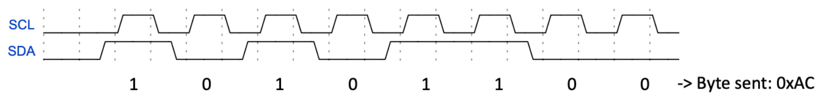
# Inter-Integrated Circuit (I2C)

- 2 wire ***serial*** protocol
  - Serial **Clock** Line (SCL)
  - Serial **Data** Line (SDA)
- *only master can initiate communication* (since master is responsible for toggling SCL and thus needs special hardware)
- handshake-side
- *open drain* config using *weak pull-up* resistors
  - *open drain* - type of output config; output uses a single N-channel MOSFET/NPN bipolar transistor
    - output only low/float; requires pull-up resistor to define high state
    - **pull the line to the ground (low state)** transistor inside device is turned on, connects output pin to GND, actively drives line to low state
    - **leave the line floating (high impedance/floating state)** transistor is turned off, disconnecting output pin from GND, leaving it in Z state (not actively connected to anything)
  - *weak pull-up resistors* - open drain cannot actively drive the line high; external pull-up to pull line to high state (VCC) when transistor is off
    - pulls line to a defined high state (weak 1) when no active device is driving it low
    - ensuring defined high state and low power consumption
    - **allows sharing of the line**
- unused bus reads as weak 1
- when dev wants to use bus, it pulls the line **low**
  - when another dev sees the line is low, it will not transmit
- **no dev can hold the bus high**; otherwise no one can use it
  - *all devices can always listen/receive when any device pulls the line low*
- only controllers create the Clk
  - dev can override this (clock stretching)
- supports different speed modes
  - faster speed → complex hardware
  - standard 100KHz; high speed 3400 KHz
- **unique ID** for each dev (7 bits)
  - Controller sets the ID of the dev it wants to communicate with (all device RX)
  - dev only uses bus if ID matches dev's own ID
- I2C dev can both transmit and receive or both(TX & RX)

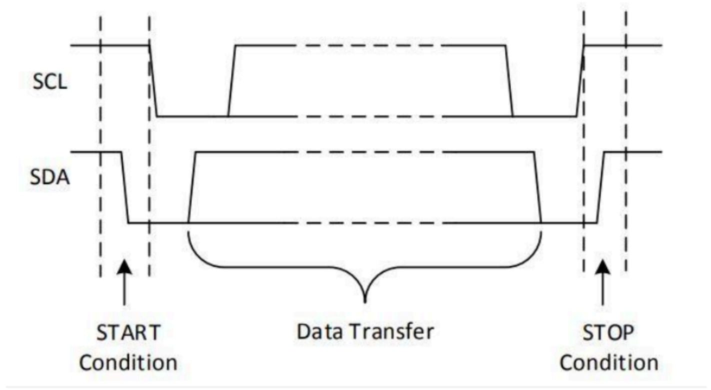
## I2C Transmission

- Each byte is sent **one bit** a time *MSB first*
  - synchronized to each pulse of SCL
  - **SDA only changes when SCL is low** (ensure SDA is stable on both edges of SCL)



## I2C Start & Stop Bits

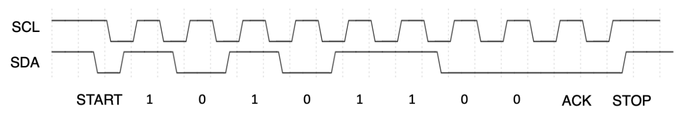
- when bus is idle, SCL & SDA float high (pull-up resistors)
  - to **start** using bus, SDA is pulled low while SCL is high
  - SCL toggles while SDA carries data
  - upon completion, SDA is set high to **stop *while SCL is high***, SCL go back to float high
    - otherwise if SDA changes high while SCL is low, becomes indistinguishable from normal transmission



## I2C ACK & NACK bits

- To tell data was received correctly…
- After each byte, TX and RX swap roles for 1 bit
  - RX sends 1 bit to TX
    - 0 = byte receive (ACK)
    - 1 = byte NOT received (NACK)
  - ACK/NACK uses the SDA line

- Earlier example, now shown with START/STOP & ACK bits.



- A single read or write consists of multiple bytes.

## I2C Register Write

- E.g., Controller writes to a single register on the device.

| Type | START | Device ID | $R/\overline{W}$ | ACK | Address | ACK | Data | ACK | STOP |
|---|---|---|---|---|---|---|---|---|---|
| Bits | 1 | 7 | 1 | 1 | 8 | 1 | 8 | 1 | 1 |

| Who holds SDA? | |
|---|---|
| Controller | ☐ |
| Device | ▨ |

1. Controller begins by ‘sending START’ (i.e., pulling SDA low).
2. First byte contains 7-bit device ID and 1-bit to indicate read or write.
3. Device with that ID responds with ACK.
4. Each byte is transferred followed by ACK/NACK.
5. Controller completes transmission by sending STOP.

## I2C Register Read

- A read requires two ‘transmissions’
  - First transmission: send the address to read to the device

| START | Device ID | $R/\overline{W}$ | ACK | Address | ACK | STOP |
|---|---|---|---|---|---|---|

  - Second transmission: get the data from the device.

| START | Device ID | $R/\overline{W}$ | ACK | Data | NACK | STOP |
|---|---|---|---|---|---|---|

| Who holds SDA? | |
|---|---|
| Controller | ☐ |
| Device | ▨ |

- Controller can also ‘chain’ these into a single transmission

| START | Device ID | $R/\overline{W}$ | ACK | Data | ACK | START | Device ID | $R/\overline{W}$ | ACK | Data | NACK | STOP |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

- In all cases, STOP always ends transmission.

## NACK

- Despite the name, receiving a NACK is not always bad.

- The meaning of the NACK depends on the context
  - NACK after device ID -> no peripheral had that ID. ❌
  - NACK after write data -> device could not do that. ❌
  - NACK after read data -> controller does not want any more data from device. ✓

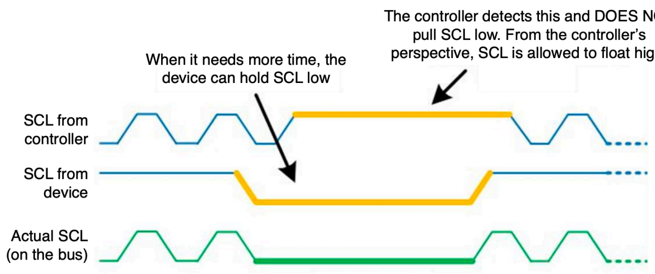
## I2C Burst Transfer

- can send as many bytes in a single transmission (1 START 1 STOP)
  - can mix reading and writing multiple locations in a single transmission with chaining
  - but no one else can use the bus...

## I2C Clock Stretching

- possible but not typical
- how
  - controller pulls SCL low (controller controls Clk frequency)
  - when controller releases SCL, if device want to stretch, device will keep/pull SCL low
  - moment where device catches and holds the clock
  - controller keeps trying to release SCL (let it go high) but device is holding it low

**I2C – clock stretching example**


The controller detects this and DOES NOT pull SCL low. From the controller's perspective, SCL is allowed to float high.
When it needs more time, the device can hold SCL low
SCL from controller
SCL from device
Actual SCL (on the bus)

## I2C Arbitration

- when multiple controllers want to use this bus...
  - controllers constantly monitor SCL and SDA to check the bus is free before transmitting - if busy, wait for stop bit before transmitting
- when 2 controllers pull SDA low at the **exact same time**
  - each controller pulls SDA low and checks to see if it went low

> I2C requires that if a master in a multi-master environment transmits a high, but see's that the line is low (another device is pulling it down), to halt communications because another device is using the bus.

- if low, that controller has bus control
  - otherwise, controller switches to device mode until it sees a STOP bit, and will try to get control again
- (If both happens to transmit the same bit) The arbitration continues bit by bit until one master notices a discrepancy and backs off. The device sending the lowest address value will win because a 0 bit (low) overrides a 1 bit (high) on the **wired-AND bus**

## Derivative Protocols

- System Management Bus (SMBus)
  - used on computers for power related chips i.e. battery
- Improved Inter-Integrated Circuit (I3C)
  - backward compatible with I2C
  - speed 12.5MHz
  - supports **dynamic address allocation**

## TI Manual

https://www.ti.com/lit/an/slva704/slva704.pdf