I2C Bus

Table of Contents

[Data Transfer 2](#_Toc106753025)

[Addressing ICs 3](#_Toc106753026)

[Special Addresses 4](#_Toc106753027)

[Acknowledgements 4](#_Toc106753028)

[Data Formats 6](#_Toc106753029)

[Multi-Master Systems 7](#_Toc106753030)

[Error Checking 7](#_Toc106753031)

[Bus Recovery 8](#_Toc106753032)

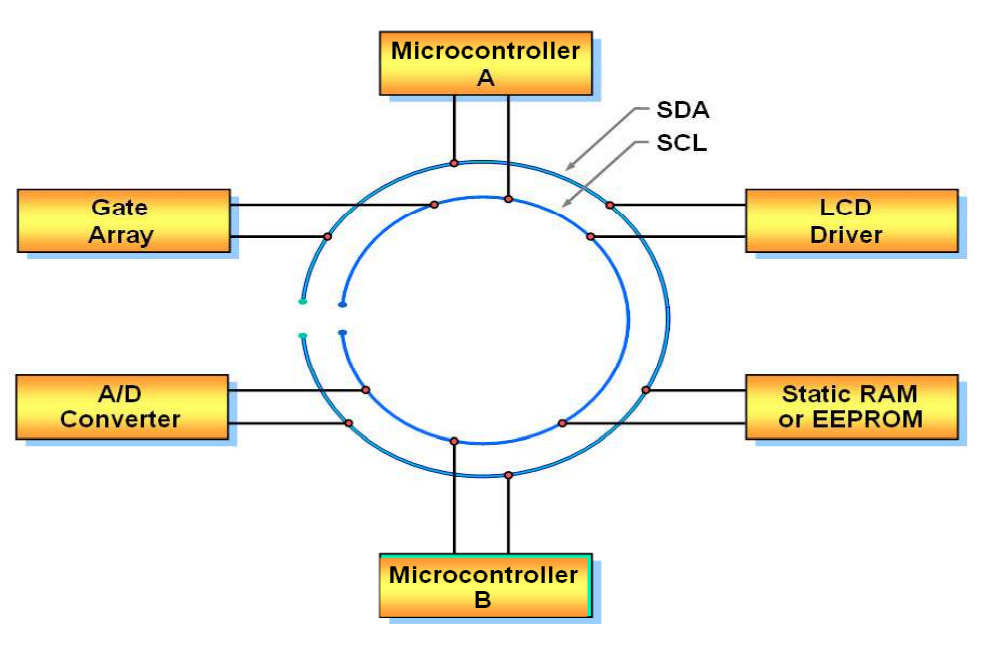
[Available I2C Devices 8](#_Toc106753033)

[End Uses 9](#_Toc106753034)

[Designer Benefits 9](#_Toc106753035)

[Manufacturer Benefits 10](#_Toc106753036)

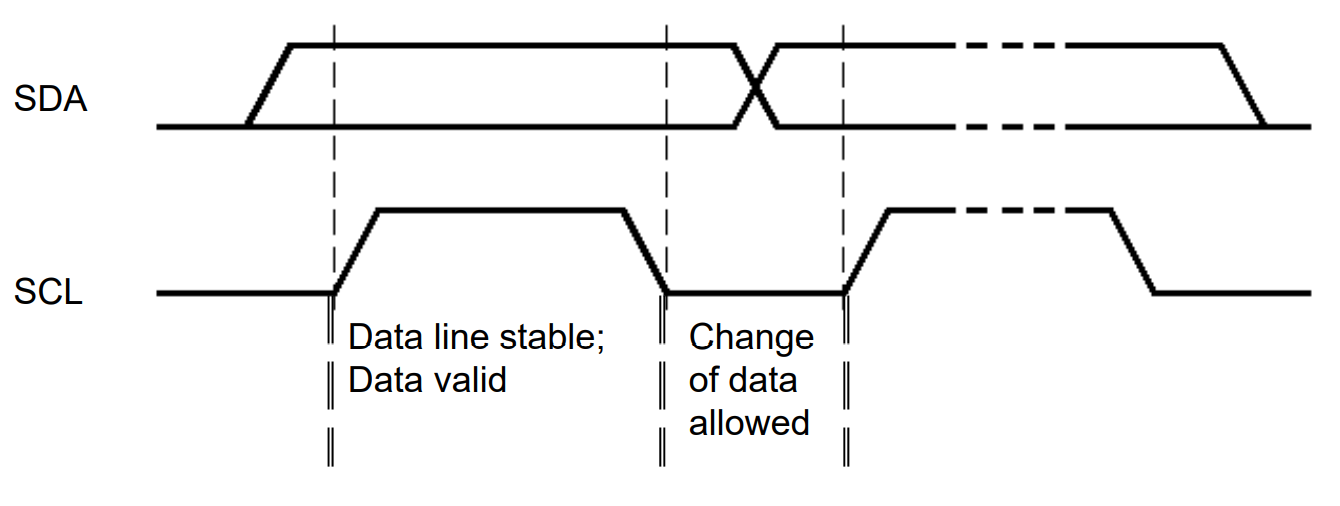
The **Inter-Integrated Circuit Bus** (I2C Bus) is used by **small-scale embedded systems**, as opposed to the CAN bus which was used by large-scale embedded systems. It has two wires, with the **outer wire** carrying serial data (SDA) and the **inner wire** carrying serial clock (SCL) signals for synchronization.



I2C busses have three modes, a classic mode, a fast mode and a high-speed mode.

## Data Transfer

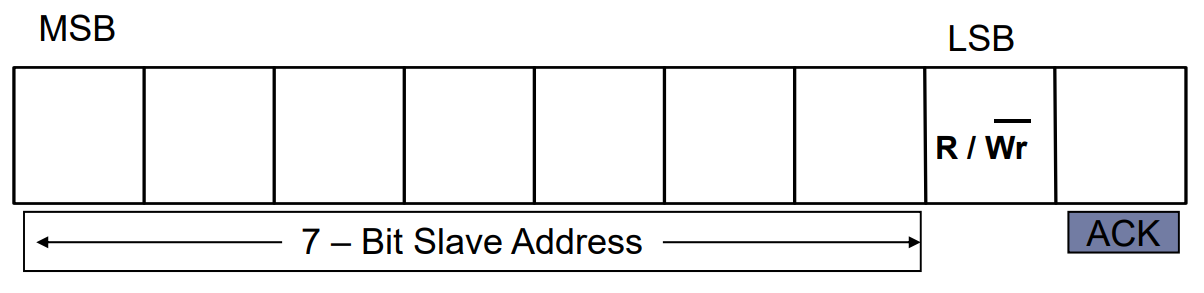
The SCL line simply has alternative high and low states, since it is a clock. The SDA line holds data. As a rule, the SDA line is only allowed to **transition** during the low state of the SCL line.



If the SDA line transitions during the high state of the SCL line, it is treated as a **start** or a **stop** signal for data transfer. The start signal is indicated by a transition from high to low, while the stop signal is indicated by a transition from low to high. During the time between the start and stop signals, the bus is considered to be busy. These two signals can only ever come from the coordinator.

## Addressing ICs

I2C busses use a **master-slave relationship**, meaning there is a **microcontroller** which controls all the other devices connected to the bus. **7-bit addresses** are used to identify the devices, although the specifications allow an extension up to 10-bit addresses.



The first byte in the data transfer process consists of the 8 bits shown above. The first 7 bits are the address of the device while the 8th bit indicates whether the device should read the data being sent by the microcontroller or write data onto the bus to be read by the microcontroller. Notice that the read/write byte is an instruction to the IC about what to do.

Once the appropriate device has received this byte, it sends an **ACK** signal.

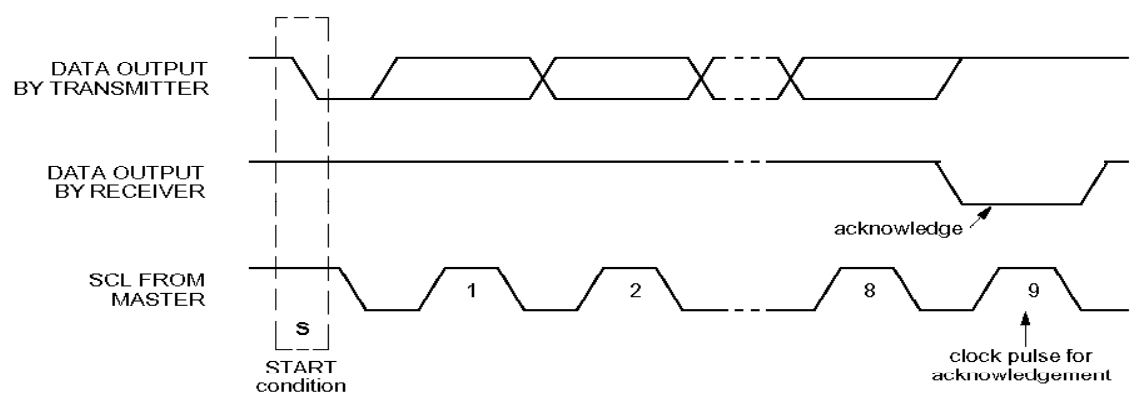
## Special Addresses

Apart from the addressing scheme discussed above, there are also some special addresses that can be used:

* 0000000 – General Call Address
* 0000001 – Null Address
* 1111XXX – Address Extension
* 1111111 – Address Extension; Next bytes are the actual address

## Acknowledgements

Since the data consist of a total of **8 bytes**, it requires **8 clock pulses** to transfer. At the **9th clock pulse**, the receiver sends a **low signal** on the **SDA**. This works as an **ACK** signal.

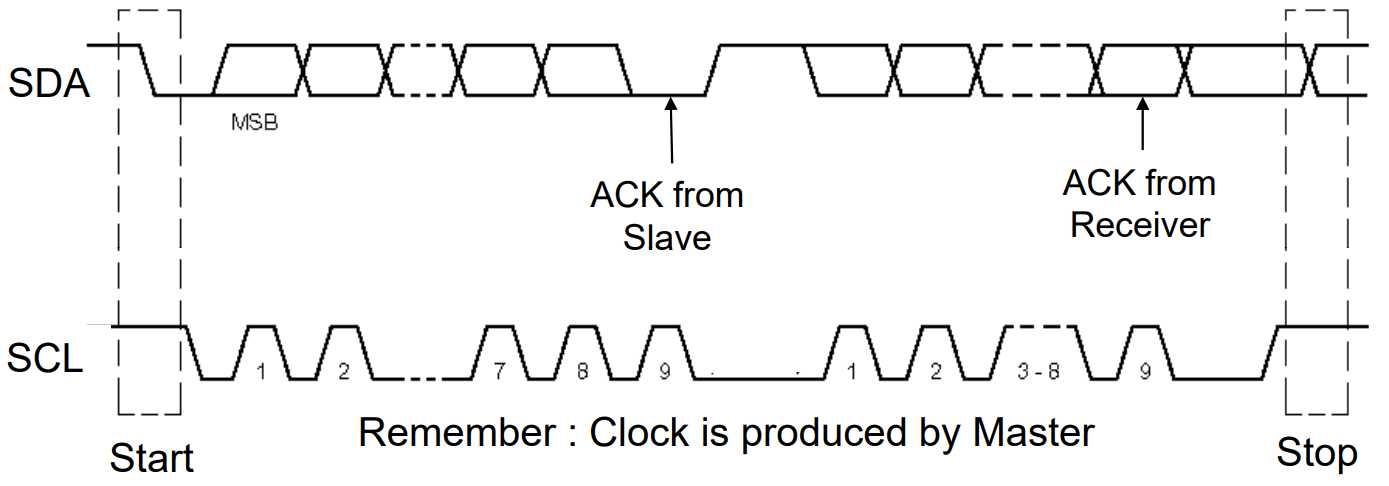


The receiver may also send a **negative acknowledgement** (NAK) by holding the SDA line **high** at the 9th clock pulse. An NAK could be issued due to:

* Incorrectly received address
* Incorrectly received data byte
* Device not being connected to the bus

Notice that, by default, the SDA line is high, so unless a device explicitly makes it low to issue an ACK, an NAK is assumed. This is why a device not being connected to the bus at all can still cause an NAK.

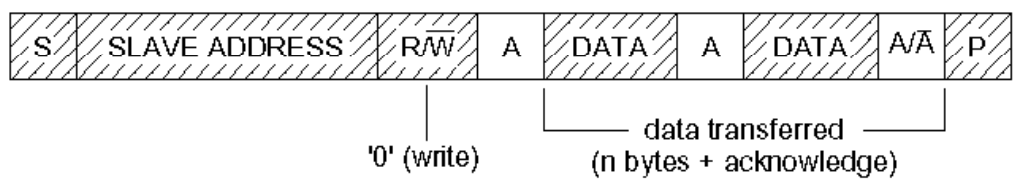
Thus, the overall process for data transfer is: Start → Address + R/W → Data Bytes → Stop.



## Data Formats

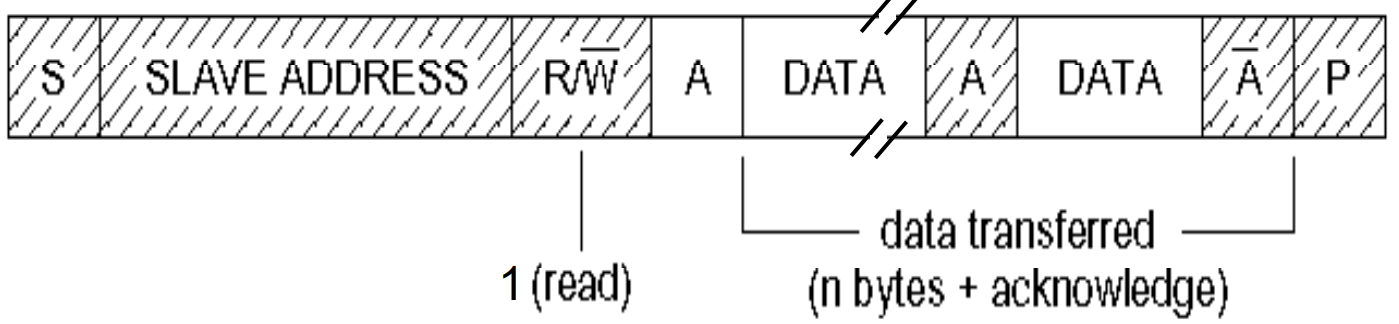
Based on the above discussion, we can create some data formats for the data transfer process. There are only two cases, the microcontroller writing to a device or the microcontroller reading from a device.

For the first case, we have the following data format:

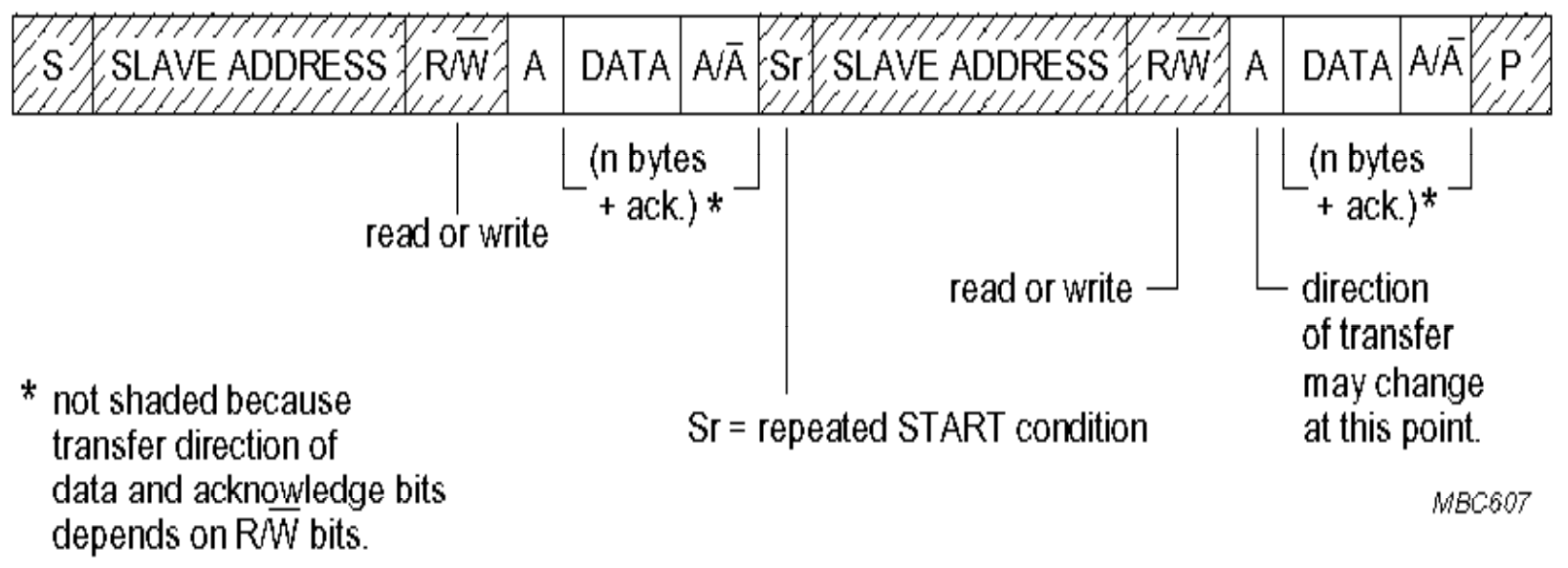


Here, indicates a start condition, indicates an ACK, indicates an NAK and indicates a stop condition. The shaded boxes are for data being placed on the SDA by the microcontroller, while the clear boxes are for data being placed on the SDA by a device.

Following a similar formatting, we can have the data format for the second case:



We can also have a **combined format**, with the direction of transfer varying. In this case, the controller does not release the bus at all. This requires a special Sr signal.



The need to not release the bus indicates the presence of a competing entity which can take over the bus if it is released. This is where **multiple controllers** come in. Contentions between controllers are resolved using the **arbitration** method.

## Multi-Master Systems

It is possible to have **multiple controllers** in a single I2C bus. In that case, contentions are resolved by using an **arbitration** technique.

## Error Checking

Error checking is performed in the I2C bus using a **checksum**.

At the sender side, an 8-bit value is available. The **1’s complement** of this value is the checksum value. At the receiver’s end, the 8-bit value and the checksum value are added, and the 1’s complement of the result is checked. If there are no errors, this value should be 0.

The CAN bus used the **CRC mechanism** to check for errors.

## Bus Recovery

There are several scenarios in which the I2C bus can get **locked**:

* The controller and the devices become out of sync
* A stop signal is omitted or missed (possibly due to noise)
* Any device holds one of the lines low improperly
* The bus line gets shorted

If the bus is locked, the controller can fix it. If the SCL is still usable, the controller can keep sending extra clock signals until the SDA becomes high and then send a stop signal. If the SCL is stuck in the low state, only the device using it can fix the problem.

## Available I2C Devices

* AD and DA Convertors
* Bus Controllers
* Bus Repeaters, Hubs and Expanders
* Real-Time Clocks and Calendars
* DIP Switches
* LCD and LED Display Drivers
* General Purpose I/O Expanders and LED Display Controls
* Multiplexers and Switches
* Serial RAM and EEPROM
* Temperature and Voltage Monitors
* Voltage Level Translates

## End Uses

* Telecom devices (base stations, mobile phones, etc.)
* Data processing devices (laptops, servers, etc.)
* Instrumentation
* Automotive devices (dashboards, etc.)
* Consumer devices (audio systems, TVs, etc.)

## Designer Benefits

* Blocks in block diagram correspond to actual ICs
* No need to design bus interfaces
* Integrated addressing and data-transfer protocols allow systems to be completely software defined
* ICs using the bus can be used with multiple different applications
* Blocks in ICs using the bus are frequently encountered and become familiar
* ICs can be added and removed without affecting other circuitry
* Fault diagnosis and debugging is easy
* Software modules can be reused
* Fewer interconnections lead to smaller and cheaper PCBs

## Manufacturer Benefits

* No need to address decoders
* Rapid testing and alignment of end-user equipment via external connections to an assembly line
* Increased design flexibility due to simple construction of equipment variants and easy upgrading
* De-facto world standard