Logic and bit operations

- **Computers** represent information by bits.
- A **bit** has two possible values, namely zero and one. This meaning of the word comes from **binary digit**, since zeroes and ones are the digits used in binary representations of numbers.
- The statistician **John Tukey** introduced this terminology in 1946. (There were several other suggested words for a binary digit, including **binit** and **bigit**, that never were widely accepted.)

John Tukey (1915-2000)
Logic and bit operations

- A bit can be used to represent a truth value, since there are two truth values, true and false.
- A variable is called a **Boolean variable** if its values are either true or false.
- Computer **bit operations** correspond to the logical connectives. We will also use the notation OR, AND and XOR for $\lor$, $\land$ and exclusive $\lor$.
- A **bit string** is a sequence of zero or more bits. The length of the string is the number of bits in the string.
Bitwise operations

- We can extend bit operations to bit strings. We define bitwise OR, bitwise AND and bitwise XOR of two strings of the same length to be the strings that have as their bits the OR, AND and XOR of the corresponding bits in the two strings.
Example: Find the bitwise OR, bitwise AND and bitwise XOR of the bit strings

\begin{align*}
01101 & 10110 \\
11000 & 11101
\end{align*}
Example: Find the bitwise OR, bitwise AND and bitwise XOR of the bit strings

```
01101  10110
11000  11101
```

Solution: The bitwise OR is

```
11101  11111
```
Example: Find the bitwise OR, bitwise AND and bitwise XOR of the bit strings

\[
\begin{align*}
01101 & \quad 10110 \\
11000 & \quad 11101
\end{align*}
\]

Solution: The bitwise OR is

\[
11101 \quad 11111
\]

The bitwise AND is

\[
01000 \quad 10100
\]
Example: Find the bitwise OR, bitwise AND and bitwise XOR of the bit strings

01101 10110
11000 11101

Solution: The bitwise OR is

11101 11111

The bitwise AND is

01000 10100

and the bitwise XOR is

10101 01011
Boolean algebra

• Circuits in computers and other electronic devices have inputs, each of which is either a 0 or a 1, and produce outputs that are also 0s and 1s.
• Circuits can be constructed using any basic element that has two different states. Such elements include switches that can be in either the “on” or the “off” position and optical devices that can either be lit or unlit.
• In 1938 Claude Shannon showed how the basic rules of logic, first given by George Boole in 1854 in his *The Laws of Thought*, could be used to design circuits.

Claude Shannon (1916-2001)
Boole’s rules of logic form the basis for Boolean algebra.
In the following we develop the basic properties of Boolean algebra.
The operation of a circuit is defined by a Boolean function that specifies the value of an output for each set of inputs.
The first step in constructing a circuit is to represent its Boolean function by an expression built up using the basic operations of Boolean algebra.
Minimization of circuits

- The expression that we obtain may contain many more operations than are necessary to represent the function.
- We will describe later methods for finding an expression with the minimum number of sums and products that represents a Boolean function.
- The procedures that we will develop are important in the design of efficient circuits.
Boolean operations

- Boolean algebra provides the operations and the rules for working with the set \( \{0, 1\} \).
- The three most used operations in Boolean algebra are complementation, the Boolean sum and the Boolean product. They correspond to the logical connectives \(^\neg\), \(\lor\) and \(\land\).
- The complement of an element, denoted by bar, is defined by \(\overline{0} = 1\) and \(\overline{1} = 0\).
- The Boolean sum, denoted by + or OR has the following values:
  \[
  1 + 1 = 1, 1 + 0 = 1, 0 + 1 = 1, 0 + 0 = 0.
  \]
- The Boolean product, denoted by \(\cdot\) or by AND, has the following values:
  \[
  1 \cdot 1 = 1, 1 \cdot 0 = 0, 0 \cdot 1 = 0, 0 \cdot 0 = 0.
  \]
• When there is no danger of confusion, the symbol \( \cdot \) can be deleted, just as in writing algebraic products.

• The rules of precedence for connectives still apply: complements have precedence over products and products have precedence over sums.

• **Example:** Find the value of \( 1 \cdot 0 + (0 + 1) \).

• The results about propositional calculus can be translated in results about Boolean algebras.
Boolean variables and Boolean functions

- Let $B = \{0, 1\}$. The variable $x$ is called a **Boolean variable** if it assumes only values from $B$.
- A function from $B^n = \{(x_1, x_2, \ldots, x_n) | x_i \in B, 1 \leq i \leq n\}$ to $B$ is called a **Boolean function** of degree $n$. 

Logic in Computer Science: Logic Gates

CS2209, Applied Logic for Computer Science
The values of a Boolean function are often displayed in tables (truth tables). For instance, the Boolean function $F(x, y)$ with the value 1 when $x = 1$ and $y = 0$ and the value 0 for all other choices of $x$ and $y$ can be represented by:

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$F(x, y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Exercise:** Find the values of the Boolean function represented by $F(x, y, z) = xy + \overline{z}$. 
Boolean functions contd.

- Two Boolean functions $F$ and $G$ of $n$ variables are equal iff
  \[ F(b_1, \ldots, b_n) = G(b_1, \ldots, b_n) \]
  whenever $b_1, \ldots, b_n$ belong to $B$.

- Boolean functions can be represented using expressions made up from variables and Boolean operations.

- Each Boolean expression (corresponding to a formula in propositional calculus) represents a Boolean function.

- Two Boolean expressions that represent the same function are called equivalent.

- **Exercise:** How many different Boolean functions of degree $n$ are there?
The number of Boolean functions of degree $n$. 

<table>
<thead>
<tr>
<th>Degree</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>256</td>
</tr>
<tr>
<td>4</td>
<td>65,536</td>
</tr>
<tr>
<td>5</td>
<td>4,294,967,296</td>
</tr>
<tr>
<td>6</td>
<td>18,446,744,073,709,551,616</td>
</tr>
</tbody>
</table>
Until now we have focused on Boolean functions and expressions. However, the results that can be established in this framework can be translated into results about propositions or sets.

Because of this, it is useful to define Boolean algebras abstractly. Once it is shown that a particular structure is a Boolean algebra, then all results established about Boolean algebras apply to this particular structure.

Boolean algebras can be defined by specifying the properties that operations must satisfy, as is done in the following definition.
A **Boolean algebra** is a set $B$ with two binary operations $+$ and $\cdot$, elements 0 and 1 and a unary operation $\overline{}$, such that the following properties hold for all $x, y, z$ in $B$:

- **Identity laws**: $x + 0 = x$ and $x \cdot 1 = x$.
- **Domination laws**: $x + \overline{x} = 1$, $x \cdot \overline{x} = 0$.
- **Associative laws**: $(x + y) + z = x + (y + z)$,
  $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
- **Commutative laws**: $x + y = y + x$, $x \cdot y = y \cdot x$
- **Distributive laws**: $x + (y \cdot z) = (x + y) \cdot (x + z)$ and $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.

Using the laws given in this definition, it is possible to prove many other laws that hold for every Boolean algebra.
Boolean algebra examples

- $B = \{0, 1\}$, with the OR (+) and AND (·) operations and the complement operator satisfy all these properties.
- The set of propositions in $n$ variables, with the $\lor$ and $\land$ operators, 1 and 0, and the negation operator, also satisfy all the properties of a Boolean algebra.
- The set of subsets of a universal set $U$, with the union and intersection operations, the empty set and the universal set, and the set complementation operator, is a Boolean algebra.
- So, to establish results about each of Boolean expressions, propositions, and sets, we need only prove results about abstract Boolean algebras.
Representing Boolean functions

- Given the values of a Boolean function, how can a Boolean expression that represents the function be found?

This problem is solved by showing that any Boolean function may be represented by a Boolean sum of Boolean products of the variables and their complements.

The method is similar to the solution to a problem encountered in propositional calculus: "Given a truth table, how can a formula that has that truth table be found?" Answer: The solution is a disjunction of minterms (conjunction of literals and their negations).
Representing Boolean functions

- Given the values of a Boolean function, how can a Boolean expression that represents the function be found?
- This problem is solved by showing that any Boolean function may be represented by a Boolean sum of Boolean products of the variables and their complements.
Representing Boolean functions

- Given the values of a Boolean function, how can a Boolean expression that represents the function be found?
- This problem is solved by showing that any Boolean function may be represented by a Boolean sum of Boolean products of the variables and their complements.
- The method is similar to the solution to a problem encountered in propositional calculus: “Given a truth table, how can a formula that has that truth table be found?”
Representing Boolean functions

- Given the values of a Boolean function, how can a Boolean expression that represents the function be found?
- This problem is solved by showing that any Boolean function may be represented by a Boolean sum of Boolean products of the variables and their complements.
- The method is similar to the solution to a problem encountered in propositional calculus: “Given a truth table, how can a formula that has that truth table be found?”

Answer: The solution is a disjunction of minterms (conjunction of literals and their negations).
• The answer to this problem shows that every Boolean function can be represented using the three Boolean operators $+, \cdot, \bar{\cdot}$.
The answer to this problem shows that every Boolean function can be represented using the three Boolean operators $+, \cdot, \bar{}$.

Is there a smaller set of operators that can be used to represent all Boolean functions?

The answer is similar to the solution to the problem: "Does there exist an adequate set of connectives with fewer than three elements?"

Answer: Yes (For example, the Sheffer stroke). Both of these problems have practical importance in circuit design.
- The answer to this problem shows that every Boolean function can be represented using the three Boolean operators $+, \cdot, \bar{\cdot}$.

- Is there a smaller set of operators that can be used to represent all Boolean functions?

- The answer is similar to the solution to the problem: “Does there exist an adequate set of connectives with fewer than three elements?”

Answer: Yes (For example, the Sheffer stroke).
• The answer to this problem shows that every Boolean function can be represented using the three Boolean operators +, ·, \( \bar{\cdot} \).

• Is there a smaller set of operators that can be used to represent all Boolean functions?

• The answer is similar to the solution to the problem: “Does there exist an adequate set of connectives with fewer than three elements?”

• **Answer:** Yes (For example, the Sheffer stroke).
• The answer to this problem shows that every Boolean function can be represented using the three Boolean operators $+, \cdot, \bar{\cdot}$.

• Is there a smaller set of operators that can be used to represent all Boolean functions?

• The answer is similar to the solution to the problem: “Does there exist an adequate set of connectives with fewer than three elements?”

• Answer: Yes (For example, the Sheffer stroke).

• Both of these problems have practical importance in circuit design.
Example

Find a Boolean expression that represents the function $F(x, y, z)$ given in the following table:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Answer: $xy + xz$

The disjunctive normal form obtained is also called sum-of-products expansion.
Example

Find a Boolean expression that represents the function $F(x, y, z)$ given in the following table:

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Answer: $xy\bar{z} + \bar{x}y\bar{z}$
Example

Find a Boolean expression that represents the function $F(x, y, z)$ given in the following table:

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Answer: $xy\overline{z} + \overline{x}y\overline{z}$

The disjunctive normal form obtained is also called sum-of-products expansion.
Functional completeness

• Every Boolean function can be expressed as a Boolean sum of minterms (products of Boolean variables and their complements). Since every Boolean function can be represented using the operators \{+, \cdot, \bar{\cdot}\}, we say that this set of operators is functionally complete.

• Can we find a smaller functionally complete set of operators?

• Yes. \{+, \bar{\cdot}\}, \{\cdot, \bar{\cdot}\}, \{|\}\ (The NAND operator or Sheffer stroke) and \{\downarrow\} (The NOR operator) are all functionally complete.

• The NAND operator is defined by \(1 | 1 = 0\) and \(1 | 0 = 0\), \(\bar{1} | 1 = 0\), \(\bar{1} | 0 = 1\).

• The NOR operator is defined by \(1 \downarrow 1 = 1\), \(1 \downarrow 0 = 0\), \(0 \downarrow 1 = 0\) and \(0 \downarrow 0 = 1\).
Functional completeness

- Every Boolean function can be expressed as a Boolean sum of minterms (products of Boolean variables and their complements). Since every Boolean function can be represented using the operators \{+, \cdot, \bar{\cdot}\}, we say that this set of operators is **functionally complete**.

- Can we find a smaller functionally complete set of operators?
Functional completeness

- Every Boolean function can be expressed as a Boolean sum of minterms (products of Boolean variables and their complements). Since every Boolean function can be represented using the operators \{+, \cdot, \bar{\cdot}\}, we say that this set of operators is functionally complete.
- Can we find a smaller functionally complete set of operators?
- Yes. \{+, \bar{\cdot}\}, \{\cdot, \bar{\cdot}\}, \{|\} (The NAND operator or Sheffer stroke) and \{\downarrow\} (The NOR operator) are all functionally complete.
- The NAND operator is defined by \(1|1 = 0\) and \(1|0 = 0|1 = 0|0 = 1\).
- The NOR operator is defined by \(1 \downarrow 1 = 1 \downarrow 0 = 0 \downarrow 1 = 0\) and \(0 \downarrow 0 = 1\).
Electronic circuits and logic gates

- Boolean algebra is used to model the circuitry of electronic devices, including electronic computers.
- Each input and output of such a device can be thought of as a member of the set \(\{0, 1\}\).
- An electronic computer is made up of a number of circuits.
- Each circuit can be designed using the rules of Boolean algebra.
- The basic elements of circuits are called logic gates.
- A gate is an electronic device that operates on a collection of binary inputs and produces a binary output.
Logic gates

- Each type of gate implements a Boolean operation.
- We will define several types of gates. Using these gates we will apply the rules of Boolean algebra to design circuits that perform a variety of tasks.
- The circuits that we study give output that depends only on the inputs, and not the current state of the circuit. In other words, these circuits have no memory abilities. Such circuits are called combinatorial circuits.
The inverter

- The inverter accepts a Boolean value as input and produces the complement of this value as its output.
- The symbol used for an inverter is shown below.
- The input to the inverter is shown on the left side entering the element, and the output is shown on the right side leaving the element.

(a) Inverter
The OR gate

- The second type gate is the OR gate.
- The input to this gate are the values of two or more Boolean variables.
- The output is the Boolean sum or their values.
- The symbol used for an OR gate is shown below.
- The inputs to the OR gate are shown on the left side entering the element and the output is shown on the right side leaving the element.

(b) OR gate
The AND gate

- The third type of gate is the AND gate.
- The inputs to this gate are the values of two or more Boolean variables.
- The output is the Boolean product or their values.
- The symbol used for an AND gate is shown below.
- The inputs to an AND gate are shown on the left side entering the element, and the output is shown on the right side leaving the element.
Gates with multiple inputs

We will permit multiple inputs to AND and OR gates, as below.

\[ x_1 x_2 \cdots x_n \]

\[ x_1 + x_2 + \cdots + x_n \]
Circuits: Combinations of gates

- Combinatorial circuits can be constructed using a combination of inverters, OR gates and AND gates.
- When combinations of circuits are formed, some gates may share inputs. This is shown in one of two ways in depictions of circuits.
- One method is to use branchings to indicate all the gates that use a given input.
- The other method is to indicate this input separately for each gate.
- Note that output from a gate may be used as input by one or more elements,
How to draw circuits

Both drawings below depict the circuit with output $xy + \overline{xy}$.
Example: Construct the circuits that produce the following outputs:

(a) \((x + y)x\)

(b) \(x(y + \overline{z})\)

(c) \((x + y + z)(\overline{x} \overline{y} \overline{z})\).
Example answers

(a) \((x + y)\overline{x}\)
(b) \( \overline{x(y + \overline{z})} \)
(c) \((x + y + z)(\overline{x} \overline{y} \overline{z})\).
Example 1. A committee of three individuals decides issues for an organization. Each individual votes either yes or no for each proposal that arises. A proposal is passed if it receives at least two yes votes. Design a circuit that determines whether a proposal passes.
Example 1. A committee of three individuals decides issues for an organization. Each individual votes either yes or no for each proposal that arises. A proposal is passed if it receives at least two yes votes. Design a circuit that determines whether a proposal passes.

Solution: Let $x = 1$ be 1 if the first individual says yes and $x = 0$ if this individual says no. Similarly for $y$ and $z$.

Then a circuit must be designed that produces output 1 from the inputs $x, y, z$ when two or more of $x, y, z$ are 1.
Example 1. A committee of three individuals decides issues for an organization. Each individual votes either yes or no for each proposal that arises. A proposal is passed if it receives at least two yes votes. Design a circuit that determines whether a proposal passes.

Solution: Let $x = 1$ be 1 if the first individual says yes and $x = 0$ if this individual says no. Similarly for $y$ and $z$.

Then a circuit must be designed that produces output 1 from the inputs $x, y, z$ when two or more of $x, y, z$ are 1.

One representation of the Boolean function that has these output values is $xy + xz + yz$. 
Circuit for majority voting

\[ xy + xz + yz \]
Example 2. Sometimes light fixtures are controlled by more than one switch. Circuits need to be designed so that flipping any one of the switches for the fixture turns the light on when it is off and turns the light off when it is on. Design circuits that accomplish this when there are two switches and when there are three switches.
Example 2. Sometimes light fixtures are controlled by more than one switch. Circuits need to be designed so that flipping any one of the switches for the fixture turns the light on when it is off and turns the light off when it is on. Design circuits that accomplish this when there are two switches and when there are three switches.

Solution:
The two-switch case.

Let $x = 1$ when the first switch is closed and 0 when it is open, and let $y = 1$ when the second switch is closed and 0 when it is open.
Example 2. Sometimes light fixtures are controlled by more than one switch. Circuits need to be designed so that flipping any one of the switches for the fixture turns the light on when it is off and turns the light off when it is on. Design circuits that accomplish this when there are two switches and when there are three switches.

Solution:
The two-switch case.

Let $x = 1$ when the first switch is closed and 0 when it is open, and let $y = 1$ when the second switch is closed and 0 when it is open. Let $F(x, y) = 1$ when the light is on and $F(x, y) = 0$ when the light is off.
We can arbitrarily decide that the light will be on when both switches are closed, so that $F(1, 1) = 1$. 

When one of the two switches is opened, the light goes off, so $F(0, 1) = F(1, 0) = 0$. When the other switch is opened the light goes on, so that $F(0, 0) = 1$. 

We see that $F(x, y) = xy + x y$. 

We can arbitrarily decide that the light will be on when both switches are closed, so that $F(1, 1) = 1$.
This determines all the other values of $F$. 
We can arbitrarily decide that the light will be on when both switches are closed, so that $F(1, 1) = 1$.

This determines all the other values of $F$.

When one of the two switches is opened, the light goes off, so $F(0, 1) = F(1, 0) = 0$. When the other switch is opened the light goes on, so that $F(0, 0) = 1$. 
We can arbitrarily decide that the light will be on when both switches are closed, so that $F(1, 1) = 1$.

This determines all the other values of $F$.

When one of the two switches is opened, the light goes off, so $F(0, 1) = F(1, 0) = 0$. When the other switch is opened the light goes on, so that $F(0, 0) = 1$.

We see that $F(x, y) = xy + \overline{x} \overline{y}$. 
Circuit for two-switch light fixture

\[ x \quad \rightarrow \quad x'y' \quad + \quad xy \]

\[ y \quad \rightarrow \quad x'y' \quad + \quad xy \]

\[ x \quad \rightarrow \quad \overline{x} \quad \rightarrow \quad xy \]

\[ y \quad \rightarrow \quad \overline{y} \quad \rightarrow \quad xy \]
Example 2 contd.

Example: The three-switch case.
In this case we have three variables, $x, y, z$ that take value 1 (switch closed) or 0 (switch open).
Example: The three-switch case.
In this case we have three variables, $x, y, z$ that take value 1 (switch closed) or 0 (switch open).
Let $F(x, y, z) = 1$ when the light is on and 0 when it is off.
Example: The three-switch case.
In this case we have three variables, $x, y, z$ that take value 1 (switch closed) or 0 (switch open).

Let $F(x, y, z) = 1$ when the light is on and 0 when it is off.

We can arbitrarily specify that the light be on when all three switches are closed, so that $F(1, 1, 1) = 1$. 
Example: The three-switch case.
In this case we have three variables, \(x, y, z\) that take value 1 (switch closed) or 0 (switch open).

Let \(F(x, y, z) = 1\) when the light is on and 0 when it is off.

We can arbitrarily specify that the light be on when all three switches are closed, so that \(F(1, 1, 1) = 1\).

This determines all the other values of \(F\).
Example 2 contd.

Example: The three-switch case.
In this case we have three variables, \(x, y, z\) that take value 1 (switch closed) or 0 (switch open).

Let \(F(x, y, z) = 1\) when the light is on and 0 when it is off.

We can arbitrarily specify that the light be on when all three switches are closed, so that \(F(1, 1, 1) = 1\).

This determines all the other values of \(F\).

The function \(F\) can be represented as

\[
xyz + x\overline{y} \overline{z} + \overline{x} \overline{y} z + x\overline{y} \overline{z} + \overline{x} y \overline{z} + \overline{x} \overline{y} \, z.
\]
Circuit for three-switch light fixture
Adders

- We will illustrate how logic circuits can be used to carry out addition of two positive integers from their binary expansions.
- We will build up the circuitry to do this addition from some component circuits.
- First we build a circuit that can be used to find $x + y$ when $x$ and $y$ are two bits.
- The input to our circuit will be $x$ and $y$, since each of these have value 0 or 1.
- The output will consist of two bits, namely $s$ and $c$ where $s$ is the sum bit and $c$ is the carry bit.
The half-adder

- This circuit is called a **multiple output circuit**.
- The circuit we are designing is called the **half-adder** since it adds two bits, without considering a carry from the previous addition.
- We show the input and output for the half adder in the following table.
The multiple output half-adder truth table

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>s</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

From the preceding table we see that $c = xy$ and $x\bar{y} + \bar{x}y = (x + y)(xy)$. 
Circuit for half-adder

\[ \text{Sum} = (x + y)(\overline{xy}) \]

\[ \text{Carry} = xy \]
The full adder

**Input**: bits \( x \) and \( y \) and the carry bit \( c_i \).

**Output**: The sum bit \( s \) and the carry bit \( c_{i+1} \).

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( c_i )</th>
<th>( s )</th>
<th>( c_{i+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Note that

\[ s = xyc_i + x\overline{y} \overline{c_i} + \overline{x} y\overline{c_i} + \overline{x} \overline{y} c_i. \]

\[ c_{i+1} = xyc_i + xy\overline{c_i} + xy c_i + \overline{x} yc_i. \]
Circuit for full adder using half adders

However, instead of building the full adder from scratch, we will use half adders to produce the desired outputs.
Adding three-bit integers

Finally, we show how full and half adders can be used to add the three-bit integers \((x_2 x_1 x_0)_2\) and \((y_2 y_1 y_0)_2\) to produce the sum \((s_3 s_2 s_1 s_0)_2\). Note that \(s_3\), the highest-order bit in the sum is given by the carry \(c_2\).
Acknowledgements

The figures in this set of notes are from