ODHS Taskphase-1

shashwat harsh|220953028|9708777568|shashwatharsh03@gmail.com

Async and sync data transfers

Resource :- https://www.ibm.com/docs/en/aix/7.3?topic=synchronization-synchronous-transmission

William Stalling – data and communication edition 10e – https://nibmehub.com/opac-service/pdf/read/Data%20and%20computer%20communications%20by%20Stallings-%20William-compressed.pdf

https://www.computer.org/publications/tech-news/trends/synchronous-asynchronous-data-transmission

Asynchronous data transfers

In this type of data transfer, the data is broken into small pieces and a start and stop bit sequence is added to such data frames to indicate the starting and ending of each such block of data.

The receiver only starts reading the data when the start bit sequence is received and stops when the stop bit sequence is received and remains idle otherwise.

Thus the timing of each such data block does not matter. And doesn't need a clock signal.

Each of these block of data generally has upto 8 bits of data , only 1 parity bit and start and stop bit sequence making a maximum of 11 bits.

To mitigate the errors due to sampling, each bit is sampled at the middle of each bit time.

Advantages

No need for synchronisation

Easy to implement

Cost effective – no need for expensive systems to parse through higher data stream.

Different bitrates can be used

Start and resume data transfer as and when needed.

Disadvantages

Low speeds

Waste of bandwidth

Need for many redundancy bits (bits other than data bits)

MAJOR- false recognition of start and stop sequence.

Timing errors.

Synchronous data transfers

In this type of data transfer, the data is send in a single stream in large blocks of data between the sender and receiver

Each block of data called frame has preamble and postamble bit pattern such as a SYN or synchronous idle character attached to it.

These SYN characters are used to sync up both sides of the connection i.e sender and receiver and then the data transfer begins.

It may also have a clock pulse to sync up the transmitter and receiver. This may be done by sending a separate clock pulse or may be done by incorporating such a clock in the encoding itself eg:- Manchester encoding, differential Manchester encoding, bi phase encoding etc.

A CRC is also added to every frame for error correction.

About the SYN character

It is a ASCII control character with integer ASCII code 22. It is used for synchronous connection in data transfers where no other characters other than the data is being transmitted. i.e no header etc.

Resource :-

https://www.asciihex.com/character/control/22/0x16/syn-synchronous-idle

https://www.ascii-code.com/character/%E2%90%96

advantages

higher speed

no need for a local buffer or storage of bits to create a block i.e frame from a sequence of bits

reduced or no timing errors

realtime communication can be established

disadvantages

requires costly hardware

requires synchronisation which gets harder and costly with distance

accuracy in parsing of data is dependent on recievers capacity.

Parallel and serial data transfers

Serial data transfer

In this system the data block is sent from transmitter to receiver in a single stream, bit by bit starting from LSB to MSB of each data frame.

A single or two data connections are needed.

Reduces costs and complexity.

Parallel data transfer

Bits in a block of data is sent 'in parallel' to each other rather than bit by bit.

i.e. one single data frame is sent to the receiver , over multiple data connections ,(same as that of number of bits in that frame.).

Thus, data is transferred frame wise i.e. one block of data is transferred after another rather than bit by bit.

Is faster but requires needs syncing of bits being received.

Resource:- https://www.contec.com/support/basic-knowledge/daq-control/serial-communicatin/

https://massive.io/file-transfer/serial-vs-parallel-write-speed/

https://www.ibm.com/docs/en/aix/7.3?topic=communication-serial-parameters

I2C and SPI communication protocol (in detail)

I2C protocol

Basic topology/physical layer

Inter Integrated circuit protocol

It works on the master-slave system of communication.

It has two lines:-

1. SDA – Serial data line , bidirectional , to allow transfer of data from controller to slave system
2. SCL – Serial Clock Signal, unidirectional.

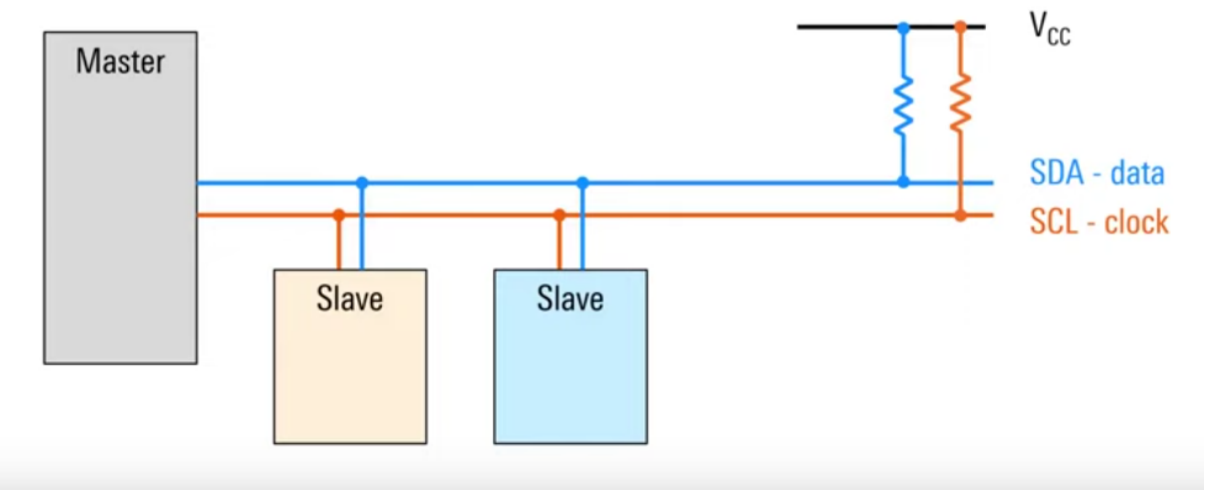
Both the lines are connected to Vcc with ‘pull up’ resistor of ~Kohm resistances

The idle state of both lines is high.

Slave devices can be added in any position at anytime.

One such I2C system has multiple slave devices and may also have multiple controllers.

However number of slaves dependent on the available addresses (theoretical maximum of 2^7).



Open drain

An I2C device is connected to ground via an NMOS in an open drain connection.

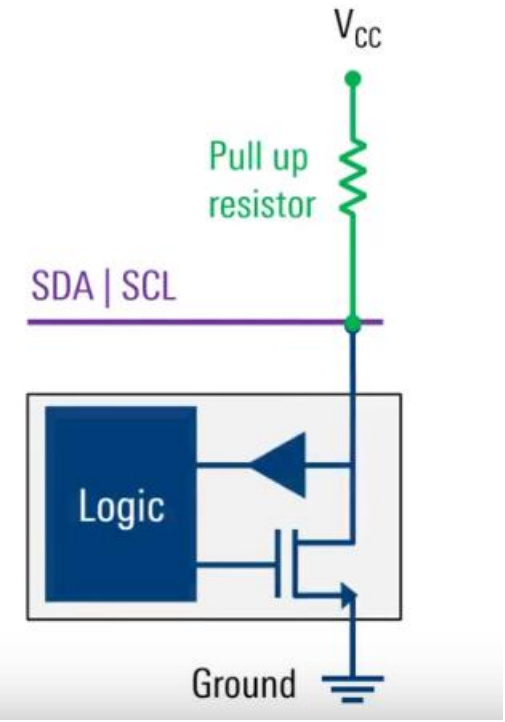
When the NMOS is on it pulls, the SDA/SCL to ground .

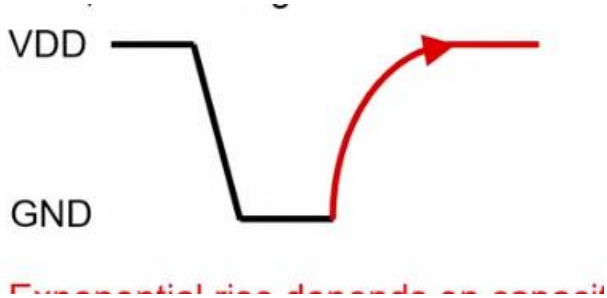
When the NMOS is OFF the pull up resistor brings SDA/SCL to Vcc.

The pull down is generally faster as NMOS pulls charge from SDA/SCL lines. While pull up is exponential rise dependent on resistor and capacitance of bus lines.

Low resistance= faster communication, more power

High resistance= slower communication, less power





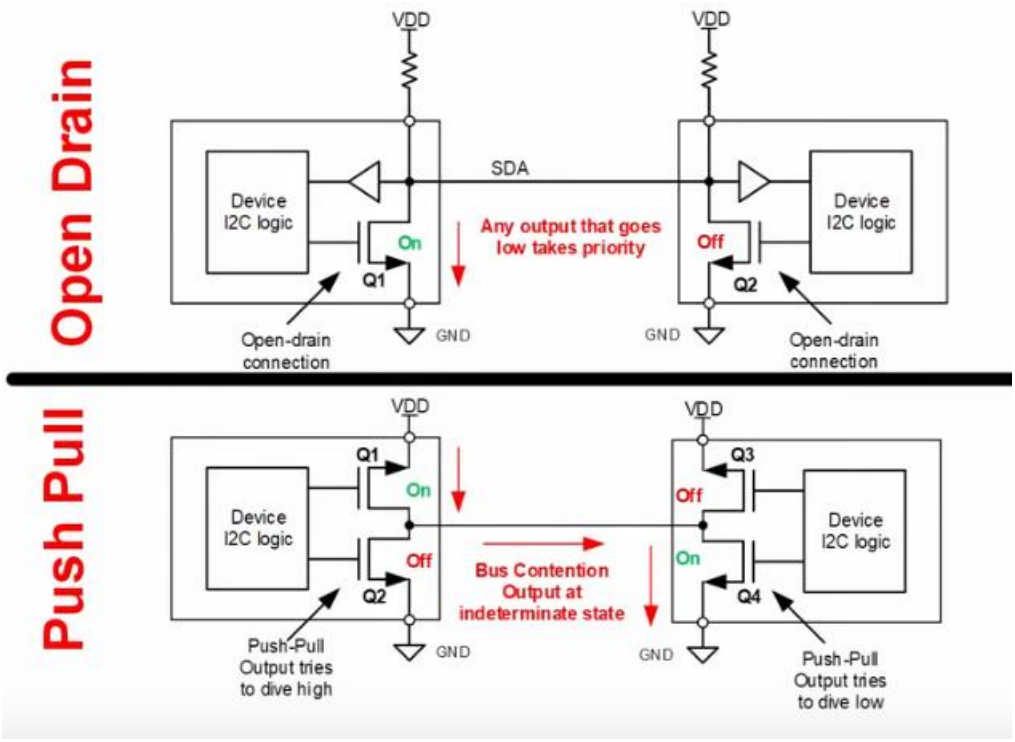
Open drain is better than push-pull config as :-

In open drain connection,devices' outputs SDA/SCL can be connected together as any output that goes low pulls the whole output low. This is a "wired-OR" connection.

In a Push-pull config, the devices has complimentary NMOS and PMOS that push-pull high or low .

When one output is high and another is low, the BUS will have contention leading to the output being at an intermediate state.

The PMOS of one NMOS is connected to the PMOS of another via output. This would lead to a low impeadance path for current to flow through thereby damaging the device.



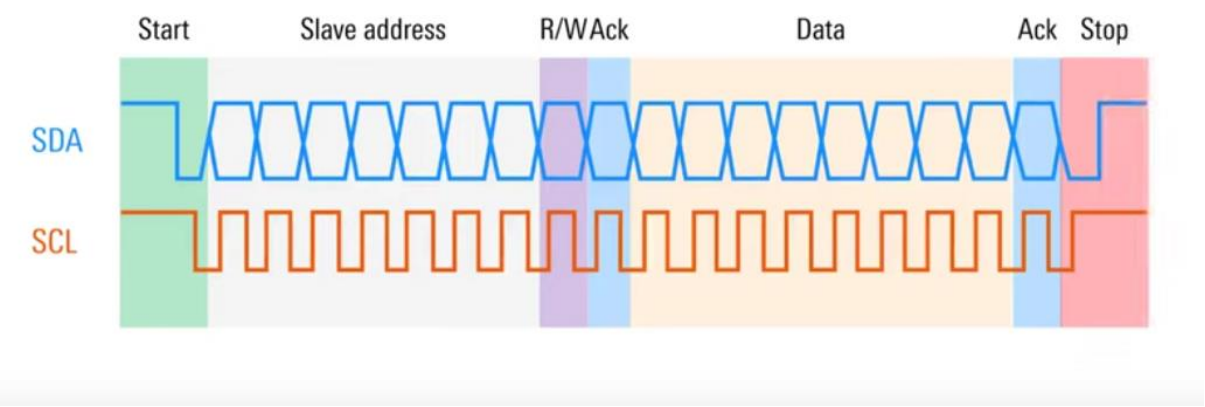
I2C Frames

Communications between the controller and devices is done in frames of data.

Half duplex

SDA and SCL lines are interpreted as 1 or 0 corresponding to high and low.

SDA doesn't change when SCL is rising or falling. SDA only transitions when SCL is low. This is to prevent misinterpretation of signal with start or stop condition.



Communication starts with a start condition.

Start condition-

The controller pulls down the SDA and then SCL to low to indicate and take control of the bus and all the other devices on the bus stops communicating. This reduces the risk of two nodes talking on the bus at the same time.

The Master with the control starts sending clock single and starts the communication.

Slave address byte-
This is followed by the (generally 7-bit upto 10 bit) slave address starting from MSB to LSB with which the master wants to read/write to.

Each node on the bus has a unique address (fixed or somewhat changeable).

The address bits are followed by the Read write bit . 0 = master wants to write . 1= master wants to read. This bit is generally considered part of the address bits making it 8 bits ie one byte long.

ACK bit- after every byte, an acknowledgment bit is sent by the receiver(either slave or master depending on whether reading or writing). 0= ACK , 1=NACK(negative ack).

If the receiver doesn't sends the ACK bit, ie lack of response , it is considered NACK (as idle SDA is high).

ACK sent after address byte by slave to indicate it exists on the bus and is ready to read/write.

ACK sent after data byte confirms data is received.

Data Byte
Sent in 8 bits (MSB to LSB) followed by ACK bit.

Multiple data bytes might be sent in a single i2c frame concatenated one after another with ACK bit for each byte in between.

The first data byte can be the register address where the data is to be written/ read.

Stop condition

Once the communication is done the SCL goes high followed by SDA going high and both remain high. This indicates that the master is giving up control.

References:-

https://www.ti.com/lit/an/slva704/slva704.pdf?ts=1698716072997&ref_url=https%253A%252F%252Fwww.google.ca%252F

https://www.youtube.com/watch?v=j9yx8LOslng

https://web.archive.org/web/20210813122132/https://www.nxp.com/docs/en/user-guide/UM10204.pdf

https://www.youtube.com/watch?v=CAvawEcxoPU

SPI Communication Protocol

SPI physical layer

Has 4 wires

Has only one master but can have multiple slave nodes/devices.

Supports full duplex communication

Loosely defined i.e. there is no specified frame formats.

Generally used for smarter devices (such as controllers) to talk to less smart devices(ADC)

The 4 wires are:

Chip Select/ Slave Select (CS/SS):-

It's a active low connection

It is pulled low by master to select a slave for communication

It is kept low until the communication is in process and then allowed to return to idle high to indicate connection termination.

Simpler way to address targets. Unlike I2C

Single or multiple CS lines can be used to select multiple devices

Serial Clock (SCLK)

Clock is generated by the master only.

Faster speed(in MHz) thus faster sampling and communication.

Clock can be idle low or idle high and data can be sampled at rising or falling edge.

Based on idle state and sampling timing, there are 4 modes of SPI

Master Out Slave In (MOSI)

On this wire the Master sends data that is to be read/ written onto the selected slave

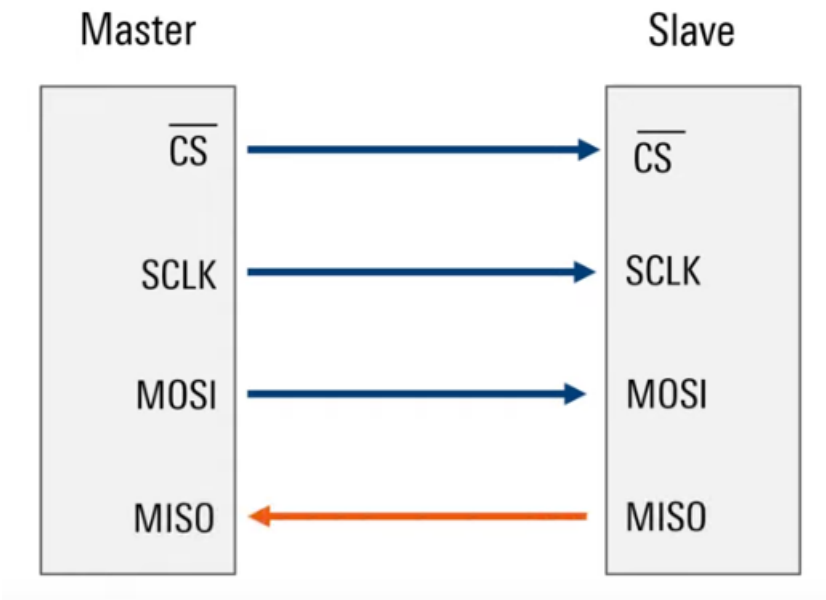
Data is sent either MSB or LSB first based on implementation

Multiple bytes can be sent sequentially to the Slave as the CS is held low.

Master In Slave Out (MISO)

Used to send data from Slave to Master

As slave doesn't generate its own Clock signal, the Master must know in advance or by query to the slave.



Clock polarity and clock phase

Based on the clock polarity (i.e. idle low or idle high) and clock phase (rising or falling edge / leading or trailing edge) the SPI has 4 modes.

Clock polarity is selected for by CPOL bit.

If CPOL= 0 then idle low, active high (non inverted)

If CPOL=1 then idle high, active low (inverted)

Clock phase is checked by CPHA bit

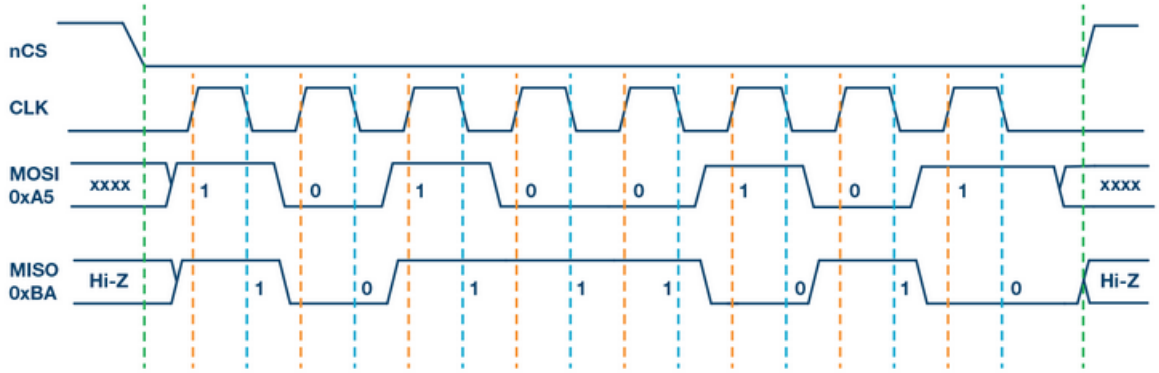
If CPHA=0 then leading edge is used to sample the data

If CPHA=1 then trailing edge

This creates the 4 modes as follows :-

SPI Mode	CPOL	CPHA	Clock Polarity in Idle State	Clock Phase Used to Sample and/or Shift the Data
0	0	0	Logic low	Data sampled on rising edge and shifted out on the falling edge
1	0	1	Logic low	Data sampled on the falling edge and shifted out on the rising edge
2	1	0	Logic high	Data sampled on the falling edge and shifted out on the rising edge
3	1	1	Logic high	Data sampled on the rising edge and shifted out on the falling edge

4 modes in detail:-



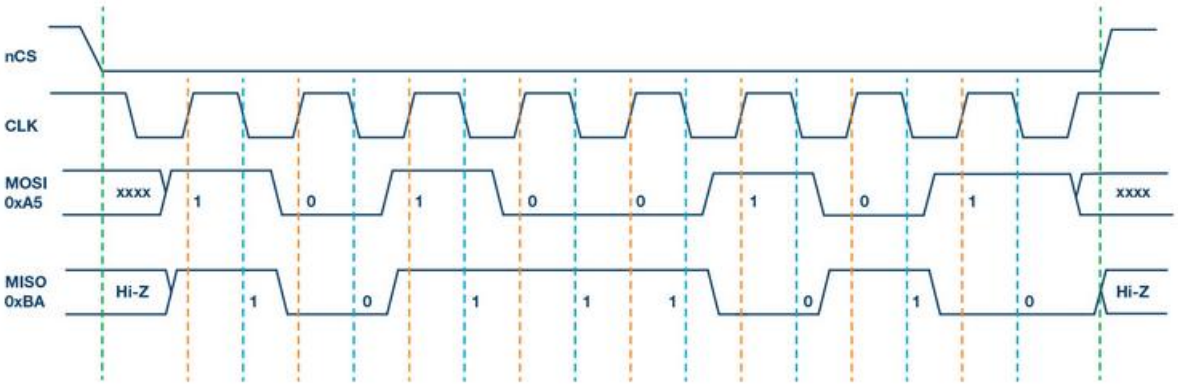
In the above waveform, clock is active low thus CPOL=0

Now if data is to be sampled on rising edge(orange line) => leading edge thus CPHA=0

Hence this will be SPI mode 0

Now if the data is to be sampled on falling edge (blue line) => trailing edge this CPHA=1

Hence this will be SPI mode 1



In the above waveform, clock is active low thus CPOL=1

Now if data is to be sampled on rising edge(orange line) => leading edge thus CPHA=0

Hence this will be SPI mode 2

Now if the data is to be sampled on falling edge (blue line) => trailing edge this CPHA=1

Hence this will be SPI mode 3

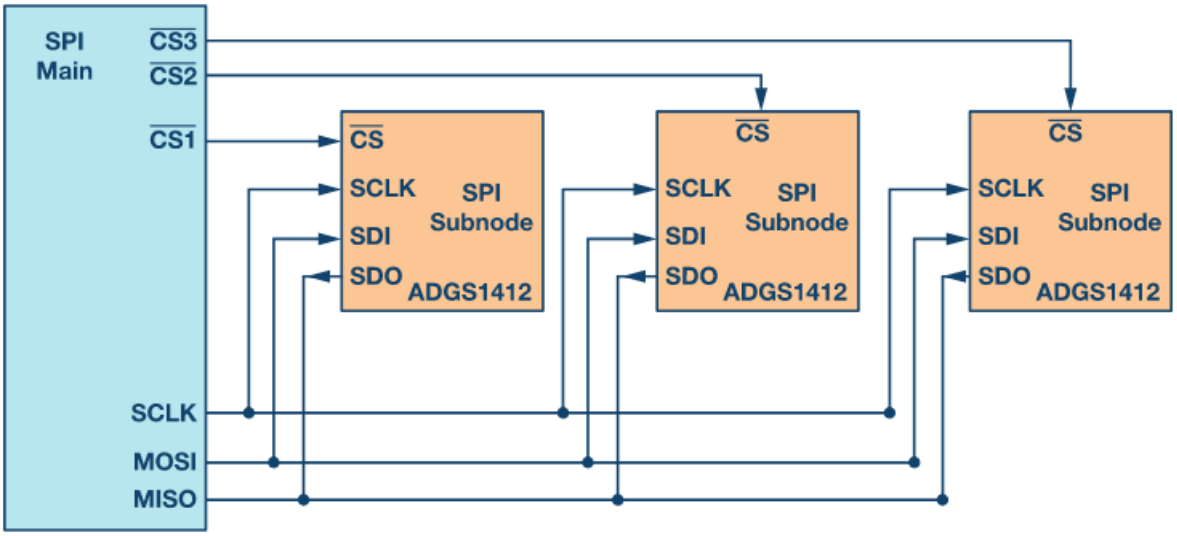
Multi Slave configurations

Regular SPI mode

Here SCLK, MOSI and MISO are shared among slaves but the CS lines are different i.e. the master has separate pins for each slave connected to it.

This is a simple method but cant be scaled up.

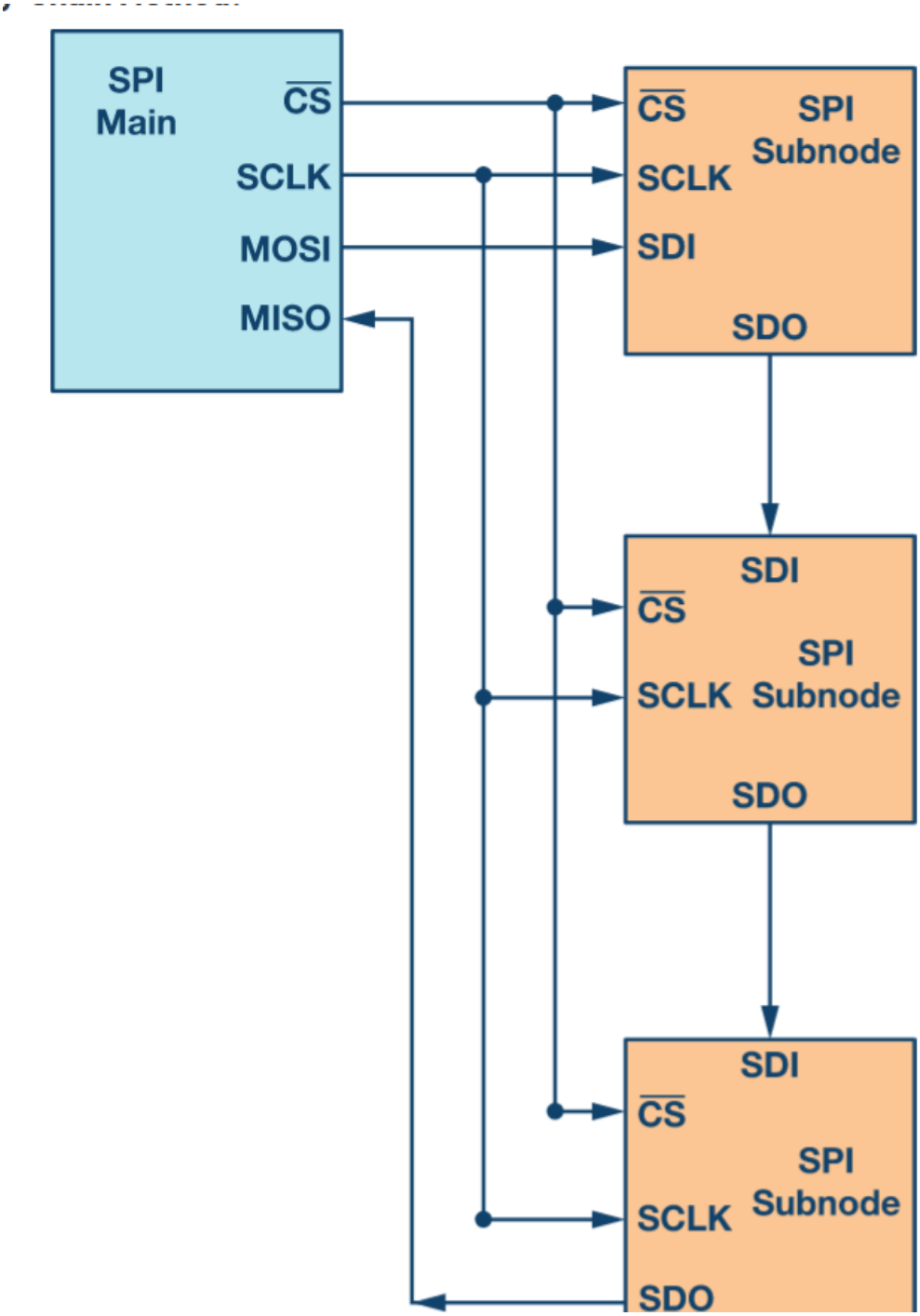
Also if multiple slaves are enabled, this can cause the data being transmitted on MISO line to be corrupted. And no way to identify which slave is transmitting what.



Daisy chain method

In this configuration, the CS and SCLK lines are shared.

The MOSI line of master is connected to the first slave and its MISO line is connected to the second slave and so on until the last slave connects to the MISO line of the Master.



In this config multiple clock pulses are required to forward the data from one slave to another and finally to the Master.

Eg:- consider a 8 bit 3 slave SPI configuration. Thus to transmit one byte of data of first slave 24 clock pulses will be needed i.e. each 8 pulses for 1 byte to transmit it to the next slave.

Resources :-

https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html

https://www.ti.com/lit/ug/sprugp2a/sprugp2a.pdf?ts=1698898813918&ref_url=https%253A%252F%252Fwww.google.co.kr%252F

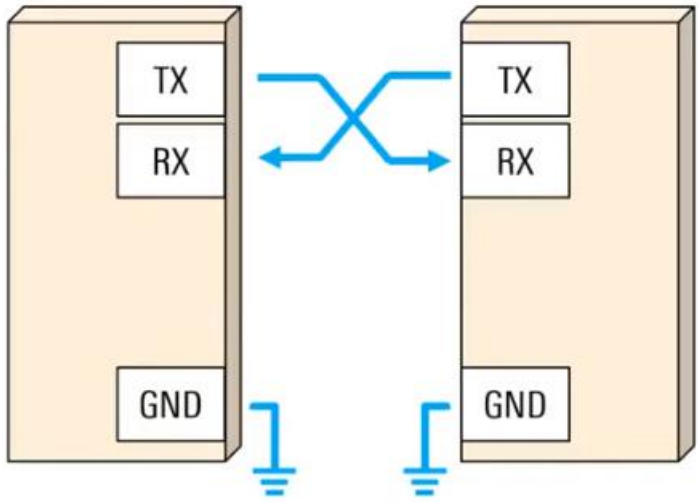
https://www.youtube.com/watch?v=SvB9AWyyRNI

https://www.youtube.com/watch?v=0nVNwozXsIc

UART Protocol

UART= Universal Asynchronous Receiver Transmitter

It only requires 2 wires for communication:- Receive (Rx) and Transmit (Tx)



Data is transmitted in Data Frames

And can be simplex, half duplex or full duplex

It is a low speed , low throughput protocol that is easy to implement due to lack of a clock i.e. asynchronous.

Since it is asynchronous, Baud rate between both devices must be equal and known.

Common baud rate:- 4800,9600,19200,57600,115200 bps

Both devices must use same parameters and frame structures

UART frames consists of :-

Start bits

Data bits

Parity bit(optional)

Stop bits.

At idle state the lines are kept high.

First the start bit is send which is generally just a transition from idle high to low .

This is followed by data bits(based on baud rate) and then optionally a parity bit

To terminate the frame a stop bit is sent. This is either a return to idle high or to remain at the high state for an additional bit time.

In some cases two stop bits are sent to allow receiver to get ready for the next frame.

The parity bit either uses even parity or odd parity which is predetermined.)

Data is sent from LSB to MSB

Below ASCII ‘S’ is being sent S= $(1010011)_2$