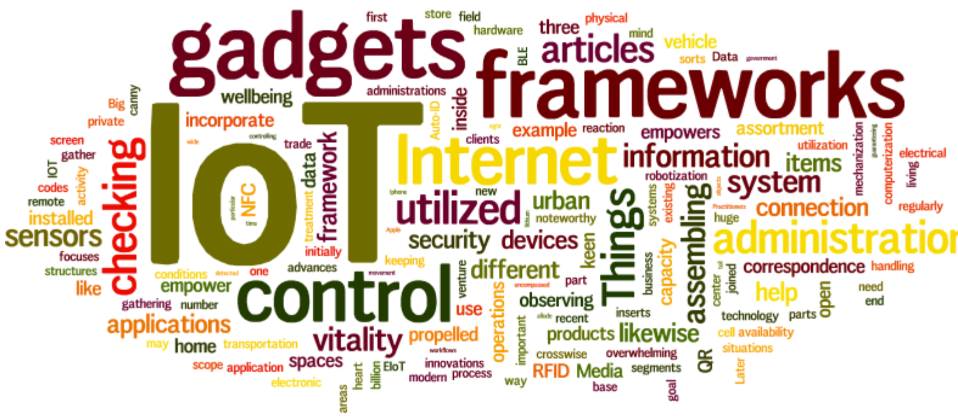
CS578: Internet of Things

I2C Protocol

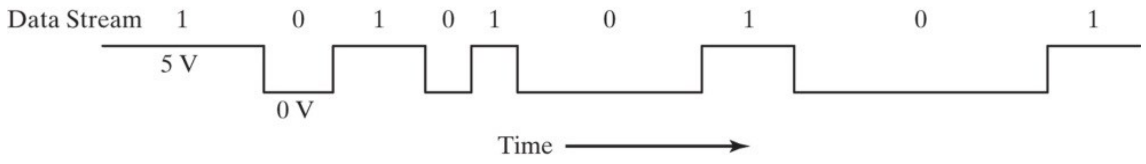
Demo Using NodeMCU and Arduino



Dr. Manas Khatua
Assistant Professor, Dept. of CSE, IIT Guwahati
E-mail: manaskhatua@iitg.ac.in

"We suffer as a result of our own actions; it is unfair to blame anybody for it." – **Ma Sarada Devi**

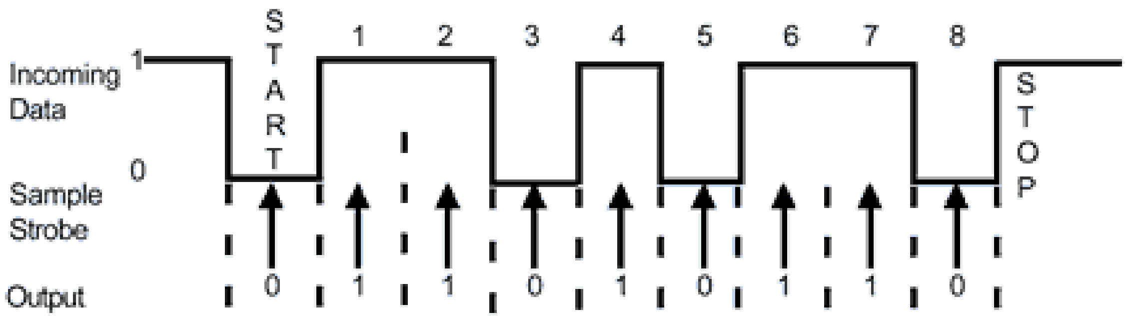
Bit serial communication concepts



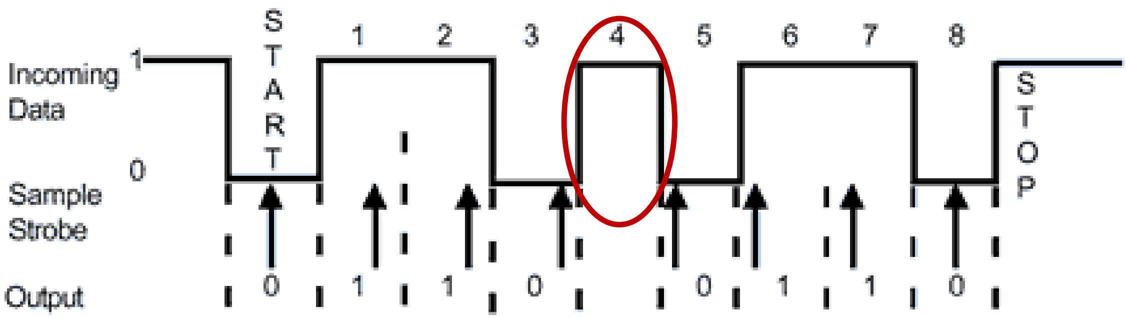
- Serial → one bit at a time
- Most often using *logic level* signals
- Timing information needs to be shared between sender and receiver
  - Timing information: *Transition point between bits*, *duration of a bit*

- Two major types of bit serial protocols
  - Asynchronous (no shared clock)
    - Sender and receiver maintain independent clocks
      - e.g. RS-232, USB, UART
  - Synchronous (shared clock)
    - e.g. SPI, I^2C

Issue in Asynchronous Data Sampling



a. Best case, receiver samples at midpoint of each bit.

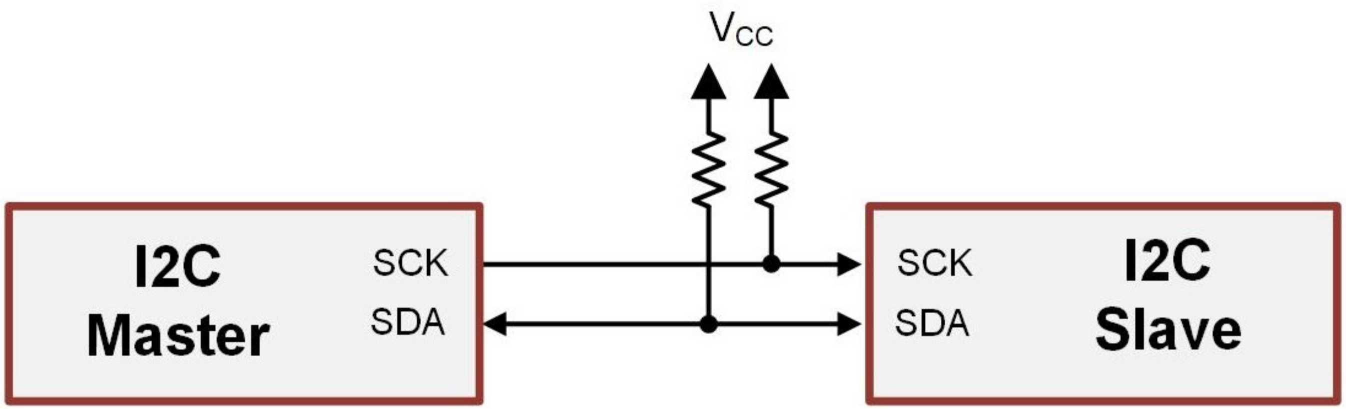


b. Receiving clock is too slow, causing bit 4 to be skipped and the data to be corrupted.

Ideal and corrupted asynchronous data sampling

Source: http://www.quatech.com/support/figures/async1.gif

I2C Introduction



- I2C – Inter Integrated Circuit
- One of the widely used Serial Communication protocols
- Created by Philips Semiconductor in 1982 (Now it is NXP Semiconductor)
- No license needed since 2006

Source: Embedded Systems Design, by Brock J. LaMeres, Springer Publisher

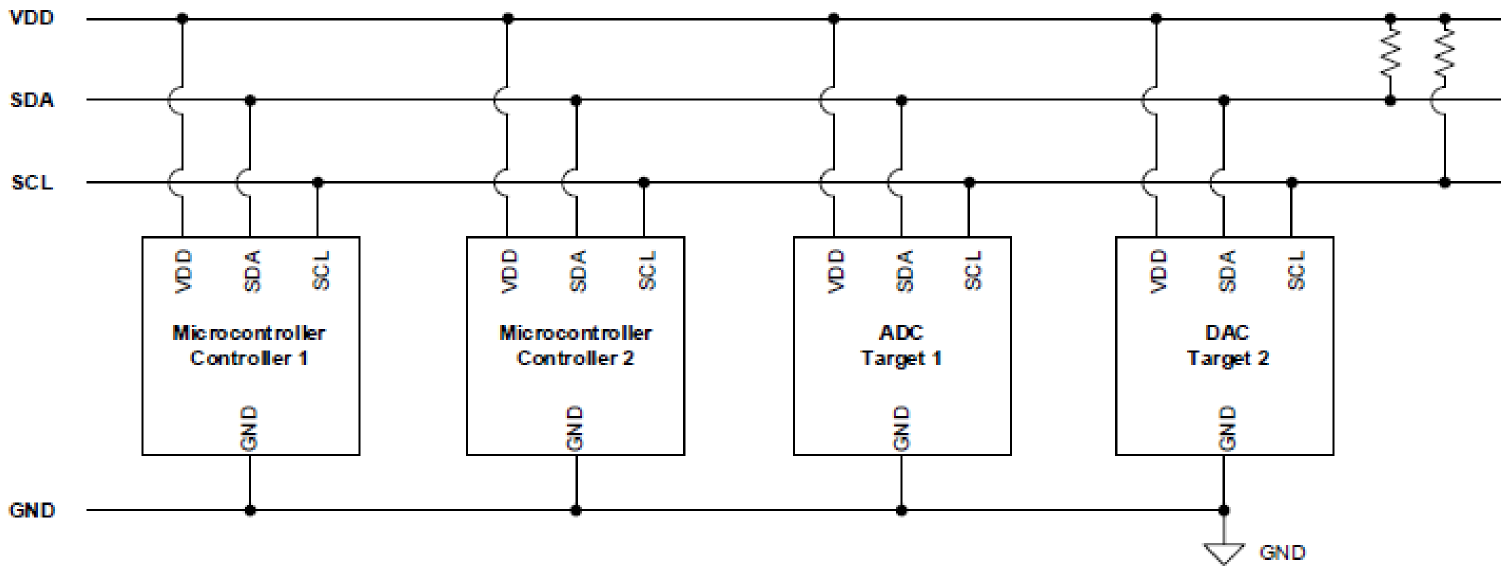
I2C Communication Modes

I2C Mode	Speed
Standard Mode	100 kbps
Fast Mode	400 kbps
Fast Mode Plus	1 Mbps
High Speed Mode	3.4 Mbps
Ultra-Fast Mode	5 Mbps

Similar in implementation, with different timing requirements (Standard Mode, Fast Mode, Fast Mode Plus)

Requires specific controller code for high speed transfer (High Speed Mode, Ultra-Fast Mode)

I2C Physical Layer



V_{DD}: Voltage Drain Drain

SDA: Serial Data

SCL: Serial Clock

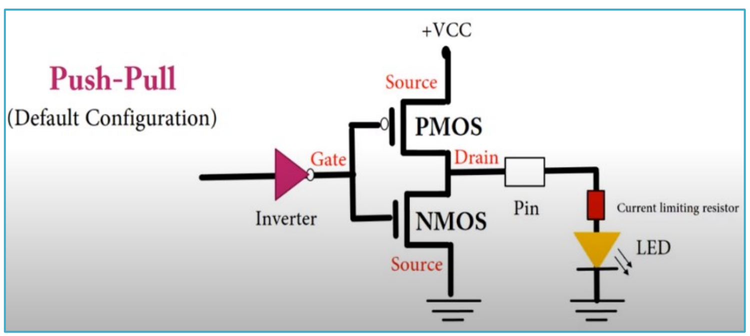
- Only two communication lines for all devices on the bus (SDA, SCL)
- Bi-directional communication
- I2C link is half-duplex, means only one device transmits at any given time.
- Allows for multiple controllers and multiple targets
- Both the signal lines (SDA, SCL) are 'open drain', thus pull-up resistors are needed

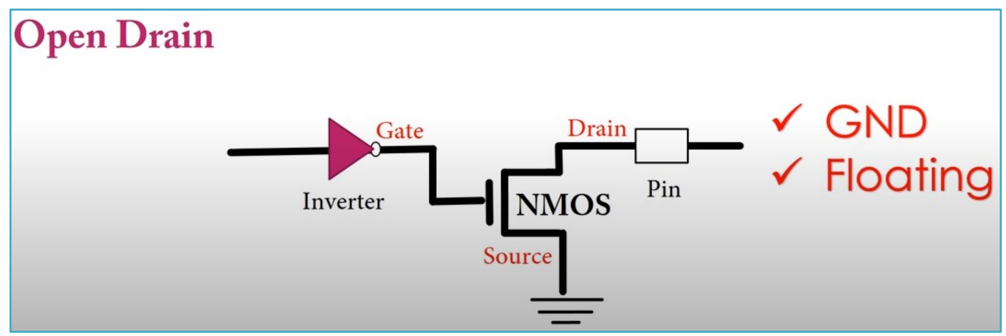
Open-Drain and Push-Pull Configuration

- Open-drain refers to a type of output which can
  - either "pull" the bus down to a voltage (ground, in most cases),
  - or "release" the bus and let it be pulled up by a pull-up resistor
- I2C uses an open-drain/open-collector with an input buffer on the same line, which allows a single data line to be used for bidirectional data flow.

- Open-drain output stage supports multiple drivers on the same signal line using NMOS transistor

When a pin is configured in output mode

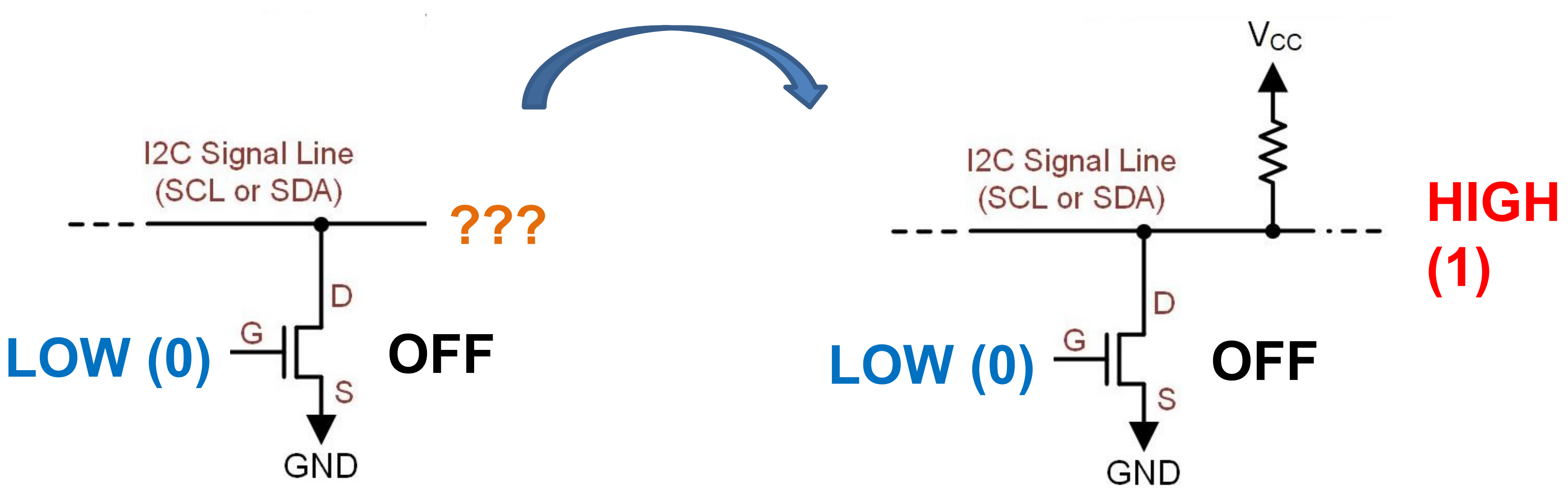




Pull Up Resistors

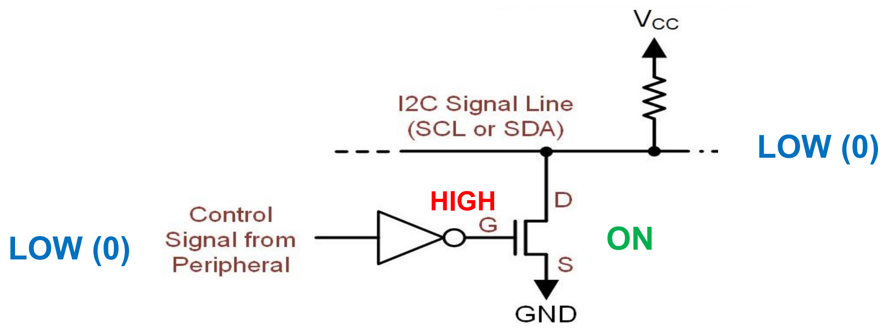
- Consider the Open-Drain Output Stage:
  - If we turn **off** the NMOS by driving a LOW (0) to its gate, the line is left *floating,* which is an unknown logic level.
  - If a pull-up resistor it placed on the line, it will pull the resistor to a HIGH (1) when the NMOS is off.

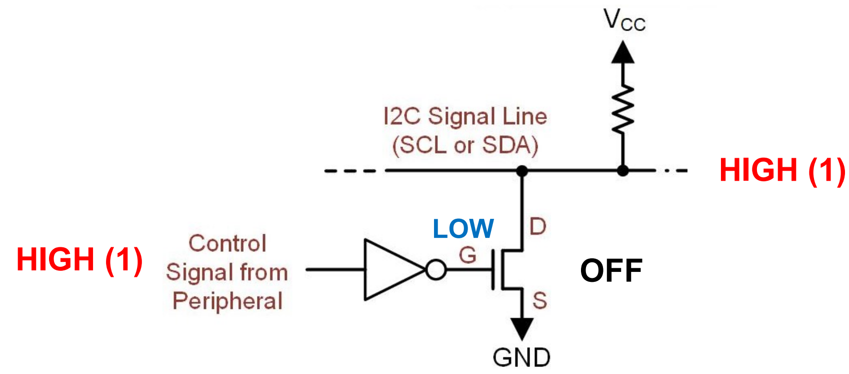
- Note, here we have an inverted logic scheme where driving a LOW to the NMOS yields a HIGH on the line, and vice-versa.



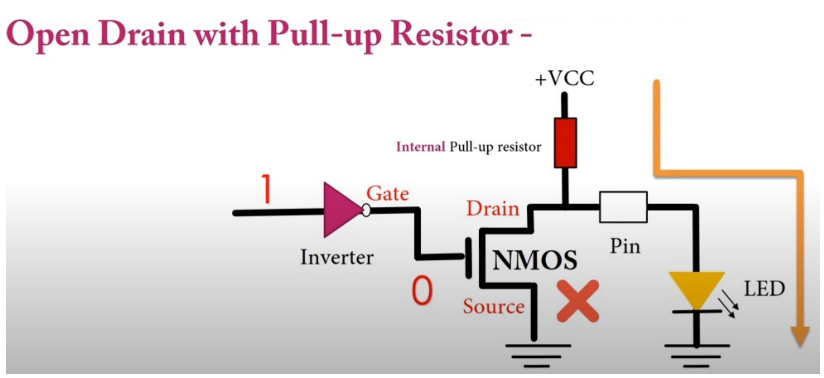
Cont…

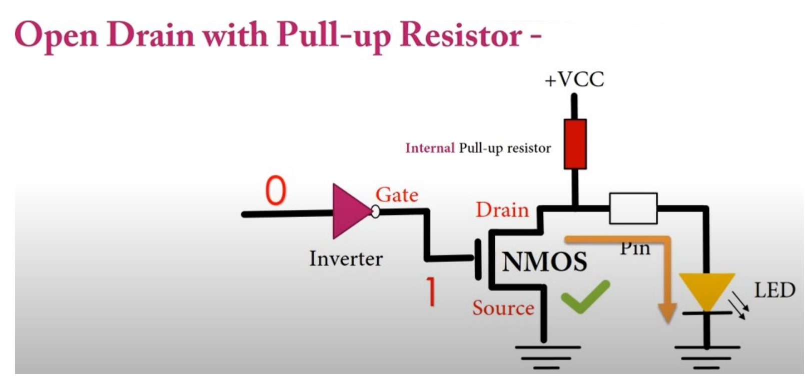
To get back to positive logic, we insert an inverter before the NMOS.





Example with LED operation:

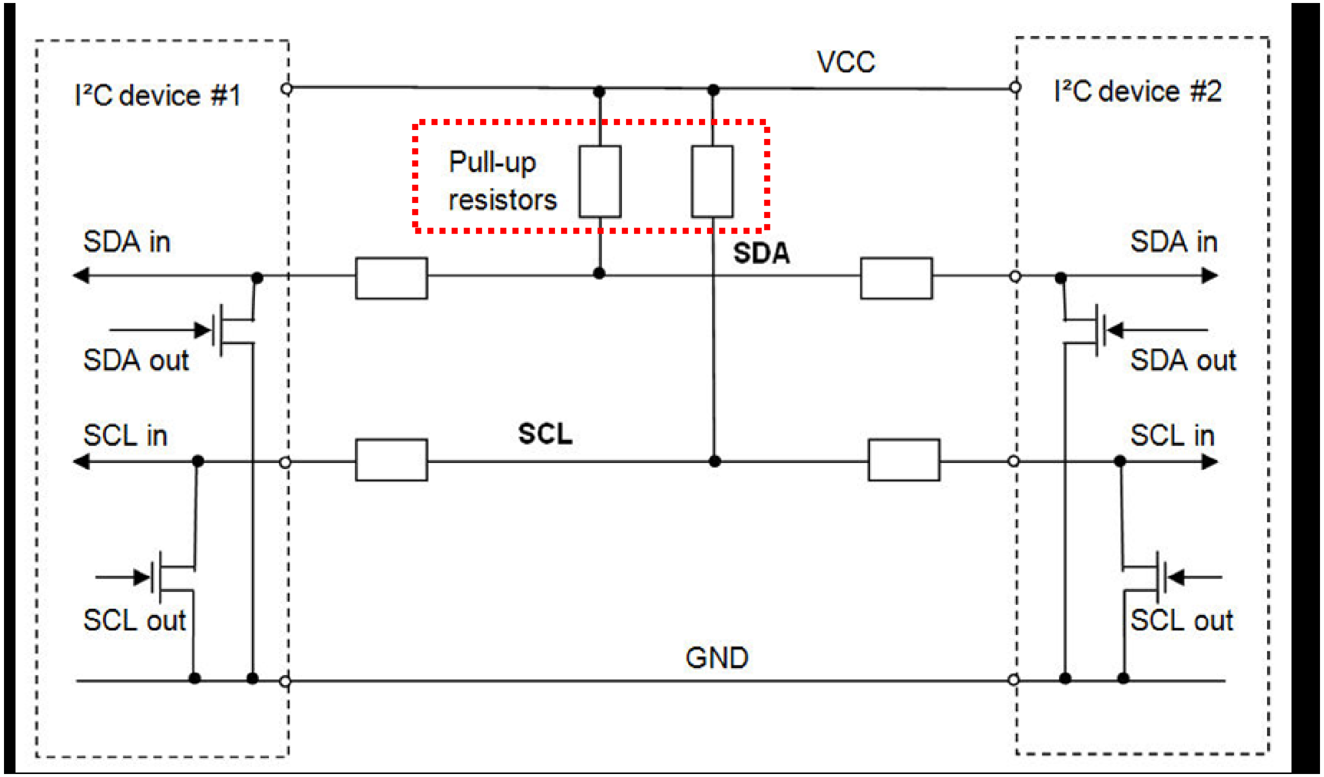




Source: Embedded Systems Design, by Brock J. LaMeres, Springer Publisher

Cont…

So, an I2C bus ALWAYS needs external pull-up resistors on each of its lines



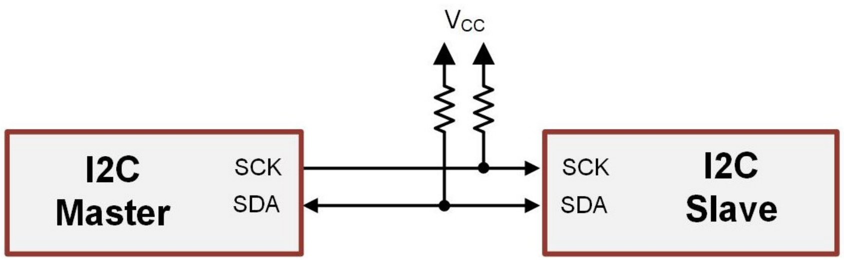
I2C Protocol Operation

Master device – the device that initiates communication and controls the clock.

Slave device – a device on the bus that is read or written to, but does not initiate transmission or provide a clock.

Slave address – a unique and predetermined address for each slave on the bus.

This address is used by the master to indicate which slave it wants to communicate with.

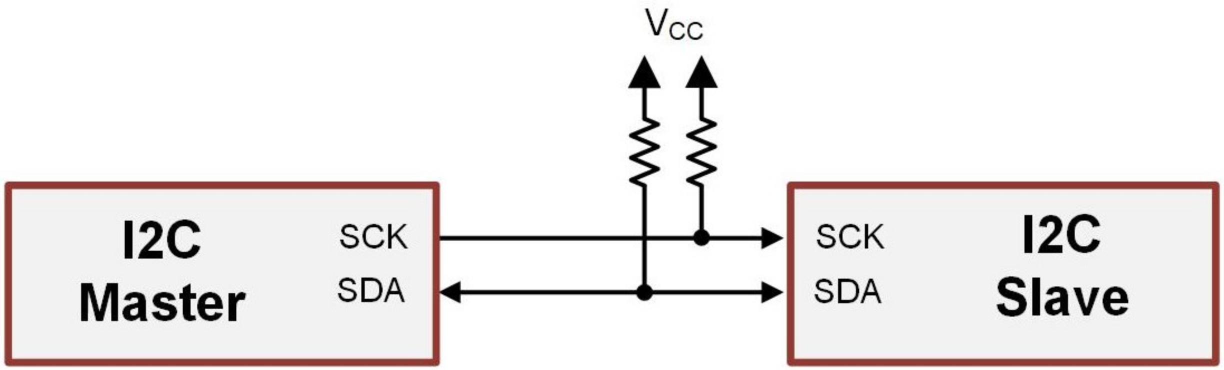


Idle – when both SDA and SCL are held high by the pull-up resistors and no I2C device is attempting to communicate.

Busy – when devices are driving the bus.

Messages – how I2C information is transferred.

Operation Steps in I2C - 1/12

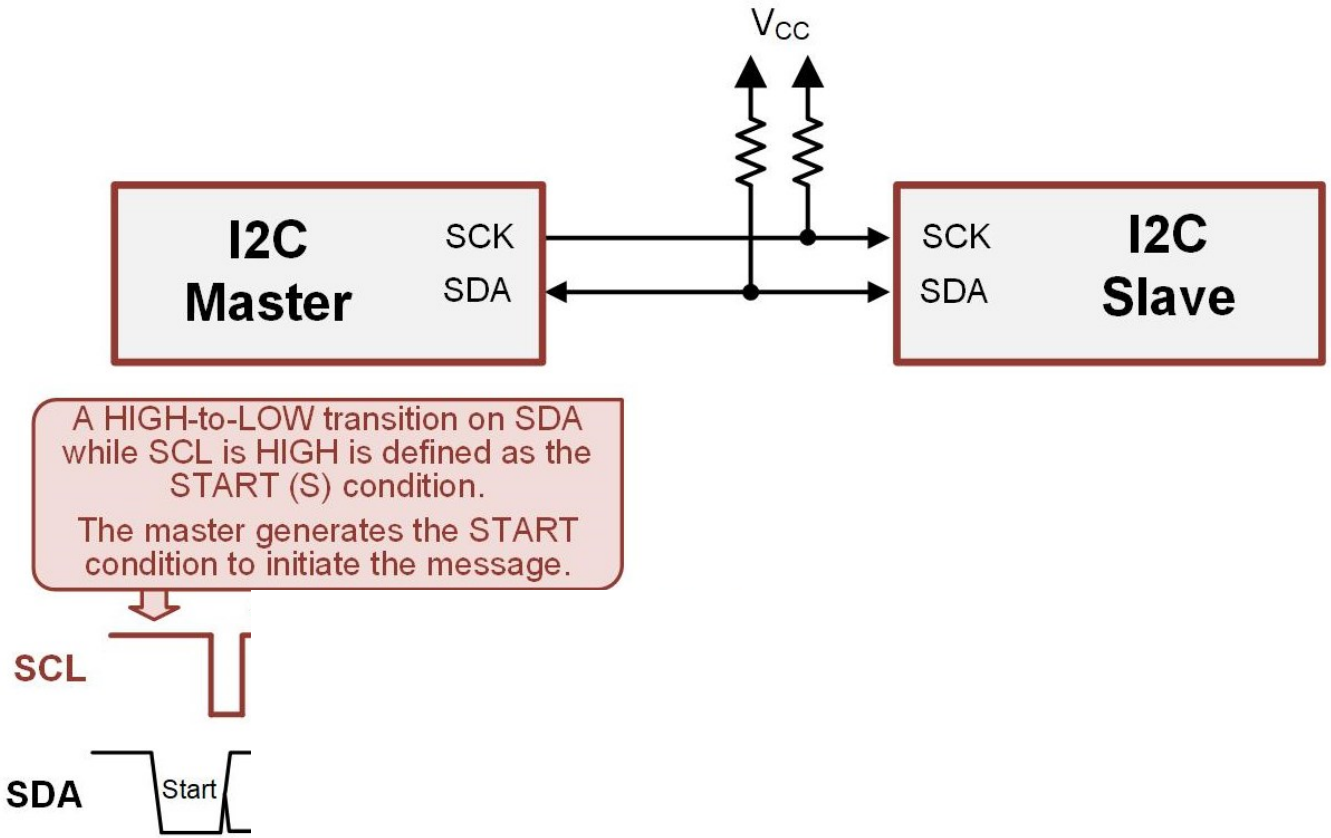


SCL

SDA

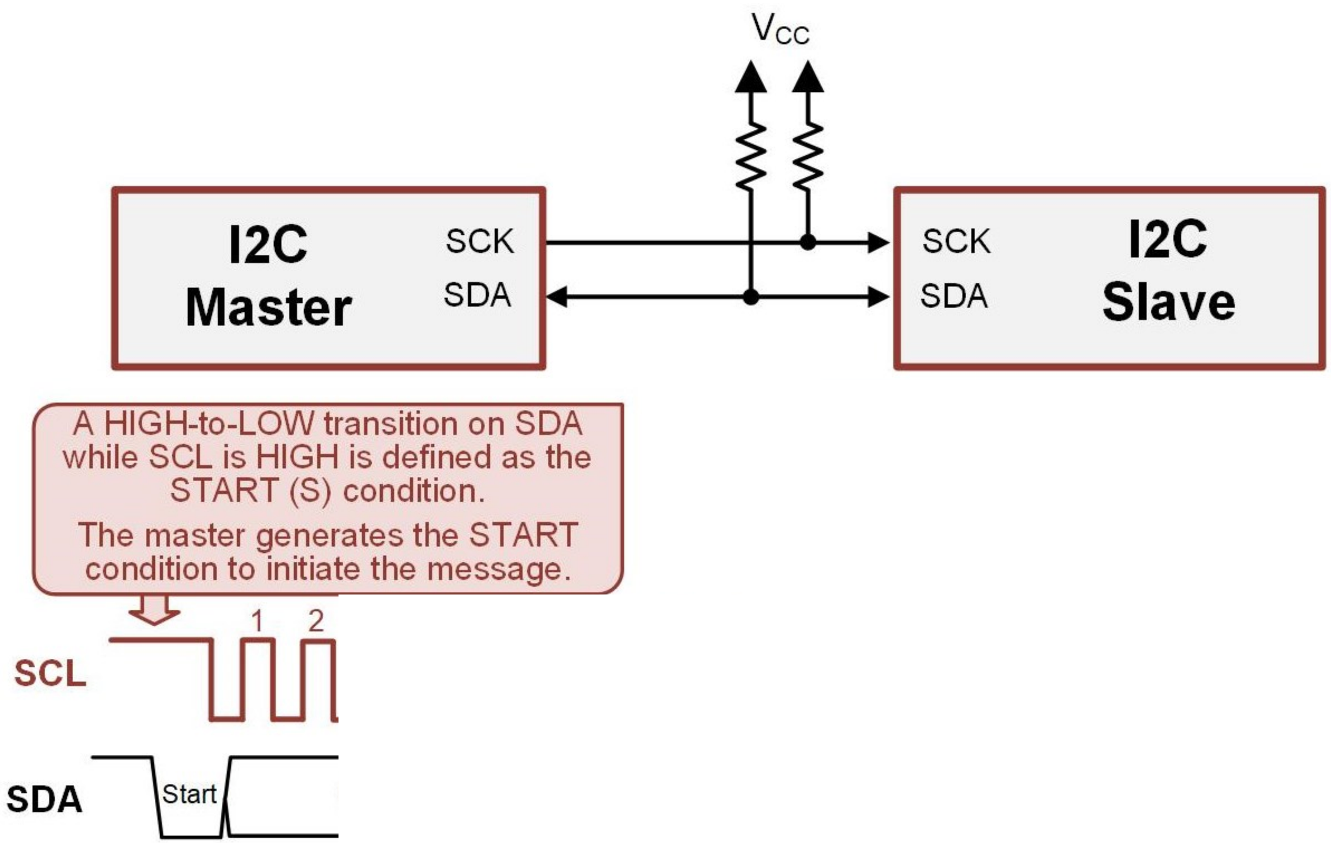
- SCL and SDA both are in Idle State at the beginning.

Operation Steps in I2C - 2/12



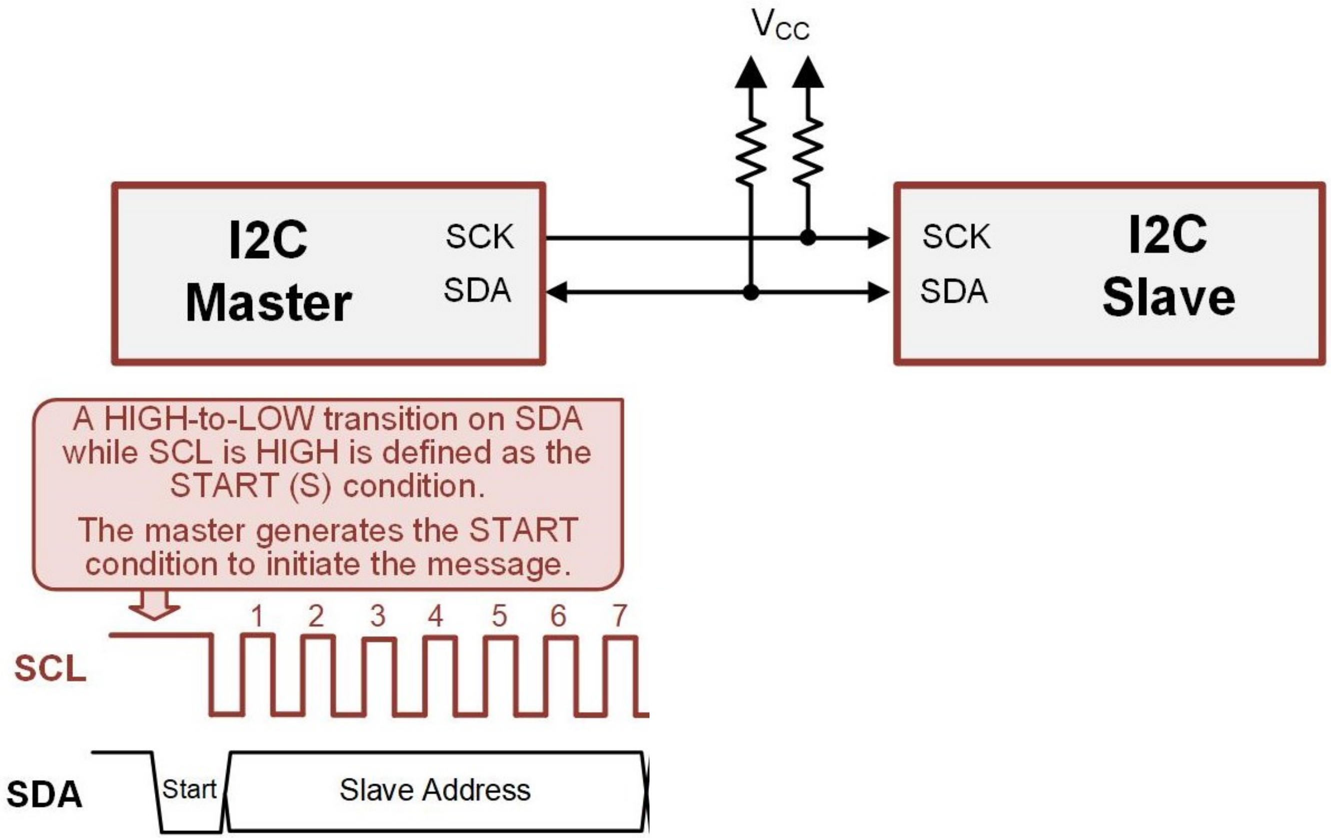
- A master initiates a new message by generating a START(S) condition by pulling SDA LOW while SCL is still HIGH.
- As soon as the START condition is generated, the SCL will be pulled LOW and start pulsing to provide the clock for the message.

Operation Steps in I2C - 3/12



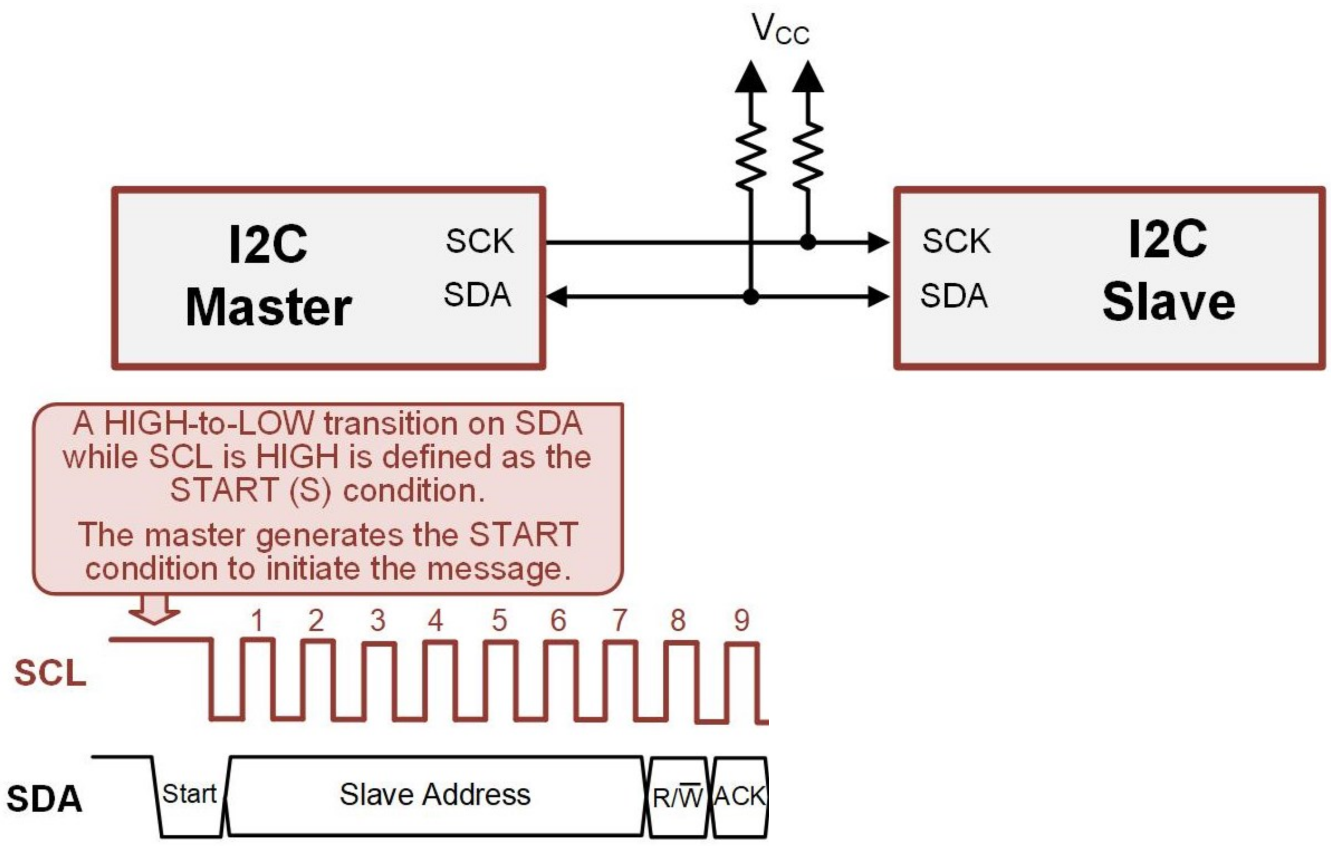
- Each clock pulse within the I2C message is numbered by periods.
- Both the master and the slaves count the number of periods that have occurred since the message started in order to know when certain frames and signals should be present.

Operation Steps in I2C - 4/12



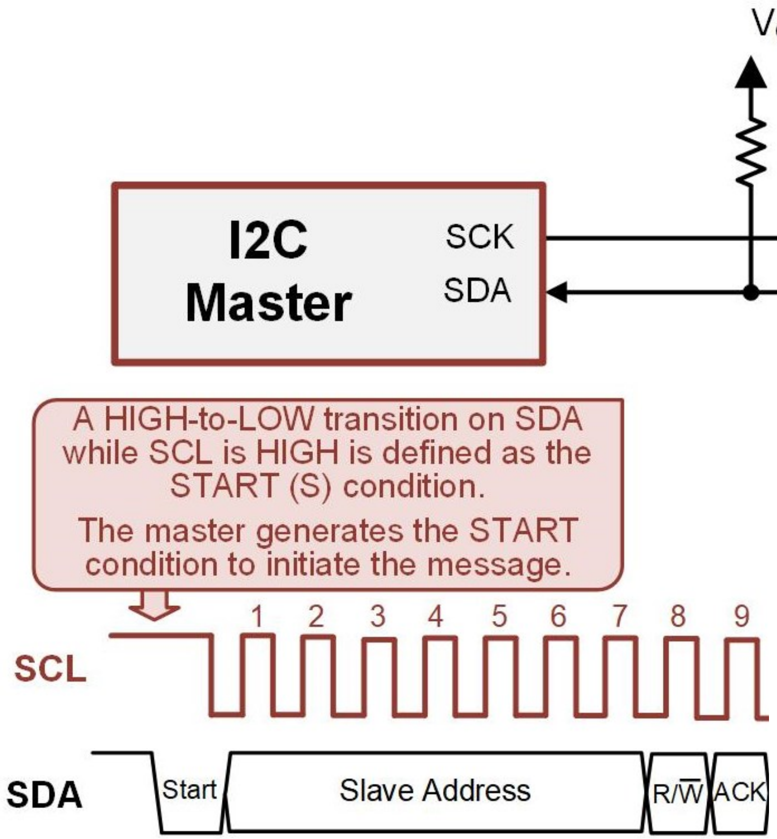
- After the master generates the START condition, it first sends the slave address that it wishes to communicate with.
- I2C slave addresses can either be 7-bit (default) or 10-bit.

Operation Steps in I2C - 5/12



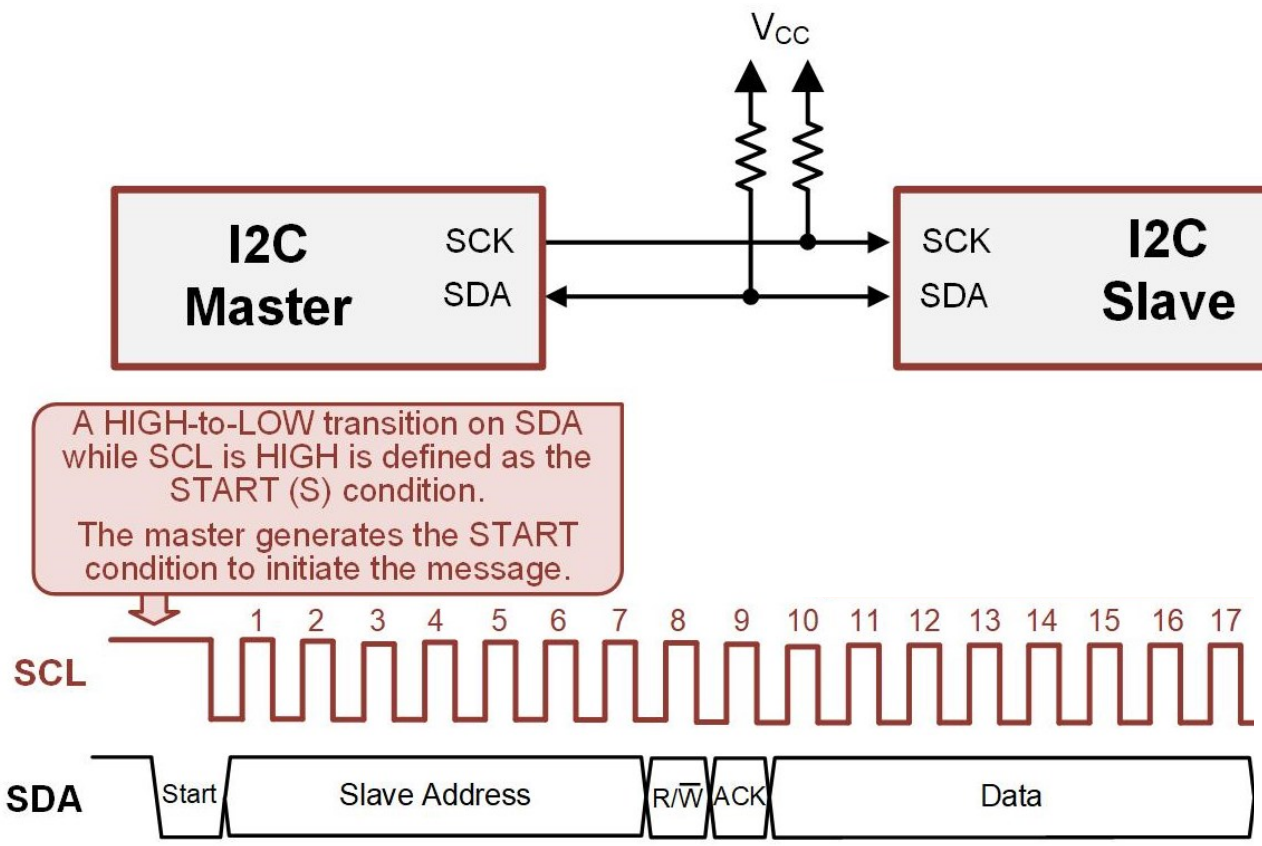
- After the slave address and read/write signal (1 bit: 0 to write, 1 to read) are sent by the master, each slave on the bus checks whether it is being addressed.
- Period 9 of the message is reserved for the slave acknowledge (ACK) or no-acknowledge (NACK) signal.

Operation Steps in I2C - 6/12



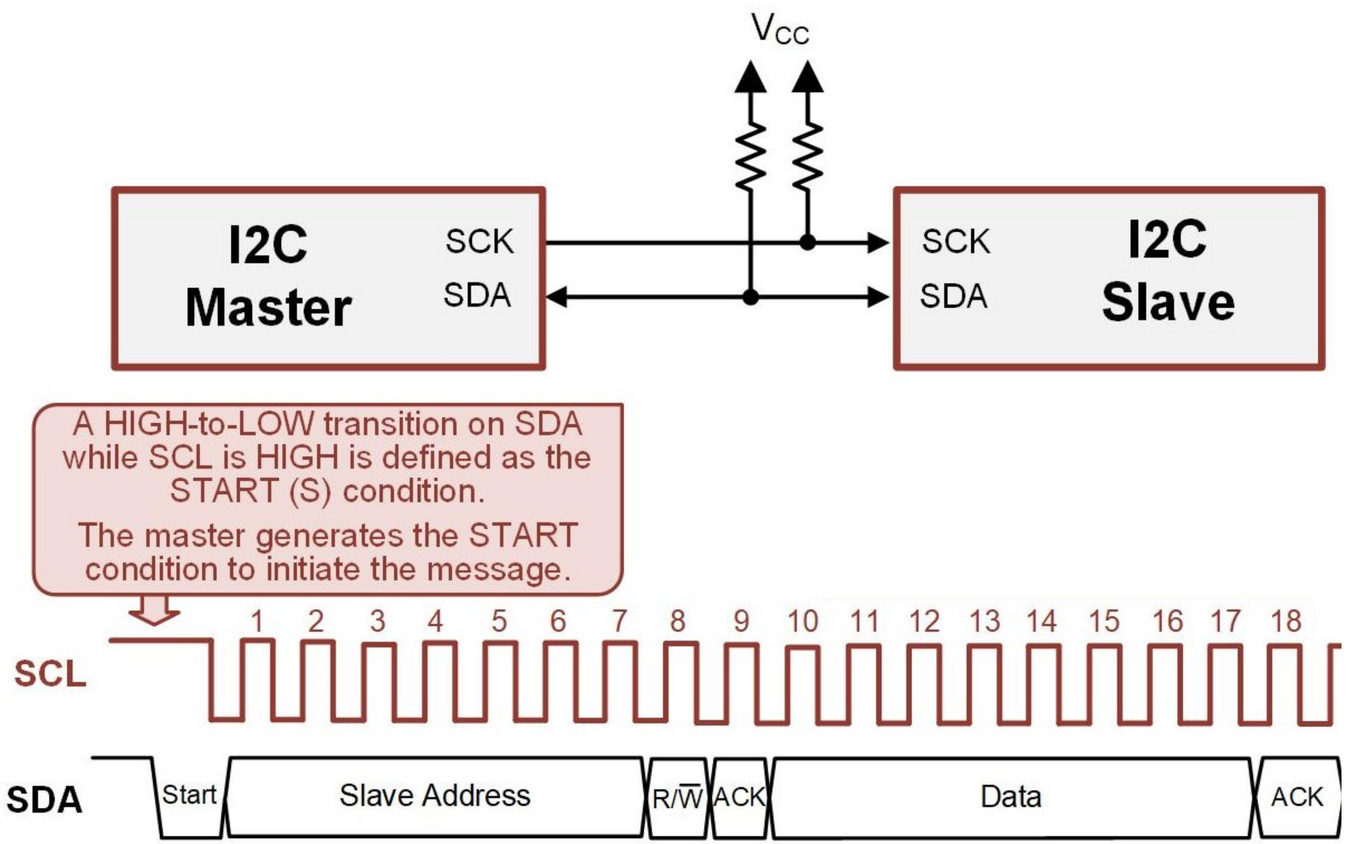
- Intended slave will send an ACK signal back to the master by pulling SDA LOW.
- If no device exists with the specified slave address, no device will pull down SDA. This will result in period 9 remaining HIGH and will be interpreted as a NACK.

Operation Steps in I2C - 7/12



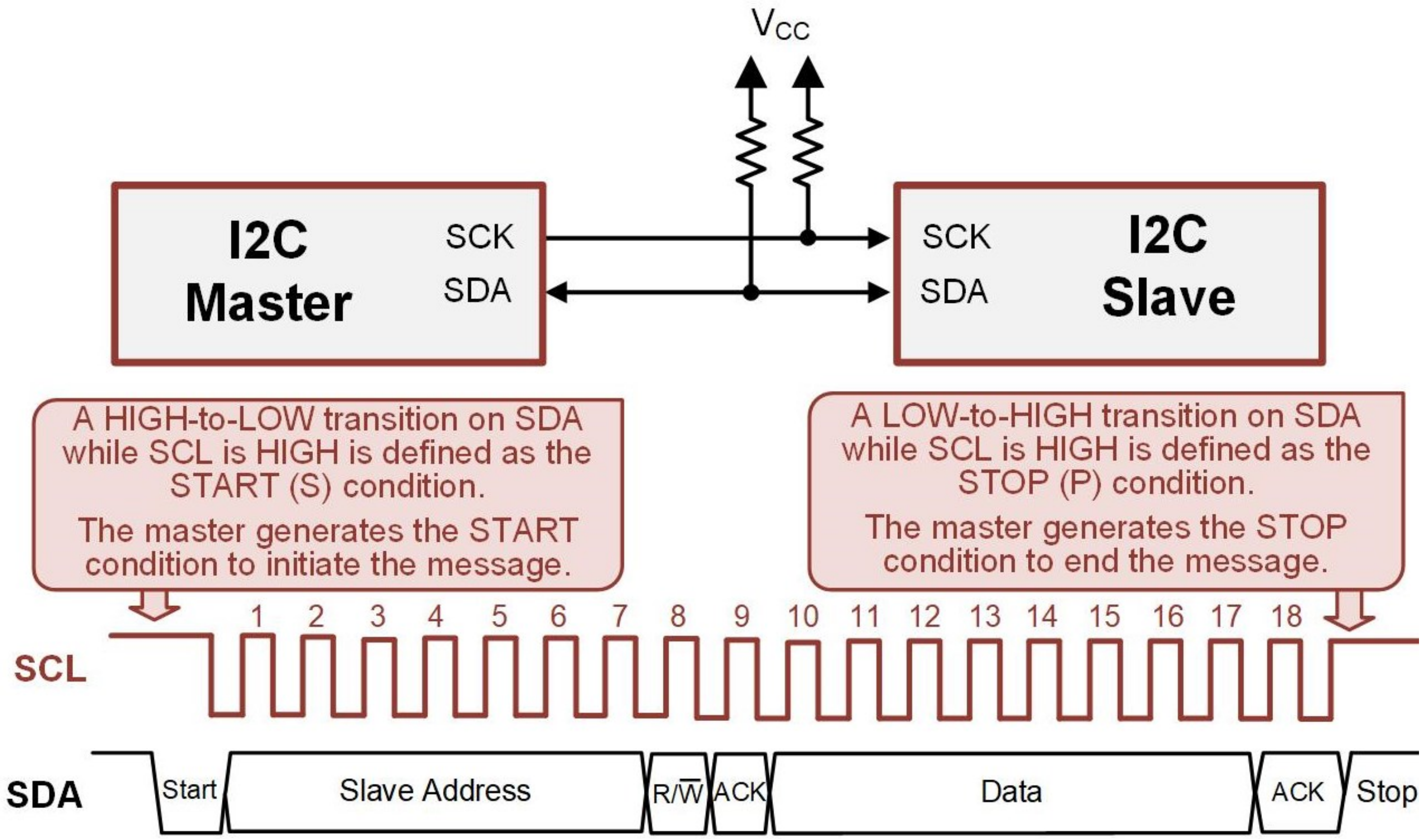
- If the master sees the ACK signal, it knows a slave exists with the specified address and proceeds with the message.

Operation Steps in I2C - 8/12



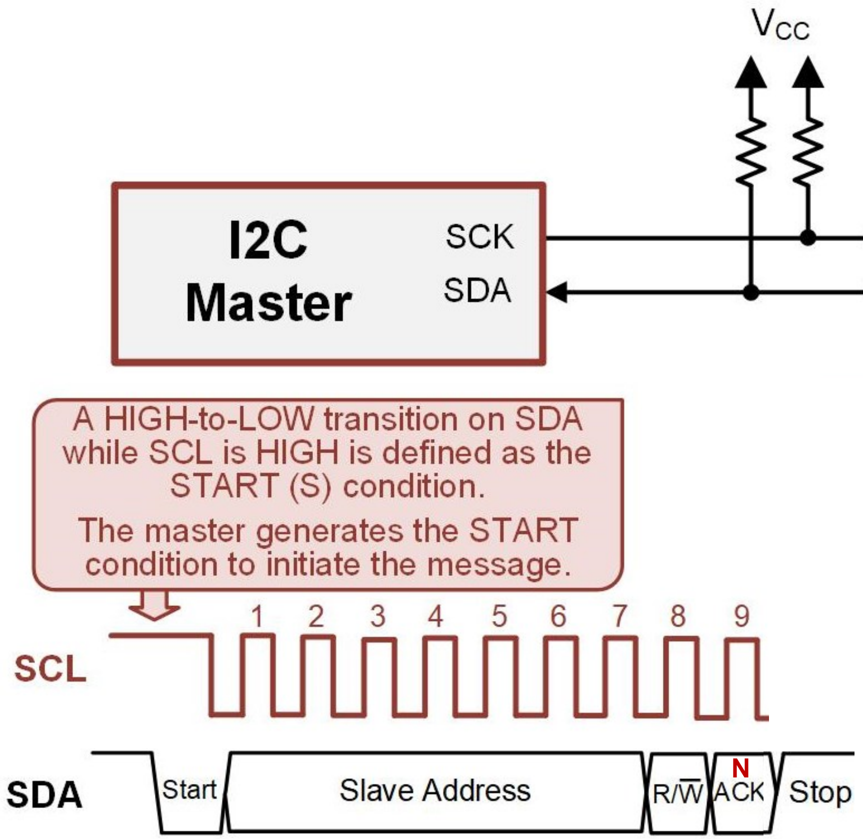
- After each byte is sent, the receiving device sends an ACK signal indicating that it successfully received the data.

Operation Steps in I2C - 9/12



- A STOP condition occurs when there is a LOW-to-HIGH transition on SDA while SCL is HIGH.
- Once, SDA goes HIGH, SCL also remains HIGH indicating that the bus is idle again.

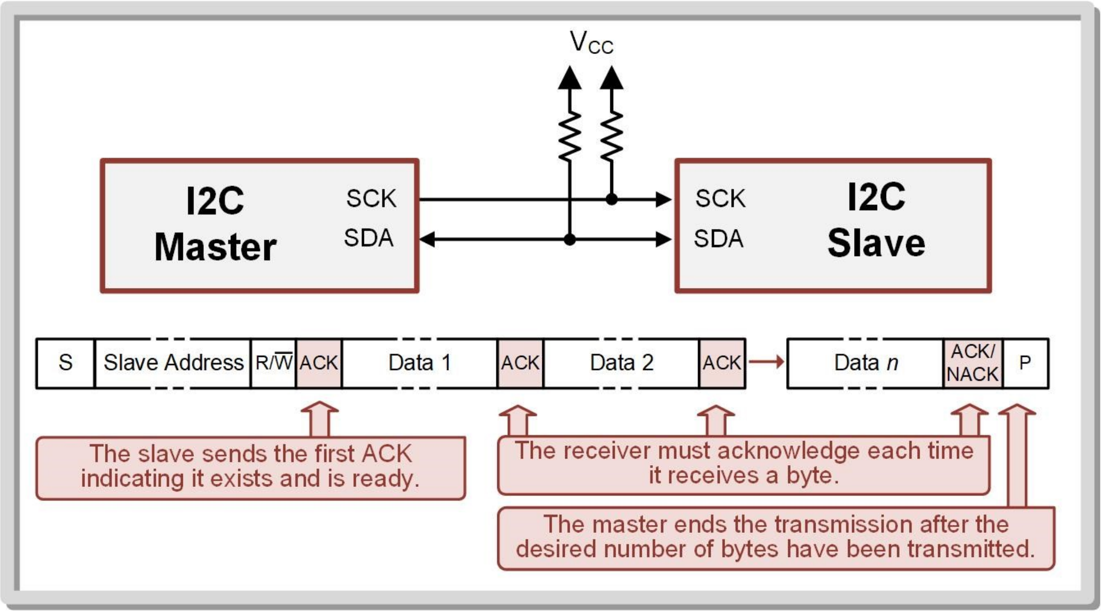
Operation Steps in I2C - 10/12



- A NACK in period 9 tells the master that no slave exists with the specified address.
- The master then generates a STOP condition and ends the message.

Operation Steps in I2C - 11/12

- When the master is writing to a slave, the master sends the 8-bits of data and the slave produces the ACK/NACK signal.
- When the master is reading from a slave, the slave sends the 8-bits of data and the master produces the ACK/NACK signal.
- After the data has been sent and acknowledged, the master can end the message by generating the STOP condition anytime.
- The Master can send multiple data bytes in a single message.



Operation Steps in I2C - 12/12

General structure of a 2-byte transfer

START	Slave address	Rd/nWr	ACK	Data	ACK	Data	ACK	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Writing 2-bytes (shaded bits are put on the bus by the master)

START	Slave address	0	0	Data	0	Data	0	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Reading 2-bytes (shaded bits are put on the bus by the master)

START	Slave address	1	0	Data	0	Data	1	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Source: http://www.byteparadigm.com/kb/article/AA-00255/22/Introduction-to-SPI-and-IC-protocols.html

Demo

- ✓ I2C communication between Arduino UNO and Node MCU
- ✓ we choose Node MCU as master device and Arduino as slave device

Hardware Connection between Arduino and NodeMCU

- Connect the SDA pin of Arduino to SDA pin (D1) of NodeMCU
- Connect the SCL pin of Arduino to SCL pin (D2) of NodeMCU

- Connect the Ground pin of Arduino to ground pin of NodeMCU
- Plug Arduino and NodeMCU to laptop / PC through USB cable to give power

Demo Code of NodeMCU (master)

```
#include <Wire.h>

void setup() {
 Serial.begin(9600);  /* begin serial for debug */
 Wire.begin(D1, D2);  /* join i2c bus with SDA=D1 and SCL=D2 of NodeMCU */
}

void loop() {
 Wire.beginTransmission(8);   /* begin with device address 8 */
 Wire.write("Hello Arduino");            /* sends hello string */
 Wire.endTransmission();      /* stop transmitting */

 Wire.requestFrom(8, 13); /* request & read 13 byte data from slave device #8 */
 while(Wire.available()){
    char c = Wire.read();
  Serial.print(c);
 }
 Serial.println();
 delay(1000);
}
```

Demo Code of Arduino (slave)

```
#include <Wire.h>

void setup() {
 Wire.begin(8);                  /* join i2c bus with address 8 */
 Wire.onReceive(receiveEvent); /* register receive event */
 Wire.onRequest(requestEvent); /* register request event */
 Serial.begin(9600);             /* start serial for debug */
}

void loop() {
 delay(100);
}

// function that executes whenever data is received from master
void receiveEvent(int howMany) {
    while (0 <Wire.available()) {
    char c = Wire.read();        /* receive byte as a character */
    Serial.print(c);             /* print the character */
  }
 Serial.println();               /* to newline */
}

// function that executes whenever data is requested from master
void requestEvent() {
 Wire.write("Hello NodeMCU");  /*send string on request */
}
```

Wire Library

- This **Wire** library allows you to communicate with I2C/TWI devices
- There are both 7 or 8-bit versions of I2C addresses. 7 bits identify the device, and the 8^{th} bit determines if it's being written to or read from.
- The **Wire** library uses 7 bit addresses throughout. However, the addresses from 0 to 7 are not used because are reserved so the first address that can be used is 8.
- **Functions in Wire.h**

begin()	end()
requestFrom()	beginTransmission()
endTransmission()	write()
available()	read()
setClock()	onReceive()
onRequest()	setWireTimeout()
clearWireTimeoutFlag()	getWireTimeoutFlag()

Demo Output

✓ Output in Master Device

COM7

```
Hello NodeMCU
Hello NodeMCU
Hello NodeMCU
Hello NodeMCU
Hello NodeMCU
Hello NodeMCU
Hello NodeMCU
Hello NodeMCU
Hello NodeMCU
Hello NodeMCU
Hello NodeMCU
Hello NodeMCU
Hello NodeMCU
```

✓ Output in Slave Device

COM4 (Arduino/Genuino Uno)

```
Hello Arduino
Hello Arduino
Hello Arduino
Hello Arduino
Hello Arduino
Hello Arduino
Hello Arduino
Hello Arduino
Hello Arduino
Hello Arduino
Hello Arduino
Hello Arduino
Hello Arduino
```

Lessons Learned

- ✓ What is serial communication
- ✓ What is synchronous communication
- ✓ I2C communication mechanism
- ✓ Demo on I2C communication

Thanks!

Acknowledgement:

- Most of the images are taken from "Embedded Systems Design", by Brock J. LaMeres, Springer Publisher

- Source of the sample code used in demo: https://www.electronicwings.com/nodemcu/nodemcu-i2c-with-arduino-ide