Interfacing Data Transfer Models

Table of Contents

[Peripherals I/O and Interfacing 2](#_Toc100240654)

[Models of Data Transfer 2](#_Toc100240655)

[Parallel Mode 2](#_Toc100240656)

[Serial Mode 3](#_Toc100240657)

[Synchronous and Asynchronous Transmission 4](#_Toc100240658)

[Conditions of Data Transfer 6](#_Toc100240659)

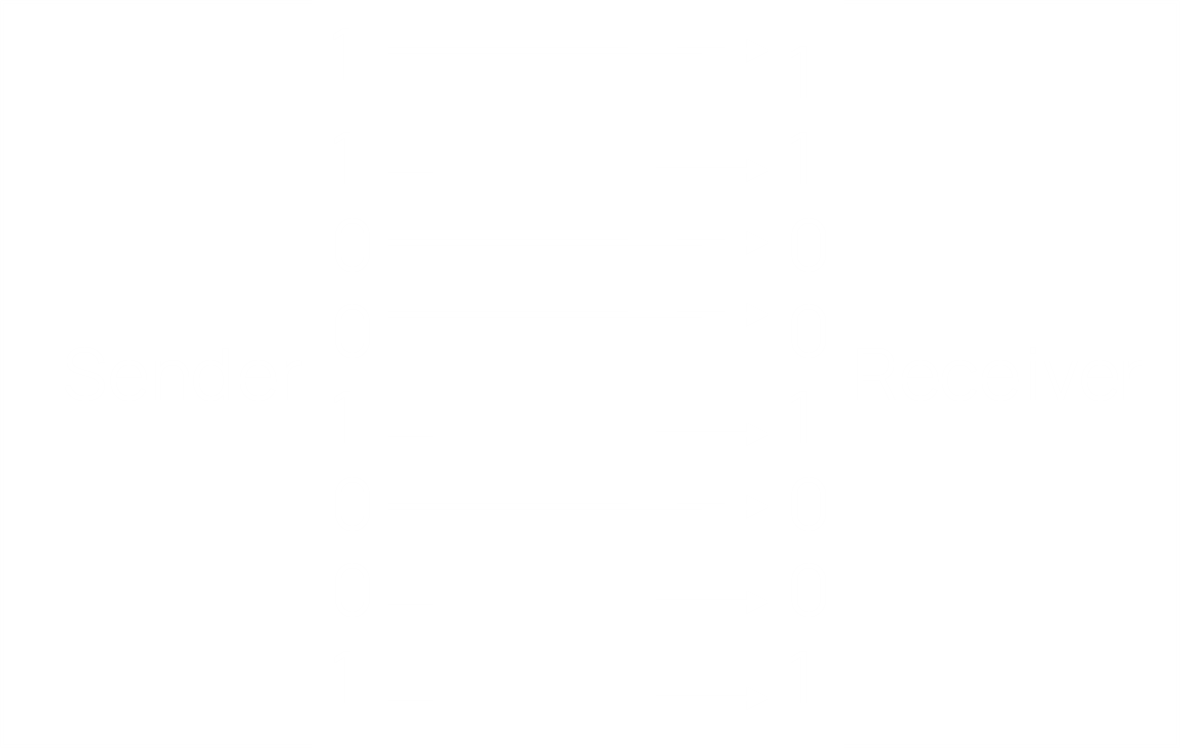
## Peripherals I/O and Interfacing

The primary functions of the microprocessor are to receive data from input devices, process the data using instructions from memory and send the results to output devices. The input and output devices are called **peripherals**. Designing logic circuits (hardware) and writing instructions (software) to enable the microprocessor to communicate with peripherals is called **interfacing**. The logic circuits themselves are called **I/O ports** or **interfacing devices**.

## Models of Data Transfer

Transmission of data occurs in one of two modes, parallel mode and serial mode.

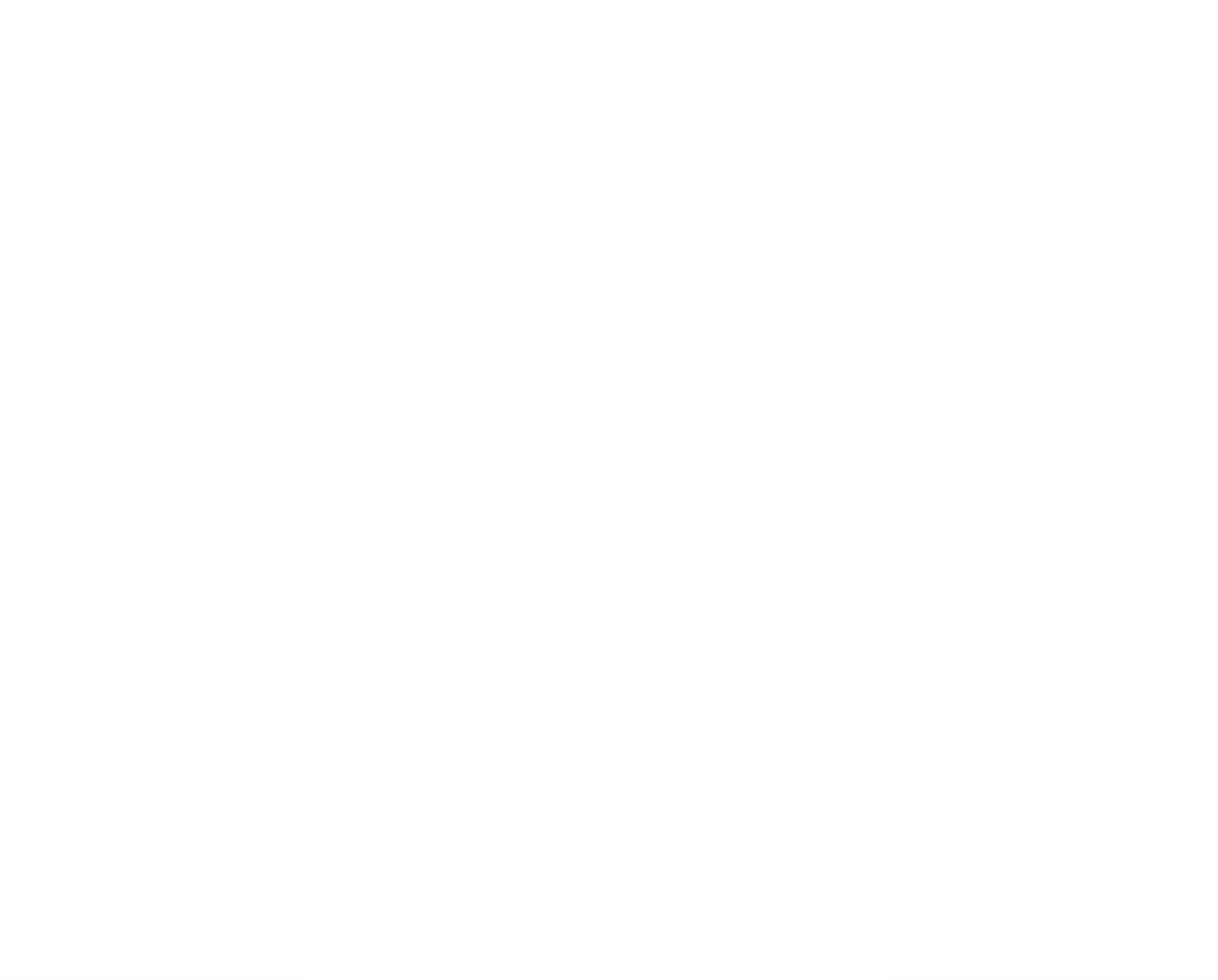
### Parallel Mode



In **Parallel Mode**, the entire data (i.e. all the bits) are transferred at one time. This requires **multiple data lines**. In the 8085 for example, it was possible to send 8 bits of data simultaneously, since the data bus had 8 lines.

* Faster
* Expensive
* Limited to small distances
* Needs separate data lines
* Bits must stay synchronized

### Serial Mode



In **Serial Mode**, there is a **single data line** and the bits of data are sent down the line one at a time. This complicates things since the machine needs to know how to **decompose** the bits at the sender’s end and how to **reconstruct** the bits at the receiver’s end.

* Cheaper
* Slower

### Synchronous and Asynchronous Transmission

If we are using **serial mode**, the transmission can be broken down into two categories, synchronous and asynchronous.

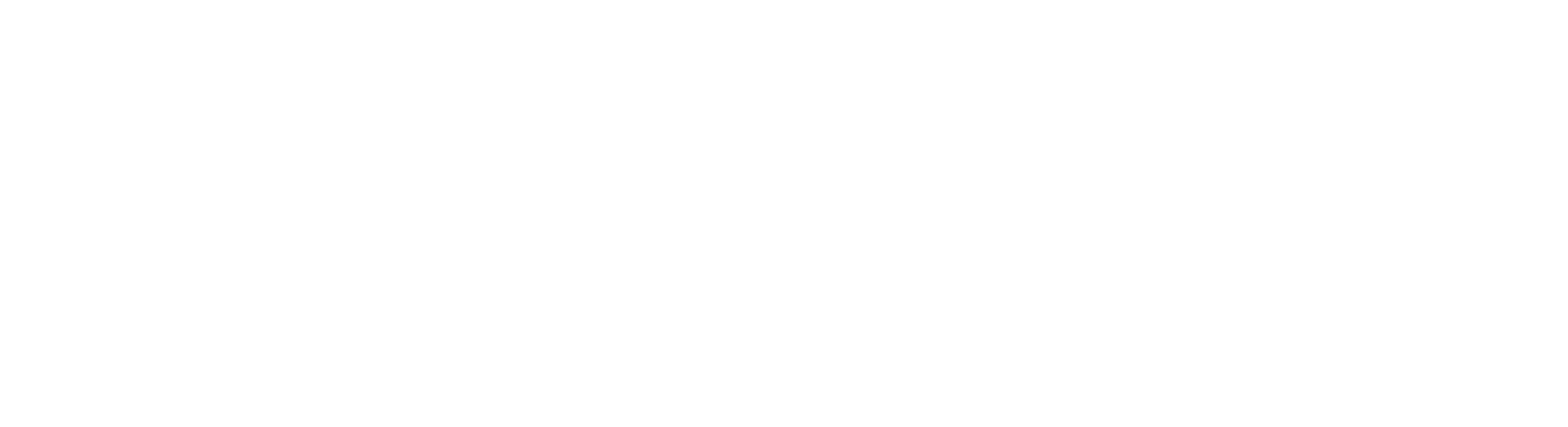
In **synchronous** transmission, the sender and receiver maintain the **same clock**. This allows us to send data in blocks with very **little overhead** and at a **high speed**.



Essentially, if each block of data has 1 byte and it takes 1 second to send it, the receiver knows that whatever data it receives in 1 second is going to consist of that 1 byte. We do not need to add any extra information to denote when a byte starts and ends or leave any transmission gaps.

To remain in sync, data is **continuously transmitted** with no stopping. The **data rate** depends on the clock rate.

By contrast, in **asynchronous** transmission, the sender and receiver clocks are **not synchronized**. This forces us to add at least 1 bit to the start and end to denote the block of data, which adds **overhead**, and also leave **transmission gaps**. The transmission gaps are also irregular, so time absolutely cannot be used to determine when a block starts and ends. All of this combines to give asynchronous transmission **low speed**. We may also add **parity bits** for error detection.



Consider that we have 250 characters, each of 8 bits, that we want to transfer. If we are using asynchronous transmission, each character must have a start and a stop bit, so there are 10 bits per character, times 250 characters, which means we must transfer a total of 2500 characters.

Instead, if we use synchronous transmission, we will only have to send characters. We will also need to send some synchronization bits from time to time. Suppose we send 1 synchronization character after every 50 characters. This brings our total up to characters. This is still 20% more efficient than the asynchronous counterpart.

Although it may seem like synchronous transmission is clearly better, we do need asynchronous transmission as well. Computers use synchronous transmission internally, due to the higher speed, but the microprocess must use asynchronous transmission to communicate with peripherals. This is because peripherals are much slower and waiting on them is inefficient.

### Conditions of Data Transfer

The data transfer process can be controlled by either the microprocessor or the peripheral device.

If the **microprocessor** is controlling the data transfer, it can take place in one of several ways:

* **Unconditional Data Transfer** – The microprocessor thinks that the peripheral device is always available.
* **Data Transfer with Polling** – Peripheral devices are polled regularly.
* **Data Transfer with Interrupt** – When a peripheral device is ready to transfer data, it sends an interrupt signal. The microprocessor pauses execution and sends the peripheral device an interrupt acknowledgement signal.
* **Data Transfer with READY Signal** – If the peripheral device has a slower response time than the microprocessor’s execution time, then states are extended to complete the data transfer.
* **Data Transfer with Handshake Signals** – Signals are exchanged between the microprocess and the peripheral devices before data transfer.

If the **peripheral device** is controlling the data transfer, then it can take place in only one way, **Direct Memory Access** (DMA). This is much faster. The DMA controller sends a HOLD signal to the microprocessor, which releases its data bus and address bus to the controller. Data can then be transferred at high speeds without further intervention from the microprocessor.