i2C communication Protocol

i2C : inter integrated circuit

used in : eg: OLED displays, barometric pressure sensors, gyroscope / accelerometer modules.

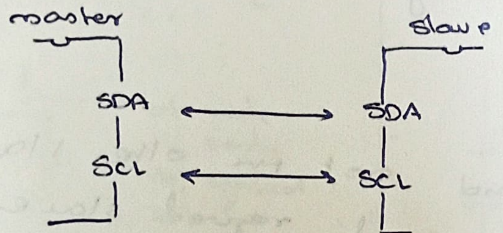
basics:

we can connect multiple masters to a single slave

or

single master to multiple slaves

' useful when more than one microcontroller logging data to a single memory card / display text to single LCD.



SDA : (serial data) : line for master and slave to send data.

SCL : (serial clock) : line that carries clock signal.

→ i2c is a serial communication protocol, so data is transferred bit by bit along a single SDA line.

→ i2c is synchronous: o/p bits are synchronized to sampling of bits by a clock signal shared b/w master and the slave. clock signal is always controlled by the master.

- wires used : 2
- max. speed : standard = 100kbps
  - Fast = 400kbps
  - High speed = 3.4 Mbps
  - ultra fast = 5 Mbps
- · synchronous
- · Serial
- max no. of masters : unlimited
- max no. of slaves : 1008

How i2c works?

- data transfered in messages.
- messages broken up into frames
- each data has an address frame that contains binary address of slave and one or more frames containing data being transmitted.
- Also includes start, stop conditions, read, write bits, Ack / NACK bits b/w each frame.

Message format:

Start condition	7 or 10 bits Address frame	Read/ write bit	ACK/ NACK bit	8 bits data frame	ACK/ NACK bit	Stop condition.

Start condition: SDA line high to low voltage before SCL line high to low voltage
Stop condition: SDA line low to high voltage after SCL line low to high voltage
Address frame: 7/10 bit sequence unique to each slave, that identifies the slave when master wants to talk to it.
Read/Write bit: low: write (data from master to slave)
high: read (data demanded by master from slave)
ACK/NACK bit: each frame in a message is followed by a acknowledge/no bit if an address/data frame was successfully recieved, an ACK bit is returned to sender from recieving device.

Addressing
—
how Particular slaves know data is sent to it and not the other slaves?
each slave have unique address frame. master send required slaves address to all connected slaves and slave which matches send an acknowledge bit back to master (by a low voltage). if address doesn't match, slave does nothing and SDA line remains high.

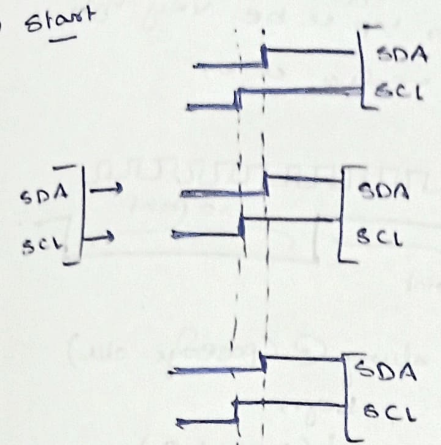
Read/Write bit
—
Address frame includes a single bit at the end that informs the slave whether the master wants to write data to it / recieve data from it.
bit : low : master send data to slave.
high : master request data from slave.

Data frame
—
. After acknowledgment bit from slave, first data frame is ready to be sent.
. Always 8 bit long and sent with most significant bit first.
. immediately followed by ACK/NACK. to verify recieved successfully.
. ACK bit recieved by master/slave (depending on who is sending data)

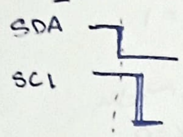
Stop:
—
After all data frames sent, master can send a stop condition. which is SDA low to high after SCL is remaining high.

steps:

1) Start



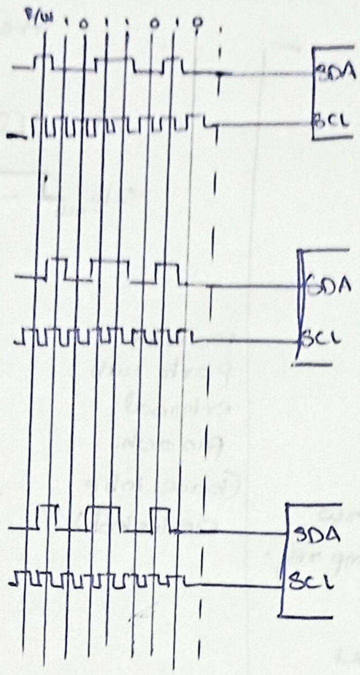
Normally



but while sending its symbolical both first

just think slowly when slave recieve it high initially then it becomes low ~~[illegible]~~ while scl remain high.

2) master send 7/10 bit address and also R/W bit

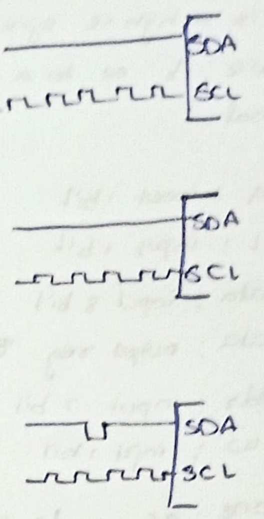


what master sending is 0101101 R/w

but this is how slave get: R/W 1 0 1 1 0 1 0

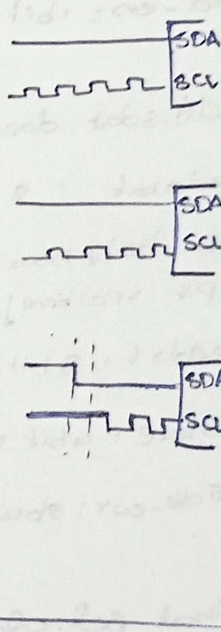
• each slave compares Address

3) if address matches slave returns ACK by pulling SDA low for one bit

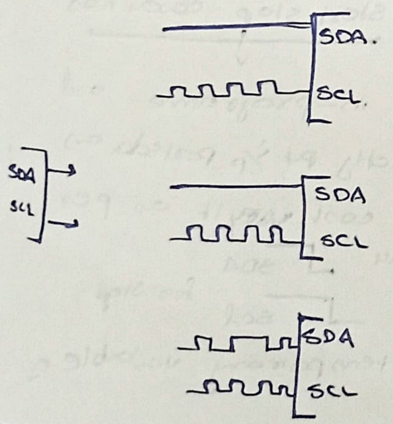


6) Stop ~~SD~~.

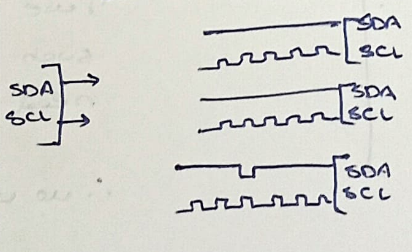
SCL low to high ~~[illegible]~~ before switching SDA high.



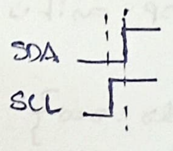
4) master sends/recieves data frame



5) After data frame transfered receiving device returns ACK.



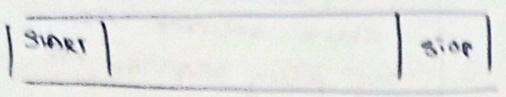
ideally:



Advantages:
- only uses 2 wires.
- Support multiple master/slave.
- ACK/NACK bit gives confirmation that each frame is transfered successfully.
- less complicated hardware than UART
- widely used protocol

Disadvantages:
- slower data transfer rate (comparing SPI: 60Mbps)
- size of data frame 8 bit
- complicated hardware than SPI

code properties:



clk: 1bit input (to FPGA)
This is a high ck signal and we reduce it ck to a sclk_wr signal.

SDA : inout 1bit
SCL : input 1bit
wdata : input 8 bit //write data
rdata : output reg 8bit //read data
addr : input 7 bit } → addrt = addr : rw
rw : input 1 bit 8 bit temp reg.
done : o/p reg to show done
ack : 1bit input //acknowledgement
rst : 1bit input //reset

module ports with external pinouts. (have to be enlisted)

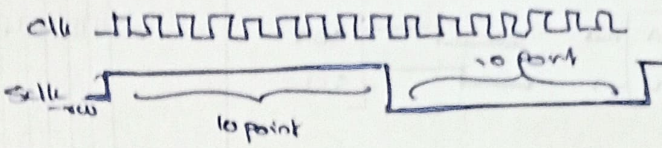
sda_en : 1bit – [low: sda = Z
high: sda = data passed.

sclt, sdat, donet : 1bit
rdatat : 8 bit reg
[wdata doesn't have to be temp. as it is input realtime]
addrt : 8 bit reg {addr : rw}
state : 4bit reg (13 state FSM)
sclk_wr : slower clock for FPGA fn.

registers used in code

Q) how: slower clock?

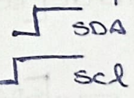
@ consider i/p $\overset{clk}{}$ ck be very high then we reduce ck by



code:
```
always @ (posedge clk)
    begin
      if (count < 9)
          count + = 1
      else
        sclk = ~sclk
        count <= 0
    end
```

Q) why we need temporary sdat, sclt etc?

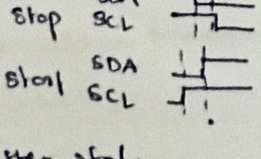
sda, scl etc. carry both address, datas and start stop conditions

These are in-programe and if we directly put i/p portsda as such we can't vary it as per need. like


for stop.
∴ we use temporary variables

Q) Final scl, sda statement:

```
assign scl = ((state == wstart) || (state == wstop) || (state == rstop)) ? sclt : sclk_wr;
assign sda = (sda_en == 1'b1) ? sdat : 1'bz;
```



This is programmed by us through code if rdata = 0 we will remain in this sda, scl value. This can't be achieved by extend signal.

during other state scl = sclk_wr.

→ So we put values in sdat, sclt and assign them to sda, scl.