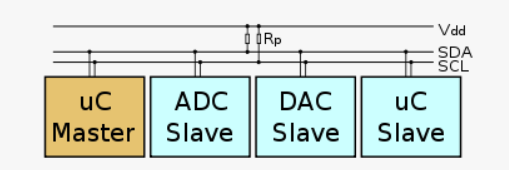
I2C

I2C is a serial protocol for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems. It was invented by Philips and now it is used by almost all major IC manufacturers. Each I2C slave device needs an address – they must still be obtained from [NXP](http://www.nxp.com/campaigns/i2c-bus/) (formerly Philips semiconductors).



I2C bus is popular because it is simple to use, there can be more than one master, only upper bus speed is defined and only two wires with pull-up resistors are needed to connect almost unlimited number of I2C devices. I2C can use even slower microcontrollers with general-purpose I/O pins since they only need to generate correct [Start and Stop conditions](https://i2c.info/i2c-bus-specification#start-stop) in addition to functions for reading and writing a byte.

Each slave device has a unique address. Transfer from and to master device is serial and it is split into 8-bit packets. All these simple requirements make it very simple to implement I2C interface even with cheap microcontrollers that have no special I2C hardware controller. You only need 2 free I/O pins and few simple i2C routines to send and receive commands.

The initial I2C specifications defined maximum clock frequency of 100 kHz. This was later increased to 400 kHz as [Fast mode](https://i2c.info/i2c-bus-specification#fast-mode). There is also a High speed mode which can go up to 3.4 MHz and there is also a 5 MHz ultra-fast mode.

I2C Interface

I2C uses only two wires: SCL (serial clock) and SDA (serial data). Both need to be pulled up with a resistor to +Vdd. There are also I2C level shifters which can be used to connect to two I2C buses with different voltages.

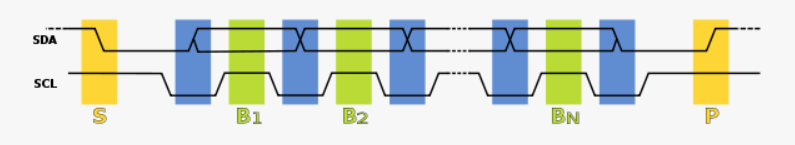
I2C Addresses

Basic [I2C communication](https://i2c.info/i2c-bus-specification#data-transfer) is using transfers of 8 bits or bytes. Each I2C slave device has a [7-bit address](https://i2c.info/i2c-bus-specification#7-bit-addressing) that needs to be unique on the bus. Some devices have fixed I2C address while others have few address lines which determine lower bits of the I2C address. This makes it very easy to have all I2C devices on the bus with unique I2C address. There are also devices which have [10-bit address](https://i2c.info/i2c-bus-specification#10-bit-addressing) as allowed by the specification.

7-bit address represents bits 7 to 1 while bit 0 is used to signal reading from or writing to the device. If bit 0 (in the address byte) is set to 1 then the master device will read from the slave I2C device.

Master device needs no address since it generates the clock (via SCL) and addresses individual I2C slave devices.

I2C Protocol



In normal state both lines (SCL and SDA) are high. The communication is initiated by the master device. It generates the [Start condition](https://i2c.info/i2c-bus-specification#start-stop) (S) followed by the address of the slave device (B1). If the bit 0 of the address byte was set to 0 the master device will write to the slave device (B2). Otherwise, the next byte will be read from the slave device. Once all bytes are read or written (Bn) the master device generates [Stop condition](https://i2c.info/i2c-bus-specification#start-stop) (P). This signals to other devices on the bus that the communication has ended, and another device may use the bus.

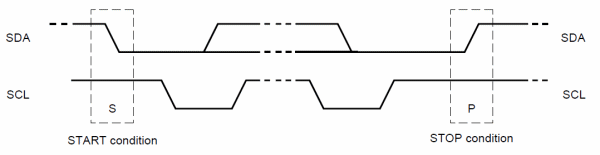
Most I2C devices support repeated start condition. This means that before the communication ends with a stop condition, master device can repeat start condition with address byte and change the mode from writing to reading.

## Conclusion

I2C bus is used by many integrated circuits and is simple to implement. Any microcontroller can communicate with I2C devices even if it has no special I2C interface. [I2C specifications](https://i2c.info/i2c-bus-specification) are flexible – I2C bus can communicate with slow devices and can also use high speed modes to transfer large amounts of data. Because of many advantages, I2C bus will remain as one of the most popular serial interfaces to connect integrated circuits on the board.

## Start and Stop Condition

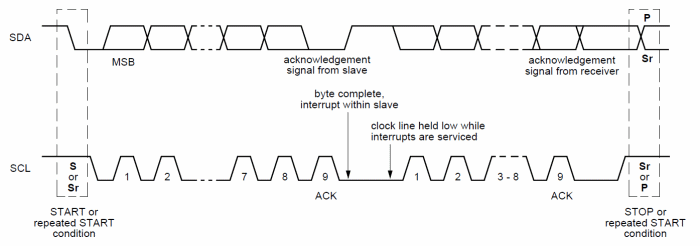
Each I2C command initiated by master device starts with a **START condition** and ends with a **STOP condition**. For both conditions SCL has to be high. A high to low transition of SDA is considered as **START** and a low to high transition as **STOP**.



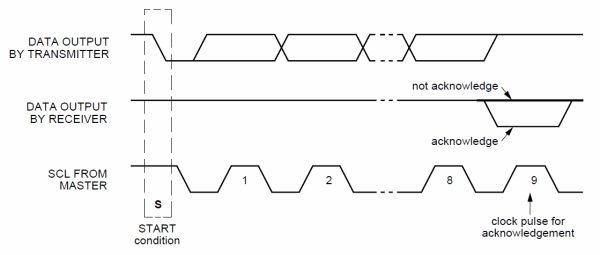
After the Start condition the bus is considered as busy and can be used by another master only after a Stop condition is detected. After the Start condition the master can generate a repeated Start. This is equivalent to a normal Start and is usually followed by the slave I2C address.

Microcontrollers that have dedicated I2C hardware can easily detect bus changes and behave also as I2C slave devices. However, if the I2C communication is implemented in software, the bus signals must be sampled at least two times per clock cycle in order to detect necessary changes.

## I2C Data Transfer



Data on the I2C bus is transferred in 8-bit packets (bytes). There is no limitation on the number of bytes, however, each byte must be followed by an Acknowledge bit. This bit signals whether the device is ready to proceed with the next byte. For all data bits including the Acknowledge bit, the master must generate clock pulses. If the slave device does not acknowledge transfer this means that there is no more data or the device is not ready for the transfer yet. The master device must either generate Stop or Repeated Start condition.



Each master must generate its own clock signal and the data can change only when the clock is low. For successful bus arbitration a synchronized clock is needed. Once a master pulls the clock low it stays low until all masters put the clock into high state. Similarly, the clock is in the high state until the first master pulls it low. This way by observing the SCL signal, master devices can synchronize their clocks.

### ARBITRATION

For normal data transfer on the I2C bus only one master can be active. If for some reason two masters initiate I2C command at the same time, the arbitration procedure determines which master wins and can continue with the command. Arbitration is performed on the SDA signal while the SCL signal is high. Each master checks if the SDA signal on the bus corresponds to the generated SDA signal. If the SDA signal on the bus is low but it should be high, then this master has lost arbitration. Master I2C device that has lost arbitration can generate SCL pulses until the byte ends and must then release the bus and go into slave mode. The arbitration procedure can continue until all the data is transferred. This means that in multi-master system each I2C master must monitor the I2C bus for collisions and act accordingly.

### CLOCK SYNCHRONIZATION AND HANDSHAKING

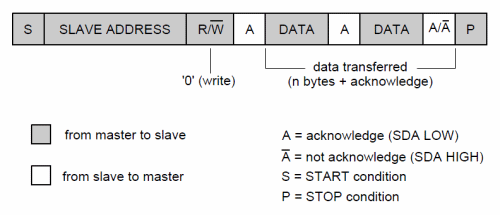
Slave devices that need some time to process received byte or are not ready yet to send the next byte, can pull the clock low to signal to the master that it should wait. Once the clock is released the master can proceed with the next byte.

### COMMUNICATION WITH 7-BIT I2C ADDRESSES

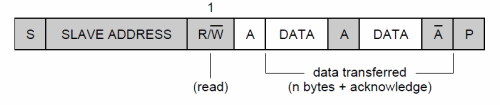


Each slave device on the bus should have a unique 7-bit address. The communication starts with the Start condition, followed by the 7-bit slave address and the data direction bit. If this bit is 0 then the master will write to the slave device. Otherwise, if the data direction bit is 1, the master will read from slave device. After the slave address and the data direction is sent, the master can continue with reading or writing. The communication is ended with the Stop condition which also signals that the I2C bus is free. If the master needs to communicate with other slaves it can generate a repeated start with another slave address without generation Stop condition. All the bytes are transferred with the MSB bit shifted first.

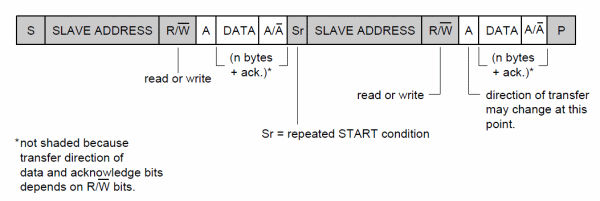
If the master only writes to the slave device then the data transfer direction is not changed.



If the master only needs to read from the slave device then it simply sends the I2C address with the R/W bit set to read. After this the master device starts reading the data.

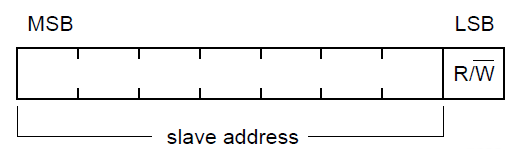


Sometimes the master needs to write some data and then read from the slave device. In such cases it must first write to the slave device, change the data transfer direction and then read the device. This means sending the I2C address with the R/W bit set to write and then sending some additional data like register address. After writing is finished the master device generates repeated start condition and sends the I2C address with the R/W bit set to read. After this the data transfer direction is changed and the master device starts reading the data.

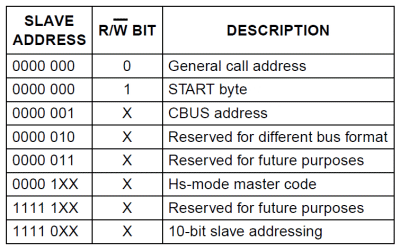


## 7-bit I2C Addressing

A slave address may contain a fixed and a programmable part. Some slave devices have few bits of the I2C address dependent on the level of address pins. This way it is possible to have on the same I2C bus more than one I2C device with the same fixed part of I2C address.



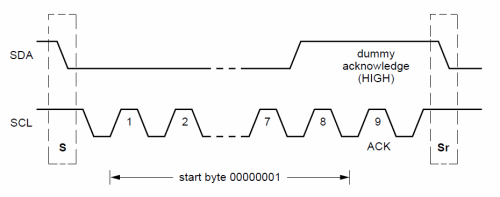
The allocation of I2C addresses is administered by the I2C bus committee which takes care for the allocations. Two groups of 8 I2C addresses are reserved for future uses and one address is used for 10-bit I2C addressing.



The general call address is used to address all devices on the slave bus. If any slave device doesn’t need to respond to such call or general call is not supported by the slave device, the call must be ignored. If the device supports general call and wants to receive the data it must acknowledge the address and read the data as a slave receiver.

### START BYTE

If microcontroller has I2C hardware and the microcontroller acts as a slave then the software needs to do nothing to check the bus state. The I2C hardware will detect Start condition, receive the I2C address and interrupt the software if necessary. However, if the I2C interface is implemented by the software, the microcontroller has to sample SDA line at least twice per clock pulse in order to detect changes. To simplify detection of I2C commands on the bus in such cases, a special I2C address called Start byte is used. Such start byte (0000 0001) is followed by an acknowledge pulse (for interface compatibility reasons). This combination holds the SDA line low for 7 clock pulses and allows simple detection of active I2C bus with lower sampling frequency.



## Extension of the I2C Specifications

Standard mode of I2C bus uses transfer rates up to 100 kbit/s and 7-bit addressing. Such I2C interface is used by many hundred [I2C-compatible devices](http://ics.nxp.com/interface/#I2C) from many manufacturers since its introduction in the 80s. However, with the advance of the technology, needs for higher transfer rates and larger address space emerged. There are cases where large amount of data needs to be transferred. Many complex embedded boards contain a large number of different I2C devices. In some cases it is very hard to avoid address collisions since 7 bits for I2C addresses allow only 127 different addresses where only 112 can actually be used. Some I2C devices on the board, despite address pins, have the same address. This resulted in few upgrades to the standard-mode I2C specifications:

* Fast Mode – supports transfer rates up to 400 kbit/s
* High-speed mode (Hs-mode) – supports transfer rates up to 3.4 Mbit/s
* 10-bit addressing – supports up to 1024 I2C addresses

There can by any combination of the devices on the bus regardless of the supported speed and addressing. Fast mode devices are downward-compatible and can work with slower I2C controllers. However, most modern I2C controllers support all speeds and addressing modes.

High-speed mode uses signals called SCLH and SDAH to emphasize the higher speed. These signals are usually separated from standard SDA and SCL lines. High-speed mode introduces also few differences (or improvements) in the specifications:

* Improved data and clock line output drivers
* Schmitt trigger and spike suppression circuits on data and clock inputs
* Clock synchronization and arbitration is not used
* Clock signal has 1 to 2 high/low ratio

From http://i2c.info