When data is sent from one device to another, it is important to consider how that data is transmitted. It is also important to ensure that the data hasn't been changed in any way.

The internet has now become an integral part of all of our lives. This chapter will consider some of the important technologies going on in the background which support the internet.

Data transmission refers to the movement of data in the form of bits between two or more digital devices. This transfer of data takes place via some form of transmission media (for example, coaxial cable, fiber optics etc.)

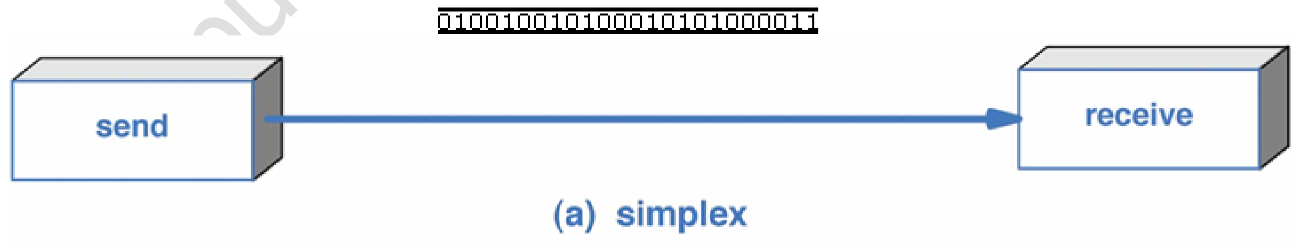
Data transmission

Data transmission can be either over a short distance (for example, from computer to printer) or over longer distances (for example, over a telephone network).
Essentially, three factors need to be considered when transmitting data (each factor has to be agreed by both sender and receiver for this to work without error):

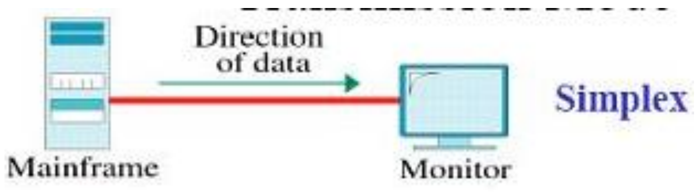
- the direction of the data transmission (i.e. in one direction only or in both directions)
- the method of transmission (how many bits are sent at the same time)
- the method of synchronization between the two devices.

Simplex, half-duplex and full-duplex

SIMPLEX DATA TRANSMISSION: is in *one direction* only (i.e. from sender to receiver).
Example: data being sent from a computer to a printer.

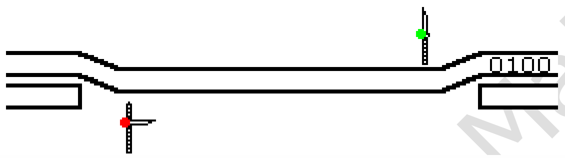


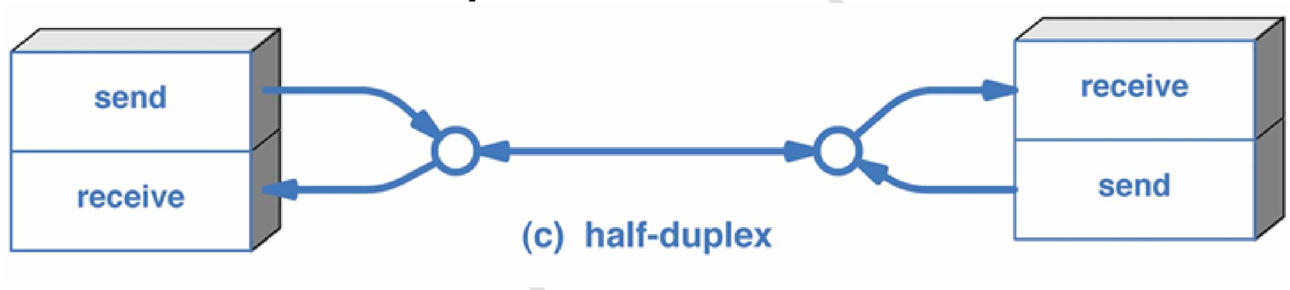
Simplex is one direction. A good example would be your keyboard to your CPU. The CPU never needs to send characters to the keyboard but the keyboard always send characters to the CPU. In many cases, Computers almost always send characters to printers, but printers usually never send characters to computers.



HALF-DUPLEX DATA TRANSMISSION is in *both directions* but *not* at the same time (i.e. data can be sent from 'A' to 'B' or from 'B' to 'A' along the same line, but not at the same time).

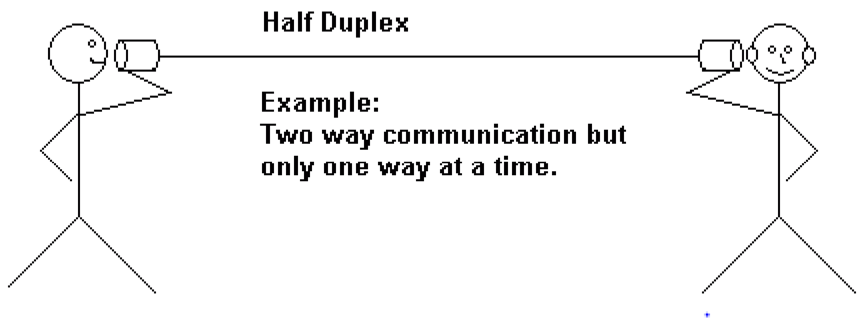
Example: a phone conversation between two people where only one person speaks at a time a Walkie Talkie.



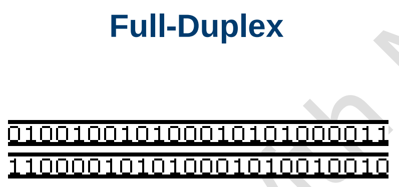


Half-Duplex is like the dreaded "one lane" road you may have run into at construction sites. Only one direction will be allowed through at a time. Railroads have to deal with this scenario more often since it's cheaper to lay a single track.

A dispatcher will hold a train up at one end of the single track until a train going the other direction goes through. The only example I could think of for Half-Duplex is actually a Parallel interface. Even though parallel is eight lanes, data travels through the lanes in the same direction at the same time but never in both directions at the same time. The IEEE-1284 allows printers to send messages to the computer. The printer cannot send these messages while the computer is sending characters but when the computer stops sending characters, then the printer can send messages back.



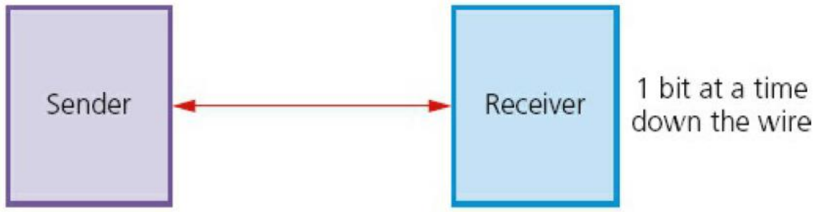
FULL-DUPLEX DATA TRANSMISSION is in *both directions simultaneously* (i.e. data can be sent from ‘A’ to ‘B’ and from ‘B’ to ‘A’ along the same line, *both at the same time*). Example: broadband connection on a phone line.



Full-Duplex is like the ordinary two-lane highway. In some cases, where traffic is heavy enough, a railroad will decide to lay a double track to allow trains to pass in both directions. In communications, this is most common with networking. Our fiber optic hubs have two connectors on each port, one for each lane of a two-lane roadway. Full-Duplex fiber is two cables bundled or tied together to form the two-lane roadway. In 100Base-TX, the two lanes are housed in the same jacket. RS232 was also designed to handle Full-Duplex but some of our short haul modems and converters give the user the option to go Half-Duplex or Simplex to reduce the number of conductors needed to connect between them.

Serial and parallel data transmission

SERIAL DATA TRANSMISSION is when data is sent, *one bit at a time*, over *a single wire or channel* (bits are sent one after the other in a single stream).

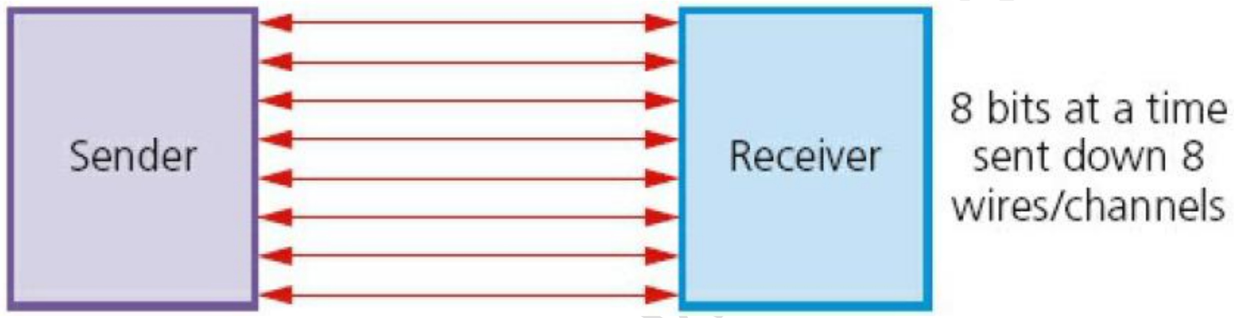


(Note: bits can be transmitted as simplex, half-duplex or full-duplex.)

This method of data transmission works well over long distances. However, data is transmitted at a slower rate than parallel data transmission. Since only one wire or channel is used, there is no problem of data arriving at its destination out of synchronisation.

An example of its use is sending data from a computer to a modem for transmission over a telephone line.

PARALLEL DATA TRANSMISSION: is when *several bits of data (usually 1 byte)* are sent down *several wires or channels at the same time*; one wire or channel is used to transmit each bit.



(Note: bits can be transmitted as simplex, half-duplex or full-duplex.)

This method of data transmission works very well over short distances (over longer distances, the bits can become 'skewed' – this means they will no longer be synchronised).

It is, however, a faster method of data transmission than serial.
An example of its use is when sending data to a printer from a computer using a ribbon connector.

A common use for serial data transmission is **(Universal Serial Bus (USB)**.

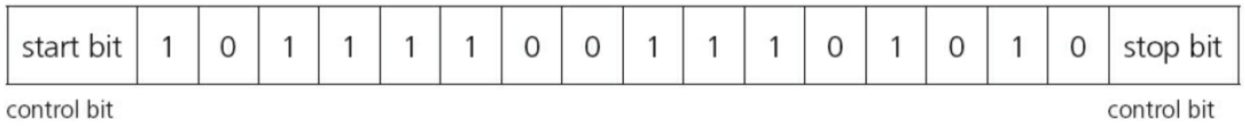
Parallel data transmission is used in the **internal electronics of the computer** system. The pathways between the CPU and the memory all use this method of data transmission.

Integrated circuits, buses and other internal components all **use parallel data transmission** because of the need for high speed data transfer.

The use of 8-bit, 16-bit, 32-bit and 64-bit buses, for example, allow much faster data transmission rates than could be achieved with single channel serial data transfer. An internal clock is used to ensure the correct timing of data transfer; it is essentially synchronous in nature and the short distances between components mean that none of the issues described earlier have any real impact on the accuracy of the data.

ASYNCHRONOUS DATA TRANSMISSION:

Asynchronous data transmission refers to data being transmitted in an agreed bit pattern. Data bits (1s and 0s) are grouped together and sent with **CONTROL BITS**:



This means that the receiver of the data knows when the data starts and when it ends.

This prevents data becoming mixed up; without these control bits, it would be impossible to separate groups of data as they arrived.

SYNCHRONOUS DATA TRANSMISSION

Synchronous data transmission is a continuous stream of data (unlike asynchronous data which is sent in discrete groups). The data is accompanied by timing signals generated by an internal clock. This ensures that the sender and receiver are synchronised with each other. The receiver counts how many bits (1s and 0s) were sent and then reassembles them into bytes of data.

The timing must be very accurate here since there are no control bits sent in this type of data transmission. However, it is a faster data transfer method than asynchronous and is therefore used where this is an important issue (for example, in network communications).

Universal Serial Bus (USB)

The **UNIVERSAL SERIAL BUS (USB)** is an asynchronous serial data transmission method. It has quickly become the standard method for transferring data between a computer and a number of devices. Essentially, the USB cable consists of:

- a four-wire shielded cable
- two of the wires are used for power and the earth
- two of the wires are used in the data transmission.

When a device is plugged into a computer using one of the USB ports:

- the computer automatically detects that a device is present (this is due to a small change in the voltage level on the data signal wires in the cable)
- the device is automatically recognised, and the appropriate **DEVICE DRIVER** is loaded up so that computer and device can communicate effectively
- if a new device is detected, the computer will look for the device driver which matches the device; if this is not available, the user is prompted to download the appropriate software.

✓	✗
Devices plugged into the computer are automatically detected; device drivers are automatically uploaded	–
The connectors can only fit one way; this prevents incorrect connections being made	The maximum cable length is presently about 5 metres
This has become the industry standard; this means that considerable support is available to users	–
Several different data transmission rates are supported	The present transmission rate is limited to less than 500 megabits per second
Newer USB standards are backward compatible with older USB standards	The older USB standard (e.g. 1.1) may not be supported in the near future

Error-checking methods

Following data transmission, there is always the risk that the data has been corrupted or changed in some way. This can occur whether data is being transmitted over short distances or over long distances. Checking for errors is important since computers aren't able to check that text is correct; they can only recognise whether a word is in their built-in dictionary or not.
Whilst you probably had little problem understanding this text, a computer would be unable to make any sense of it.

This is why error checking is such an important part of computer technology. This section considers a number of ways that can be used to check for errors so that you don't end up with text as shown in the example above!

A number of methods exist which can detect errors and, in some cases, actually correct the error. The methods covered in this section are:

- parity checking
- automatic repeat request (ARQ)
- checksum
- echo checking.

Parity checking

PARITY CHECKING is one method used to check whether data has been changed or corrupted following transmission from one device or medium to another device or medium.

A byte of data, for example, is allocated a **PARITY BIT**. This is allocated before transmission takes place. Systems that use **EVEN PARITY** have an even number of 1-bits; systems that use **ODD PARITY** have an odd number of 1-bits.

Consider the following byte:

	1	1	0	1	1	0	0

If this byte is using even parity, then the parity bit needs to be 0 since there is already an even number of 1-bits (in this case, 4).

If odd parity is being used, then the parity bit needs to be 1 to make the number of 1-bits odd.

Therefore, the byte just before transmission would be: either (even parity)

0	1	1	0	1	1	0	0

parity bit

or (odd parity)

1	1	1	0	1	1	0	0

parity bit

Before data is transferred, an agreement is made between sender and receiver regarding which of the two types of parity are used. This is called **HANDSHAKING.**

Automatic Repeat Request (ARQ)

AUTOMATIC REPEAT REQUEST (ARQ) is another method used to check whether data has been correctly transmitted.

It uses an **ACKNOWLEDGEMENT** (a message sent by the receiver indicating that data has been received correctly) and **TIMEOUT** (this is the time allowed to elapse before an acknowledgement is received).

If an acknowledgement isn't sent back to the sender before timeout occurs, then the message is automatically resent.

Checksum

CHECKSUM is another way to check if data has been changed or corrupted following data transmission. Data is sent in blocks and an additional value, the checksum, is also sent at the end of the block of data.

To explain how this works, we will assume the checksum of a block of data is 1 byte in length. This gives a maximum value of 28 – 1 (i.e. 255). The value 0000 0000 is ignored in this calculation. Example 3 explains how a checksum is generated.

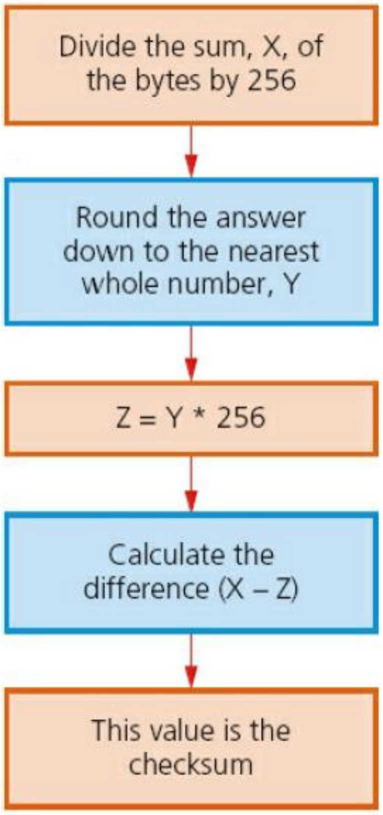
Example

If the sum of all the bytes in the transmitted block of data is <= 255, then the
38 checksum is this value.

However, if the sum of all the bytes in the data block > 255, then the checksum is found using the simple algorithm in

Suppose the value of X is 1185, then tracing through the algorithm, we get:
X = 1185

- 1185/256 = 4.629
- Rounding down to nearest whole number gives Y = 4
- Multiplying by 256 gives Z = Y * 256 = 1024
- The difference (X – Z) gives the checksum: (1185 – 1024) = 161
- This gives the checksum = 161


Divide the sum, X, of the bytes by 256
Round the answer down to the nearest whole number, Y
Z = Y * 256
Calculate the difference (X – Z)
This value is the checksum

When a block of data is about to be transmitted, the checksum for the bytes is first of all calculated. This value is then transmitted with the block of data. At the receiving end, the checksum is recalculated from the block of data received.

This calculated value is then compared to the checksum transmitted. If they are the same value, then the data was transmitted without any errors; if the values are different, then a request is sent for the data to be retransmitted.

CHECK DIGIT:

Number added to a code (such as a bar code or account number) to derive a further number as a means of verifying the accuracy or validity of the code as it is printed or transmitted. A code consisting of three digits, for example, such as 135 may include 9 (sum of 1, 3, and 5) as the last digit and be communicated as 1359.

Check digits can identify 3 types of error:
(1) If 2 digits have been inverted e.g. "23**45**9" instead of "23**54**9"
(2) An incorrect digit entered e.g. 23559 instead of 23549
(3) A digit missed out altogether e.g. 2359 instead of 23549

Now you will learn how check digits are calculated. The ISBN-10 (used on books) has been chosen as the example; this uses a module 11 system which includes the letter X to represent the number 10.

Example 1: To calculate the check digit for the ISBN"0 - 2 0 1 - 5 3 0 8 2 - ?

(i) the position of each digit is considered:

"10 9 8 7 6 5 4 3 2 1" ← digit position

0 - 2 0 1- 5 3 0 8 2 - ? ← Number

(ii) Each digit is then multiplied by its digit position and the totals added together

(0x10) + (2x9) + (0x8) + (1x7) + (5x6) + (3x5) + (0x4) + (8x3) + (2x2)
= 0 + 18 + 0 + 7 + 30 + 15 + 0 + 24 + 4
= 98

(iii) The total is then divided by 11 (modulo – 11) and the remainder, if any, is subtracted from 11 to give the check digit.
98 ÷ 11 = 8 remainder 10
11 – 10 = 1
This gives a check digit of "1"
Final ISBN becomes"0 -2 0 1 - 5 3 0 8 2 –1"

<u>Example 2</u>: To check the correctness of a check digit the computer re-calculates it as follows:
The ISBN to check is: 0 - 1 3 1 5 - 2 4 4 7 - X

(i) The position of each digit is considered:

10	9	8	7	6	5	4	3	2	1	← digit position
0	-1	3	1	-5	2	4	4	7	-X	← number

(ii) Each digit is then multiplied by its digit position and the totals added together
(0x10) + (1x9) + (3x8) + (1x7) + (5x6) + (2x5) + (4x4) + (4x3) + (7x2) + (Xx1)
= 0 + 9 + 24 + 7 + 30 + 10 + 16 + 12 + 14 + 10 (recall that X = 10)
= 132

(iii) The total is then divided by 11; if there is no remainder then the check digit is correct:
132 ÷ 11 = 12 remainder 0 Hence the check digit is correct

Echo check

With **ECHO CHECK**, when data is sent to another device, this data is sent back again to the sender. The sender compares the two sets of data to check if any errors occurred during the transmission process.

As you will have no doubt worked out, this isn't very reliable. If the two sets of data are different, it isn't known whether the error occurred when sending the data in the first place, or if the error occurred when sending the data back for checking!
However, if no errors occurred then it is another way to check that the data was transmitted correctly.