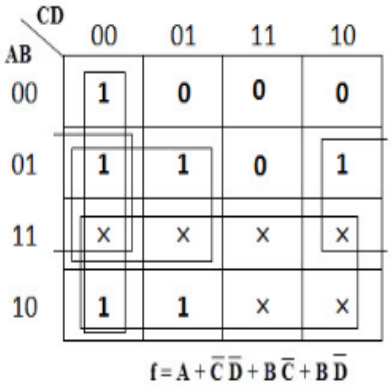




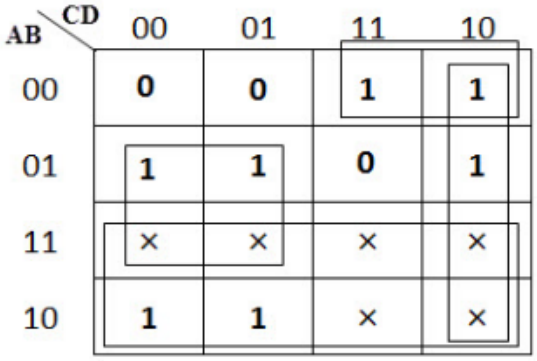
$d = \overline{B}\,\overline{D} + C\,\overline{D} + B\,\overline{C}\,D + \overline{B}\,C + A$



$e = \overline{B}\,\overline{D} + C\,\overline{D}$



$f = A + \overline{C}\,\overline{D} + B\,\overline{C} + B\,\overline{D}$



$g = \overline{B}\,C + C\,\overline{D} + B\,\overline{C} + B\,\overline{C} + A$

From the above simplification, we get the output values as

$$a = A + C + BD + \overline{B}\,\overline{D}$$

$$b = \overline{B} + \overline{C}\,\overline{D} + CD$$

$$c = B + \overline{C} + D$$

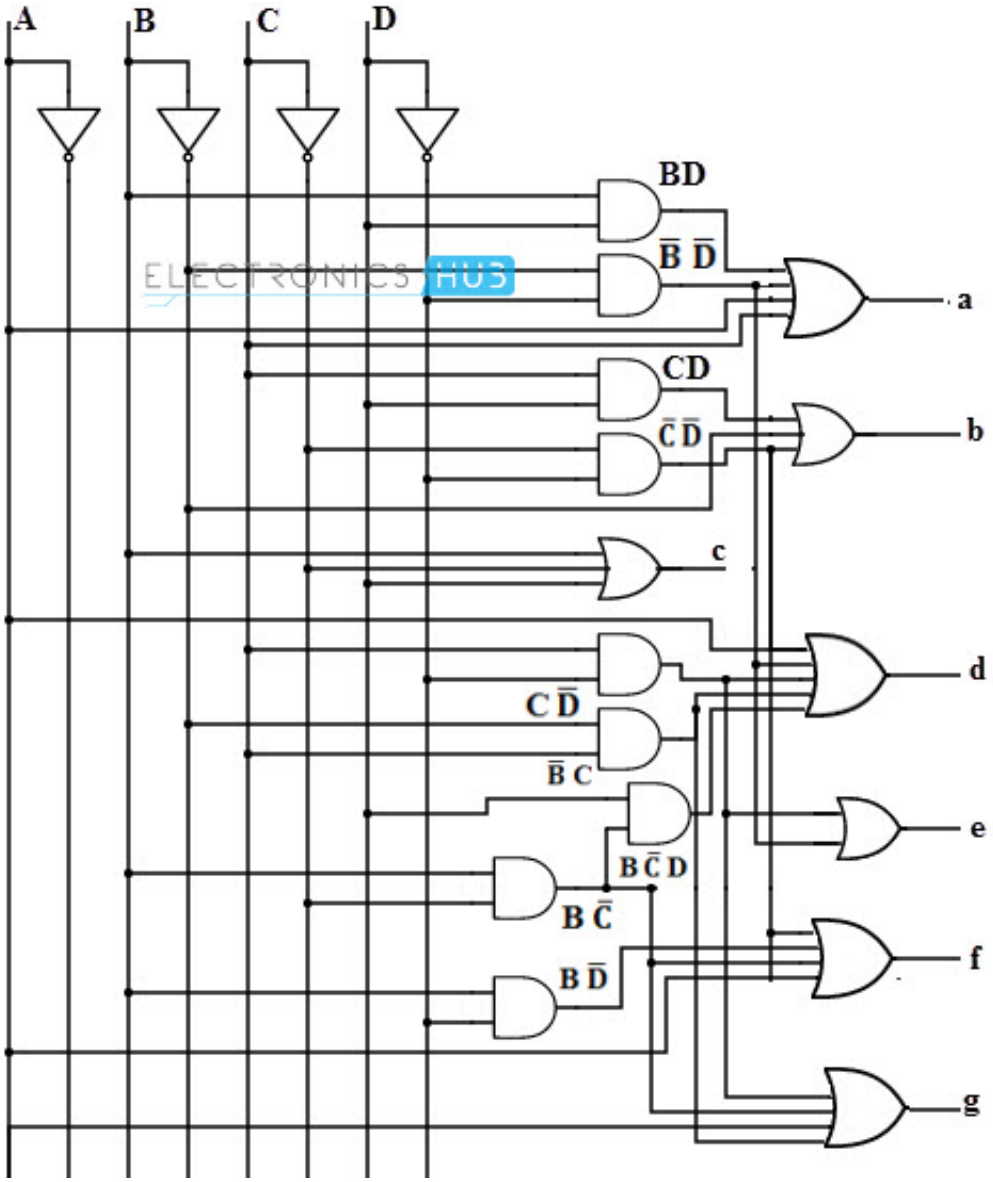
$$d = \overline{B}\,\overline{D} + C\,\overline{D} + B\,\overline{C}\,D + \overline{B}\,C + A$$

$$e = \overline{B}\,\overline{D} + C\,\overline{D}$$

$$f = A + \overline{C}\,\overline{D} + B\,\overline{C} + B\,\overline{D}$$

$$g = A + B\,\overline{C} + \overline{B}\,C + C\,\overline{D}$$

Step 4: The final step involves drawing a combinational logic circuit for each output signal. Once the task was accomplished, a combinational logic circuit can be drawn using 4 inputs (A, B, C, D) and a 7- segment display (a, b, c, d, e, f, g) as output.



Display Decoder Circuit Operation

The circuit operation can be understood through the truth table itself. When all the inputs are connected to low logic, the output of the combinational logic circuit would be so as to drive all the output LEDs except ‘g’ to conduction. Thus the number 0 will be displayed. Similar operation would take place for all other combinations of the input switches.

Applications of Display Decoder Circuit

1. This circuit can be modified using timers and counters to display the number of clock pulses.
2. This circuit can be modified to develop an alphabet display system instead of a decimal number display system.
3. It can be used as a timer circuit.

In digital imaging, a **pixel**, **pel**, **dots**, or **picture element** is a physical point in a raster image, or the smallest addressable element in an all points addressable display device, so it is the smallest controllable element of a picture represented on the screen. The address of a pixel corresponds to its physical coordinates. LCD pixels are manufactured in a two-dimensional grid, and are often represented using dots or squares, but CRT pixels correspond to their timing mechanisms and sweep rates.

Each pixel is a sample of an original image, more samples typically provide more accurate representations of the original. The intensity of each pixel is variable. In color image systems, a color is typically represented by three or four component intensities such as red, green, and blue, or cyan, magenta, yellow, and black.

Bits per pixel

The number of distinct colors that can be represented by a pixel depends on the number of bits per pixel (bpp). A 1 bpp image uses 1-bit for each pixel, so each pixel can be either on or off. Each additional bit doubles the number of colors available, so a 2 bpp image can have 4 colors, and a 3 bpp image can have 8 colors:

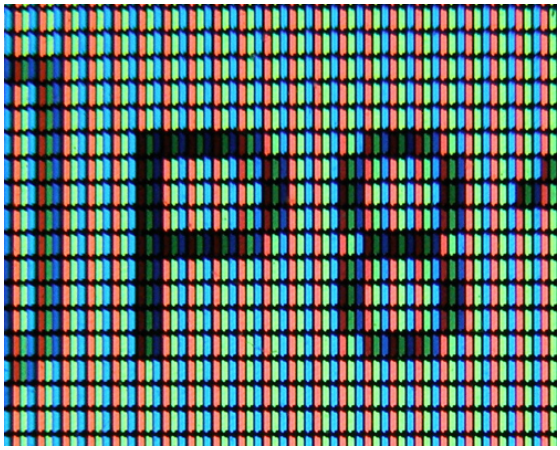
- 1 bpp, 2^1 = 2 colors (monochrome)
- 2 bpp, 2^2 = 4 colors
- 3 bpp, 2^3 = 8 colors

...

- 8 bpp, 2^8 = 256 colors
- 16 bpp, 2^{16} = 65,536 colors ("Highcolor")
- 24 bpp, 2^{24} = 16,777,216 colors ("Truecolor")

For color depths of 15 or more bits per pixel, the depth is normally the sum of the bits allocated to each of the red, green, and blue components. Highcolor, usually meaning 16 bpp, normally has five bits for red and blue, and six bits for green, as the human eye is more sensitive to errors in green than in the other two primary colors. For applications involving transparency, the 16 bits may be divided into five bits each of red, green, and blue, with one bit left for transparency. A 24-bit depth allows 8 bits per component. On some systems, 32-bit depth is available:

this means that each 24-bit pixel has an extra 8 bits to describe its opacity (for purposes of combining with another image).



A photograph of sub-pixel display elements on a laptop's LCD screen

Nobel Prize for Blue LED

In 2014, **Isamu Akasaki**, **Hiroshi Amano** and **Shuji Nakamura** were rewarded for having invented a new energy-efficient and environment-friendly light source – the blue light-emitting diode (LED). In the spirit of Alfred Nobel, the Prize rewards an invention of greatest benefit to mankind; using blue LEDs, white light can be created in a new way. With the advent of LED lamps, we now have more long-lasting and more efficient alternatives to older light sources.

Red and green diodes had been around for a long time but without blue light, white lamps could not be created. Despite considerable efforts, both in the scientific community and in industry, the blue LED had remained a challenge for three decades.

White LED lamps emit a bright white light, are long-lasting and energy-efficient. They are constantly improved, getting more efficient with higher luminous flux (measured in lumen) per unit electrical input power (measured in watt). The most recent record is just over 300 lm/W, which can be compared to 16 for regular light bulbs and close to 70 for fluorescent lamps. As about one fourth of world electricity consumption is used for lighting purposes, the LEDs contribute to saving the Earth's resources. Materials consumption is also diminished as LEDs last up to 100,000 hours, compared to 1,000 for incandescent bulbs and 10,000 hours for fluorescent lights.

Bibliography

While making this Scribe, some help and content were taken from:

Books: Fundamentals of Logic Design- Larry L. Kinney & Charles H. Roth, Jr.

Websites: Wikipedia, Google