I2C (pronounced I squared C meaning Inter IC) Bus protocol
by NXP semiconductors (formerly Philips Semiconductors)
Short introduction by Gandhi Puvvada

There is more info about Special addressing, additional modes of operation. One needs to consult the documentation for complete info.
I2C references:

Philips Semiconductors is now NXP Semiconductors

A tutorial:
http://tmd.havit.cz/Papers/I2C.pdf

Latest

Old version
http://i2c2p.twibright.com/spec/i2c.pdf

Other references:

Wikipedia

best-microcontroller-projects.com

Robot Electronics
http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html

Engineers Garage
http://www.engineersgarage.com/tutorials/twi-i2c-interface

I used extracts of these in this lecture for educational purposes.
Serial Bus

1-bit per clock

I2C, UART, SPI, USB

Quick start

Standard mode
100 kHz → 100 kbit/s
Later versions
400 kHz, 1 MHz,
3.4 MHz, 5 MHz.

much faster
I2C  generic name is

TWI
(TWO WIRE INTERFACE)

uC Master  ADC Slave  DAC Slave  uC Slave

SDA  SCL
Fig 1: Block diagram

Fig 2. Example of an I²C-bus configuration using two microcontrollers

http://www.engineersgarage.com/tutorials/twi-i2c-interface
Parallel communication bandwidth is much higher, but cost is also higher.
A lot of low-speed devices such as LCD panels, temperature sensors, etc. do not need high-bandwidth.

Such slaves can be connected using **I2C** using just two wires (besides the VCC and GND) to interface to a single master or a group of masters.

It is quite pedagogical to understand how they managed to incorporate handshaking with slow devices and arbitration with other masters using just two wires!
Initially, let us assume that there is a common clock, SCL, driven by a SINGLE master.

Later we will discuss how the multiple masters together generate SCL and arbitrate over SDA.
Master: Sends address, sets direction
7-bit R/W

Slave: Responds

Master → address → Slave

Transmitter → DATA → Receiver

Write Master Slave
Read Slave Master
I2C basic command sequence

1. Send the START (S).
2. Send the slave address (ADDR).
3. Send the \( R / \overline{W} \) Read(R) 1 / Write(W) 0.
4. Wait for/Send an active-low acknowledge (A).
5. Send/Receive the data byte (8 bits) (DATA).
6. Expect/Send an active-low acknowledge (A).
7. Send the STOP (P).

\[ \text{ACK} = 0 \]
\[ \text{NACK} = 1 \] Acknowledge (ACK) and Not Acknowledge (NACK)
Burst Transfer: Data after data from/to the slave identified by the 7-bit slave address

<table>
<thead>
<tr>
<th>START condition</th>
<th>7 bit slave address</th>
<th>Data direction bit</th>
<th>Slave ACK</th>
<th>8 bit data transmission</th>
<th>Receiver ACK</th>
<th>STOP condition</th>
</tr>
</thead>
</table>
SDA is valid and stable during SCL clock high part when carrying an address bit or a data bit or R/W or ACK. If SDA changes during clock high part, it is a START condition or a STOP condition.
The master initiates the communication by sending a *Start condition* on the SDA and SCL line. A high to low transmission on SDA line while SCL is high is defined as a *Start condition.*

http://www.engineersgarage.com/tutorials/twi-i2c-interface
When data transmission is completed the **Stop Condition** is issued by master to stop the communication. A low to high transmission on SDA line while SCL is high is defined as a **Stop condition**.
SDA is driven by either the MASTER or the SLAVE.

Possible collision?

MSB of the next byte from Slave (in a Read transaction) and

STOP condition from the master?
NACK by master
= Stop reading

Master should drive a NACK (opposite of ACK) to warn slave not to start the next byte.
Material Supporting The previous slide

Fig 11. A master-transmitter addressing a slave receiver with a 7-bit address (the transfer direction is not changed)

Fig 12. A master reads a slave immediately after the first byte

Fig 13. Combined format
NACK

NACK by master

= Stop reading

Master tells slave to read, and the slave reads out byte after byte. After the 2nd byte, master wants to say STOP.

A master reads a slave immediately after the first byte.

So slave will not drive MSB here.

:: No collision
Bus Free and Bus Busy

Bus Busy  Bus Free  Bus Busy

S  P  S  P

Start Condition  Stop Condition  Start Condition  Stop Condition
review of open-collector type (open-drain type)

By analogy, we remove the source transistors in the totem-pole output stage of the NAND gates. The role of the external blower is held by the external pull-up resistance (usually 10Kohms) here. If

Such gates without source transistor are called "open-collector output gates" because the collector leg of the sink transistor is left "open" in the output-stage.

BiPolar

Drain

MOS

Gate

Source

WIRE-ANDing

VCC = 5V

10KΩ

Imaginary AND gate formed by the wire-interconnection
A clever and interesting design to perform

handshake between a slow slave and a fast master

and

arbitration between multiple masters

using open-collector type (open-drain type)

SCL and SDA
normal handshake between separately
clocked devices

The 4-way handshake

P1: Take it
   C1: Got it
P2: I see that you got it
   C2: I see that you saw that I got it
The 2-way handshake is better than the 4-way.

The 4-way handshake

P1: Take it
C1: Got it
P2: I see that you got it
C2: I see that you saw that I got it

The 2-way handshake

P1: Take it
C1: Got it
P2: Take next
C2: got next
In I2C, either the master or the slave or both can take extra time by **stretching** the clock **SCL**.

**SCL** is driven by the current master using a open-collector (open-drain) gate. So he can only force a zero (0), can not force a one (1). Slave can stretch the zero as long as he needs.

**SCL** is not a free-running clock. It is derived by counting another higher-frequency clock.

High-period count and Low-period count.
SCL  Stretching SCL = hold SCL low

Fig 6. Data transfer on the I^2C-bus
normal arbitration between masters using a bus arbiter
Fixed Priority Resolvers  |  Rotating priority Resolvers

High-active Signals

Low-active Signals

ID of the device, who received service most recently
Arbitration between I2C masters using contention on SDA.

But first we discuss how they all together generate SCL.

We said before that the current master drives SCL. But how to decide the current master first? SCL is not a free-running clock. It is derived by counting another higher-frequency clock.

High-period count and Low-period count.
During Bus Free period, one or more masters may start generating clock SCL and create START condition together.

High-period count and Low-period count

Synchronization on SCL

• frame started $\rightarrow$ SCL = 1
  • first 1$\rightarrow$0 transition
    – involved masters restart their clocks
    – master holds 0 until its low-period is over
  • master finished its low-period
    – releases SCL
    – SCL = 0 $\rightarrow$ switches to wait state, waits for SCL = 1
    – SCL = 1 or waiting finished $\rightarrow$ starts counting high-period

• first master finished its high-period
  – sets SCL = 0
  – equivalent to (*) state

If High-period count is equal to Low-period count, and if individual internal frequencies are slightly different, then the SCL low-period is slightly (longer / shorter) than the SCL high-period.
Synchronized Clocks

- low-period
  - determined by max\{low-perIODs of involved masters\}
- high-period
  - determined by min\{high-perIODs of involved masters\}
Arbitration on SDA

- frame started, SCL synchronized → high periods = valid data
- each master generates its data
- master aborts if there is another level on SDA than it generates
  → it loses arbitration, releases SDA and tries again when bus is free

![Arbitration Diagram]

Notes on Arbitration

- can continue for many bits
- adresses are compared at first stage
- if the same slave is addressed in the same mode (R/W):
  - masters are transmitters → data-bits are compared
  - masters are receivers → acknowledge-bits are compared
- if arbitration is not over before stop or repeated start:
  - comparation not allowed: stop-data, repeated start-data, stop-repeated start
  - involved masters must generate stops/repeated starts in the same positions
- loser can generate SCL pulses for synchronization until the byte is over
- losing master which can be also a slave
  - must switch immediately to slave mode