I2C Basics

Yogesh M Iggalore

What is I2C?

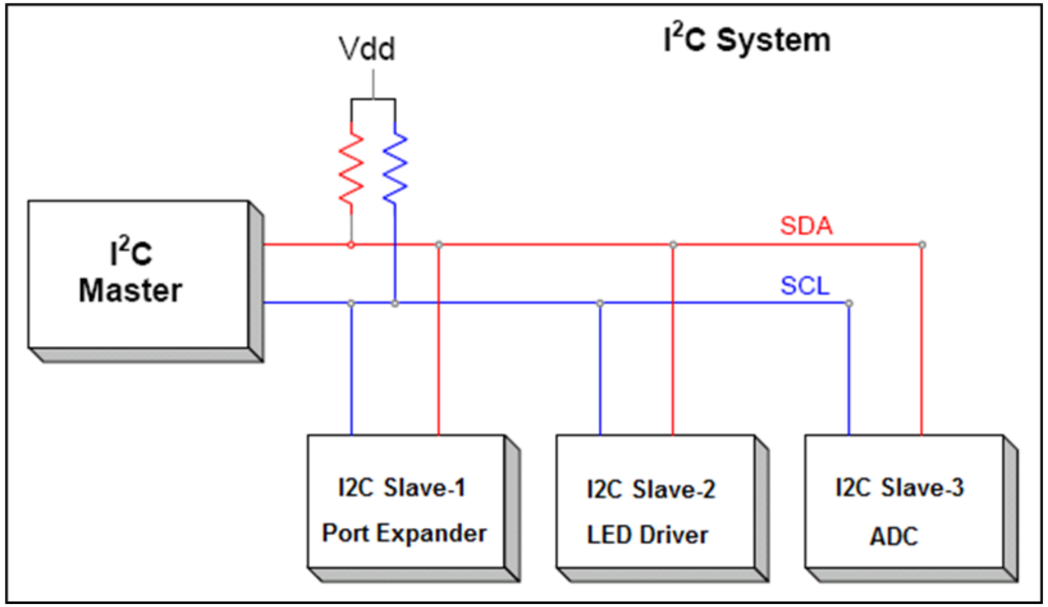
Inter-Integrated Circuit (IIC or I2C) is a common chip-to-chip digital communications protocol.

I2C was originally designed by Phillips Semiconductor (now NXP) in the early '80s

It's a simple two wire synchronous protocol

It supports multiple slaves on the same bus

It also supports multiple masters on the same bus



Why do we need I2C?

Applications : Wide usage

- Simple to use and quick to implement into designs
- Intended to control, check & update the status and do maintenance functions
- Enhance feature set of applications
- Standard adopted by all Industry segments and used in many system applications

Interface : Well known bus interface standard

- Bidirectional transfer of data between a master and several slave devices on the bus
- The master device controls the bus
- Each slave device on the same bus has an unique I2C address
- More than 25 years of existence

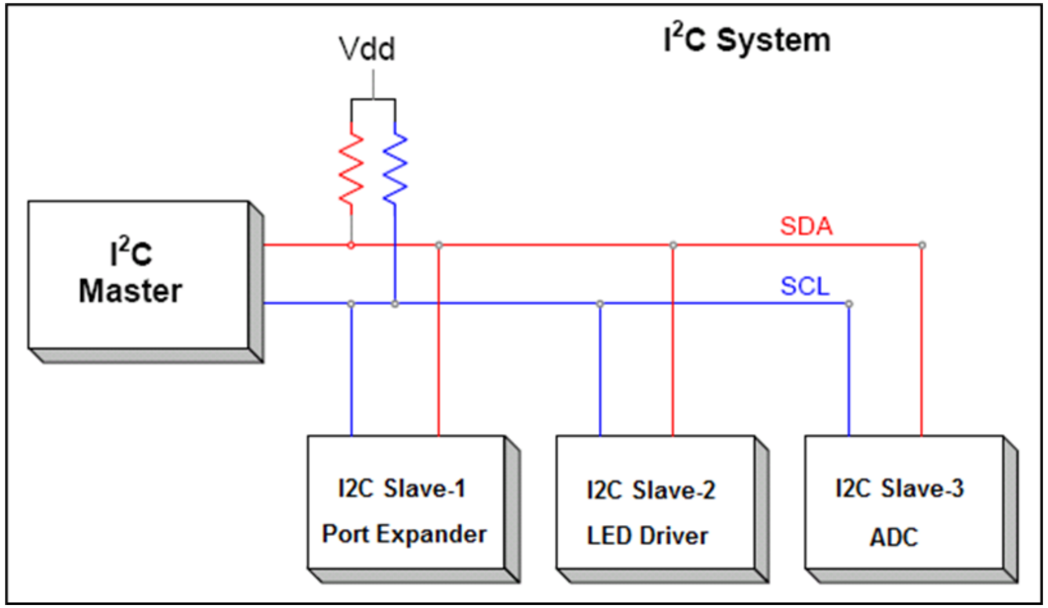
Why do we need I2C?

Architecture :

- Two-wire communication bus

Speed: Three modes of operation

- Standard mode (0 to 100 KHz)
- Fast mode (0 to 400 KHz)
- High-speed mode (0 to 3.4 MHz)

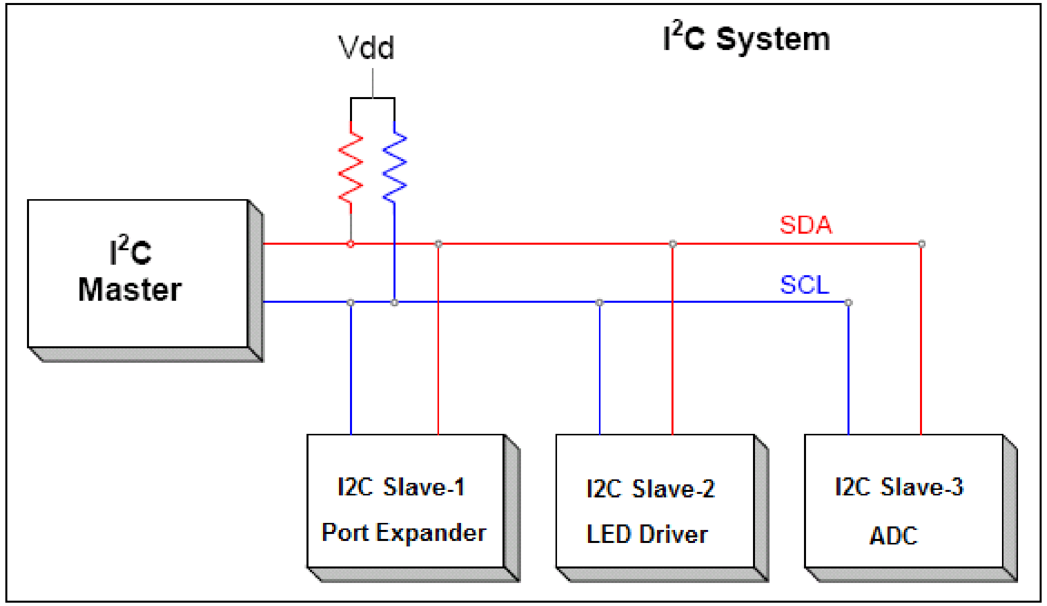


I2C - Master and slave

A master device is in charge of the bus and this device controls the clock and generates START and STOP signals.

Slaves simply listen to the bus and act on controls and data that they are sent.

The master can send data to a slave or receive data from a slave - slaves do not transfer data between themselves.

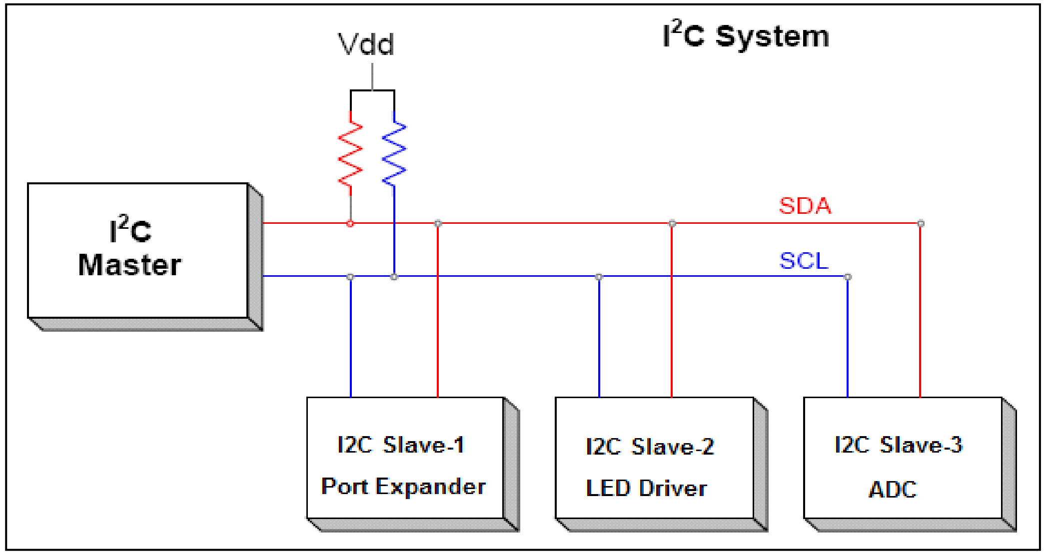


I2C - Data and Clock lines

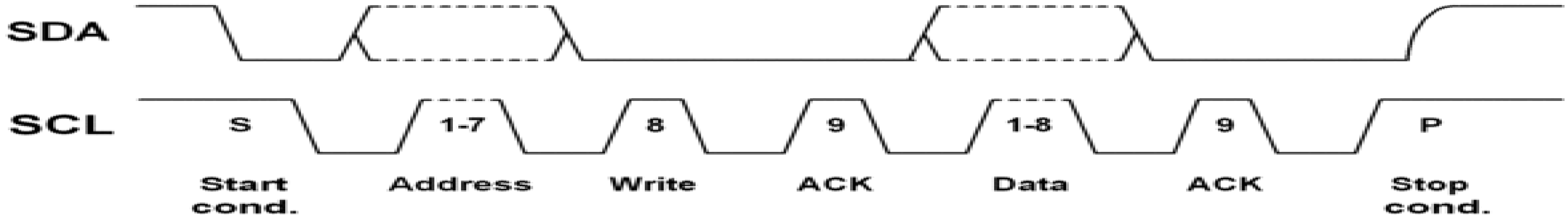
The I2C interface uses two bi-directional lines – Serial Clock (SCL) and Serial Data (SDA)

The two wires must be driven as open collector/drain outputs by all devices in the bus, and must be pulled high using one resistor each

This implements a 'wired AND function' - any device pulling the wire low causes all devices to see a low logic value - for high logic value all devices must stop driving the wire (make it as High-Z)



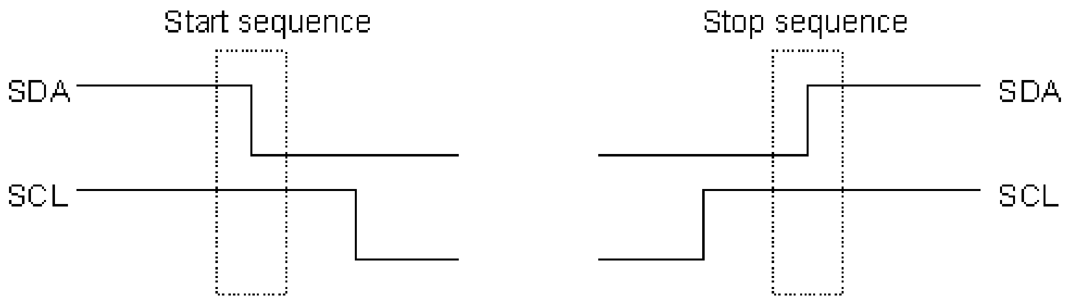
I2C - Basic Command Sequence



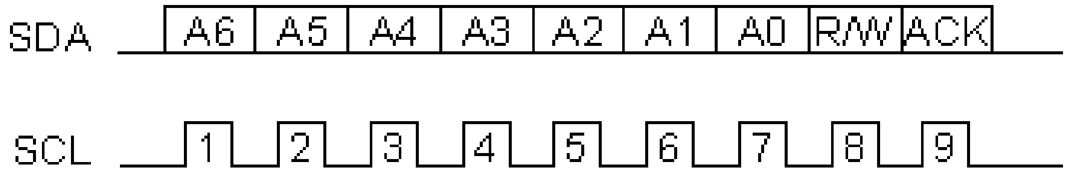
1. Send the START sequence - SDA transitions to low with SCL being high
2. Send the slave address - You can use 7 bit or 10 bit addresses.
3. Send the Read(R)-1 / Write(W)-0 bit.
4. Wait for/Send an acknowledge bit (A) – (ACK-0 / NACK-1)
5. Send/Receive the data byte (8 bits) (DATA).
6. Expect/Send acknowledge bit – (ACK-0 / NACK-1)
7. Send the STOP sequence - SDA transitions to high with SCL being high

The I2C Physical Protocol

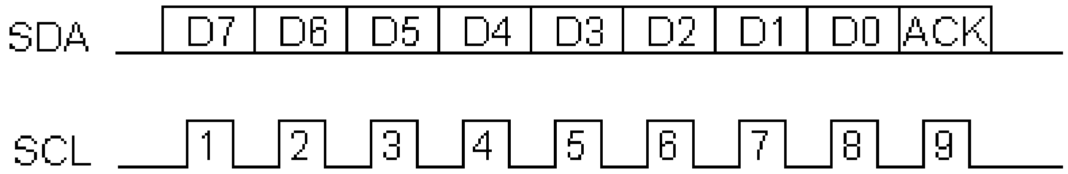
Start and Stop sequence



Sending Device Address



Transfer of data



I2C - MASTER writes to a SLAVE (typical)

1. Send a START sequence
2. Send the I2C address of the SLAVE with the R/W bit low (even address)
3. *send the device register you want to write to*
4. Send the data byte
5. *send any further data bytes*
6. Send the STOP sequence.

	START	ADDRESS	W	ACK	*[device register]* DATA	ACK	DATA	ACK	STOP
Bits		7	0	0	8	0	8	0	
			W		1		0x10		

ACK/NACKs

- Slave will ACK/NACK after every 8 bits
- If Slave ACKs, master can send one more byte of data
- If Slave NACKs, master should abort the communication

	START	ADDRESS	W	ACK	DATA	NACK	STOP
Bits		7	0	0	8	1	

(white box)	**Sent by master**
(grey box)	**Sent by Slave**

I2C - MASTER reads from a SLAVE

Typical Read :

1. Send a START sequence
2. Send I2C address of the SLAVE with the R/W bit high (odd address)
3. Read data byte from SLAVE
4. Send the STOP sequence.

Read from register address: (Concept : First write, then read)

1. Send a START sequence
2. Send I2C address of the SLAVE with the R/W bit low (even address)
3. Send device register address you want to read from
4. Send a START sequence again (or a repeated start – see next page)
5. Send I2C address of the SLAVE with the R/W bit high (odd address)
6. Read data byte from SLAVE
7. Send the STOP sequence.

Device Registers	
Address	Definition
0	Status
1	LED1_Status
2	LED2_Status
3	ADC_MSB
4	ADC_LSB
..	
..	
..	

I2C - MASTER reads from a SLAVE

General Read:

	START	ADDRESS	R	ACK	DATA	ACK	DATA	NACK	STOP
Bits		7	1	0	8	0	8	1	

OR

Read from register address:

	START	ADDRESS	W	ACK	Register Data	ACK	Repeat START	ADDRESS	R	ACK	DATA	ACK	DATA	NACK	STOP
Bits		7	0	0	8	0		7	1	0	8	0	8	1	

- ACK/NACKs
  - Slave will ACK only once for the address byte
  - Master will ACK, if it wants to read one more byte from slave
  - Master will NACK, if it wants to stop the reading
  - Note that the slave does not have an option to stop the communication once it has acknowledged the address.

Sent by master

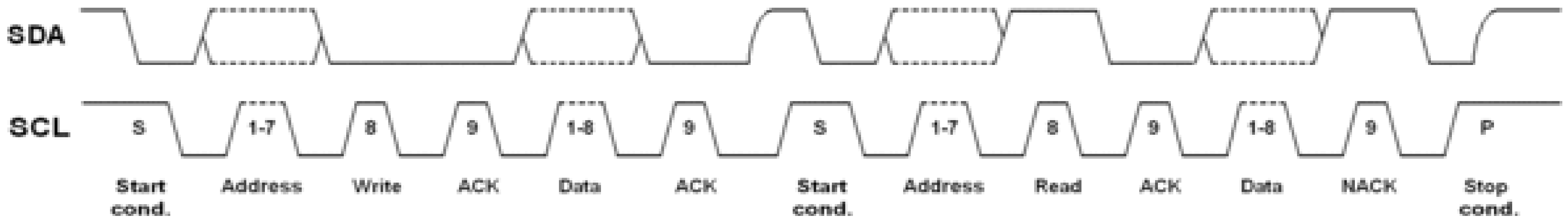
Sent by Slave

Repeated Start

A way to claim the bus : During an I2C transfer there is often the need to first send a command and then read back an answer right away. This has to be done without the risk of another (multimaster) device interrupting this atomic operation.

Send a start condition before sending a stop

Typically used when you want to write and read immediately or when you want to write to two different registers sequentially



Note : Regardless of the number of start conditions sent during one transfer the transfer must be ended by exactly one stop condition.

Clock stretching

A slow slave device may need to stop the bus while it gathers data or services an interrupt etc.

It can do this while holding the clock line (SCL) low forcing the master into the wait state.

The master must then wait until SCL is released before proceeding.

As per specification,

- the slave can hold the SCL line after any clock bit
- any device can hold down SCL as long as it likes!!

Arbitration

Occurs in Multi-master mode.

Happens when two masters start a transfer at the same time.

During the transfer, the masters constantly monitor SDA and SCL.

If one of them detects that SDA is low when it should actually be high, it assumes that another master is active and immediately stops its transfer.

This process is called arbitration.

I2C - Typical devices types

IO expanders

EEPROMs

DACs, ADCs

LCD/LED drivers

Capacitive sensors

NVRAM

Real-Time clocks (RTC)

Digital Temperature ICs

Accelerometers

Compass

Third party sensors with an I2C interface (sonars, IR range finders, NXT I2C sensors)

More materials on I2C…

Want to learn more?

- Check the I2C Primer from http://www.i2c-bus.org
- The latest I2C specification is available directly from NXP.
- There are a number of I2C-like buses, see Definitions and Differences Between I2C, ACCESS.bus and SMBus.
- The I2C-Bus and how to use it is a well-known document from Philips discussing the use of this bus in applications.

Wire Lib

```
void begin(int sda, int scl, uint8_t address)
```

Description	This function to start i2c module
Parameter	sda, scl and address
Return	Void
Example	Wire.begin(D2,D1,0x10); // starts i2c master with address 0x10

```
void setClock(uint32_t frequency)
```

Description	This function sets i2c clock speed
Parameter	frequency
Return	Void
Example	Wire.setClock(400000); // set clock speed to 400Khz

Wire Lib

```
void beginTransmission(uint8_t address)
```

Description	This function begins i2c transmission
Parameter	address
Return	Void
Example	Wire. beginTransmission(0x10); //begins transmission to i2c address 0x10

```
uint8_t endTransmission(uint8_t sendStop)
```

Description	This function ends i2c transmission
Parameter	address
Return	Void
Example	Wire. endTransmission(0x10); // ends i2c transmission to address 0x10

Wire Lib

```
size_t write(uint8_t data)
```

Description	This function writes data to i2c
Parameter	Data
Return	Void
Example	Wire. write(‘a’); // writes character ‘a’ to i2c

```
size_t write(const uint8_t *data, size_t quantity)
```

Description	This function writes array bytes to i2c
Parameter	Data, quantity
Return	Void
Example	Wire. write(buffer,10); //writes 10 bytes of data in buffer.

Wire Lib

`int available(void)`

Description	This function checks any i2c data available
Parameter	None
Return	int
Example	Wire. available();

`int read(void)`

Description	This function reads data from i2c
Parameter	None
Return	int
Example	Wire. read();