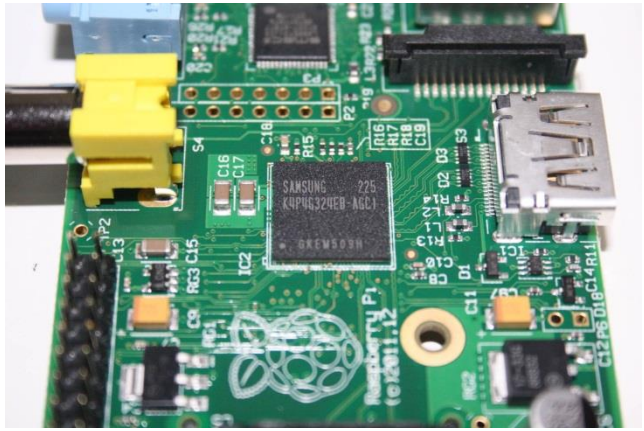
RaspberryPi

Raspberry pi interfacing

Lecture 3: Embedded Communication Protocols



Embedded Communication Protocols Agenda

- Embedded systems communication concepts.
- UART protocol.
- I2C protocol.
- SPI protocol.

Embedded systems communication concepts

- **What is communication in embedded systems ?**
  - It is a way of exchanging data or commands between 2 or more different embedded systems devices.
  - Communication protocols are standards that contains data exchange rules and format between embedded systems.
- **Communication protocols examples in embedded systems**
  - UART.
  - I2C.
  - SPI.
  - CAN.
  - LIN.

Embedded systems communication concepts

- **Bit rate**
  - It is the number of bits that are sent per unit of time usually bits/sec.
- **Baud rate**
  - It is the number of symbols that are sent per unit of time usually, symbol can be combination of any number of bits according to design choice.
  - In case that symbol is only 1 bit, then baud rate will equal the bit rate.

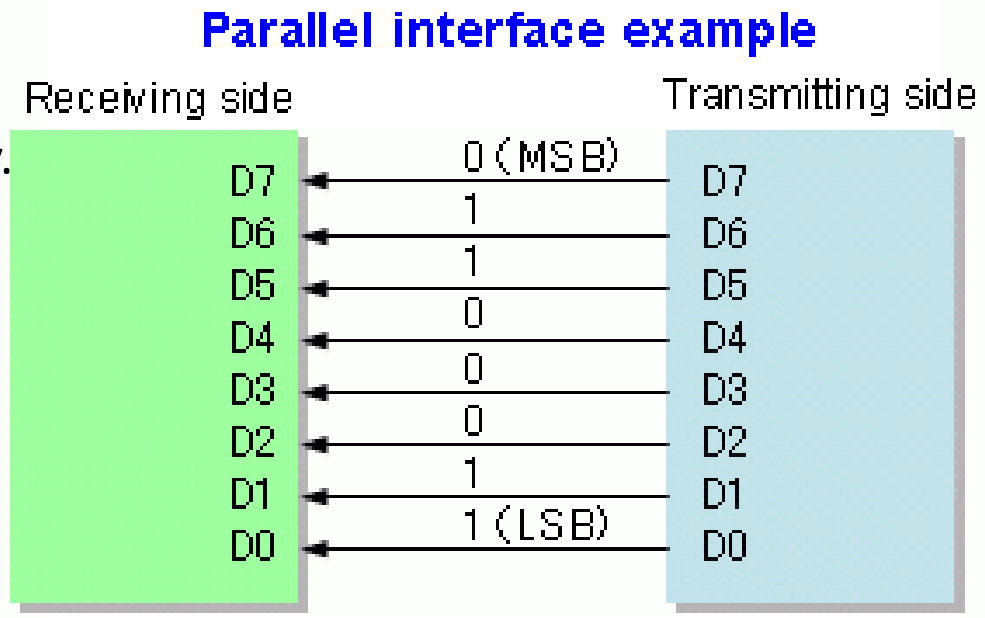
Embedded systems communication concepts

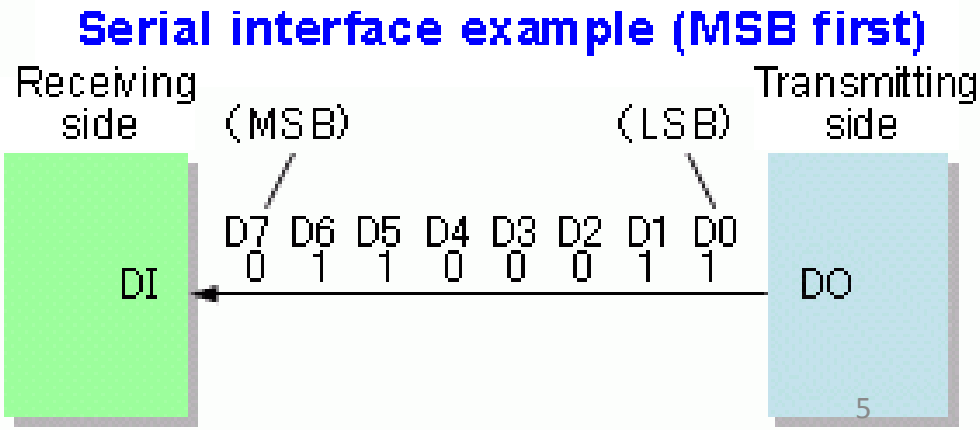
- **Parallel Communication**

  Sending multiple bits simultaneously.

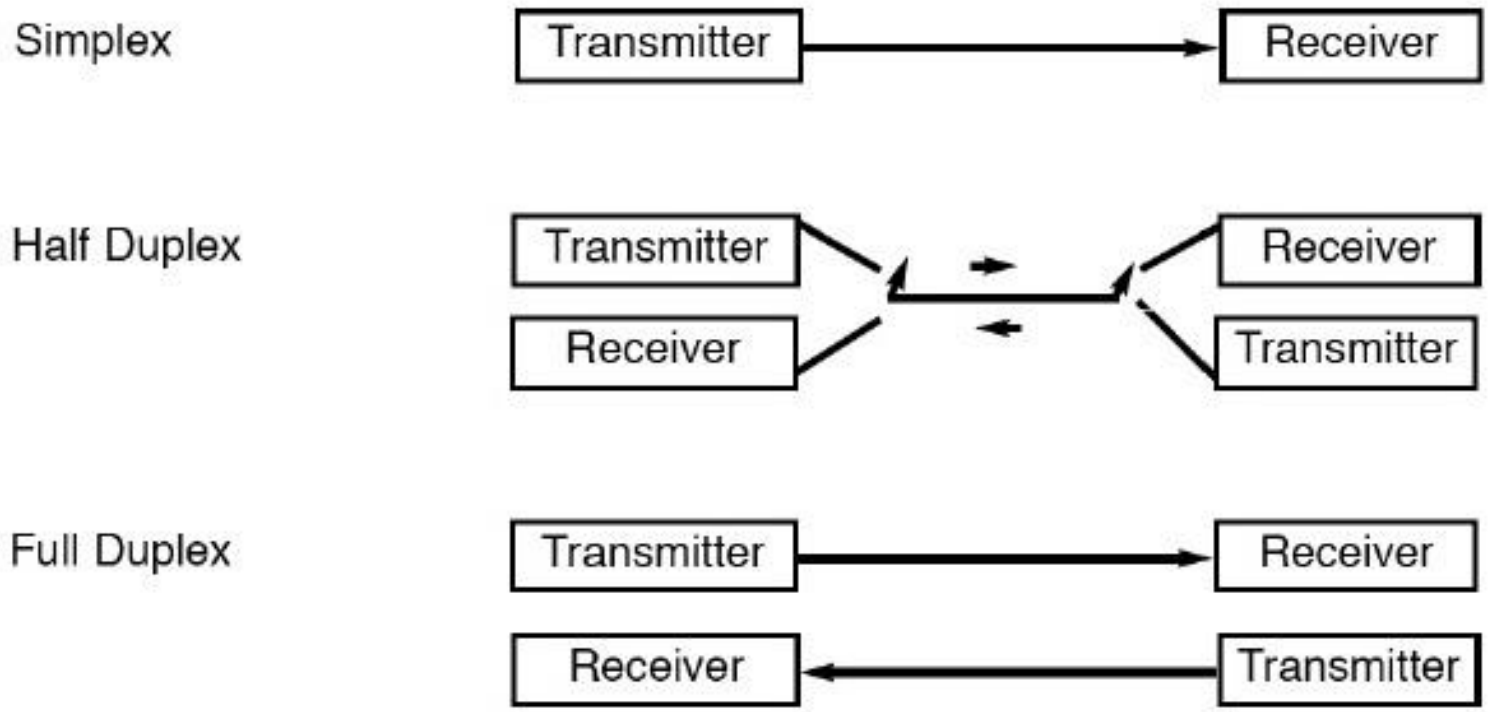
- **Serial communication**

  Sending bit by bit.





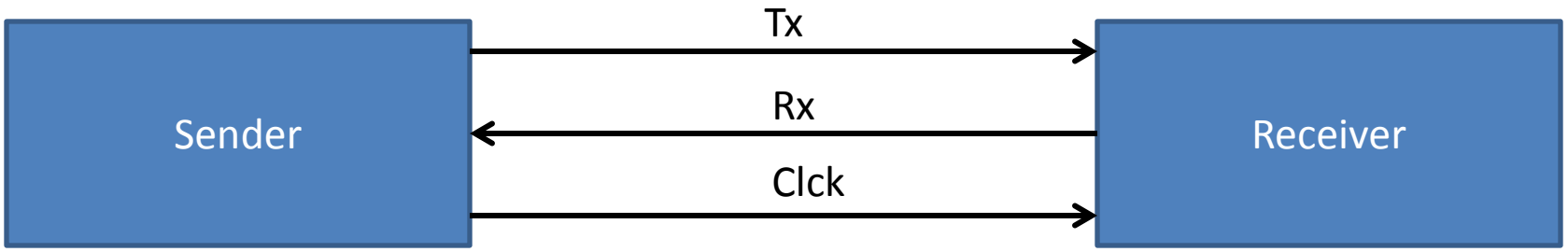
Embedded systems communication concepts



Embedded systems communication concepts

➢ Synchronous serial communication

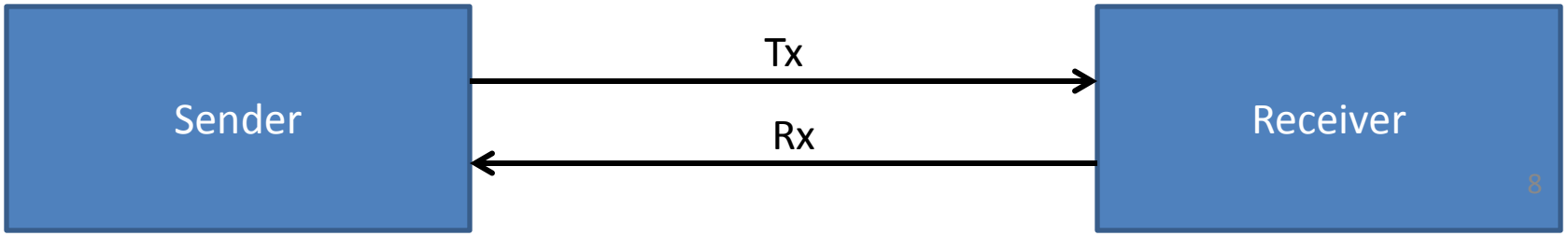
- Synchronous serial communication describes a serial communication protocol in which data is sent in a continuous stream at a constant rate.
- Synchronous communication requires that the clocks in the transmitting and receiving devices are synchronized – running at the same rate – so the receiver can sample the signal at the same time intervals used by the transmitter so that there is an extra wire for clock carrying.



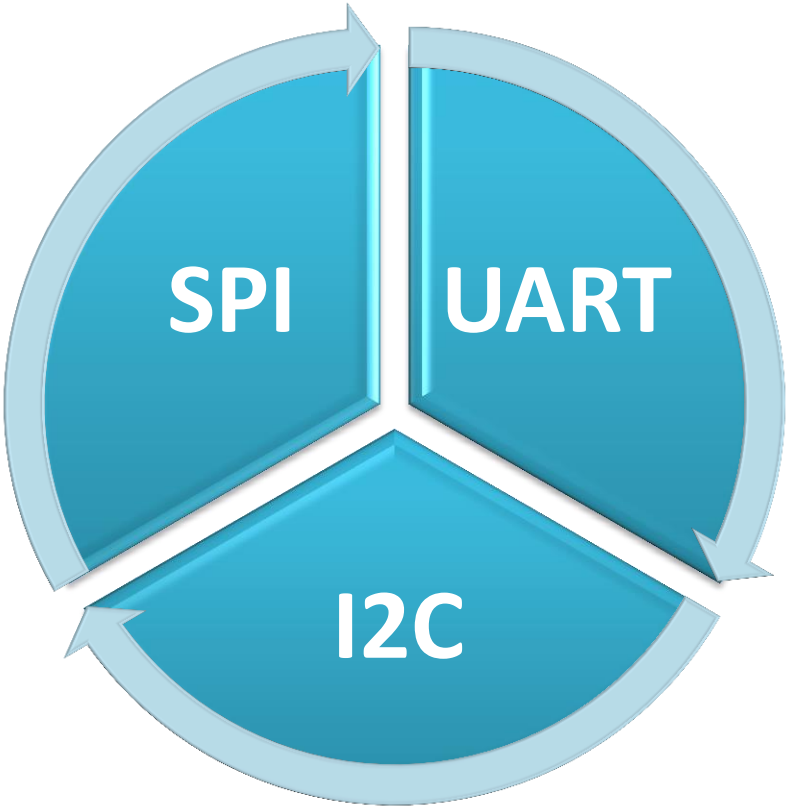
Embedded systems communication concepts

➢ Asynchronous serial communication

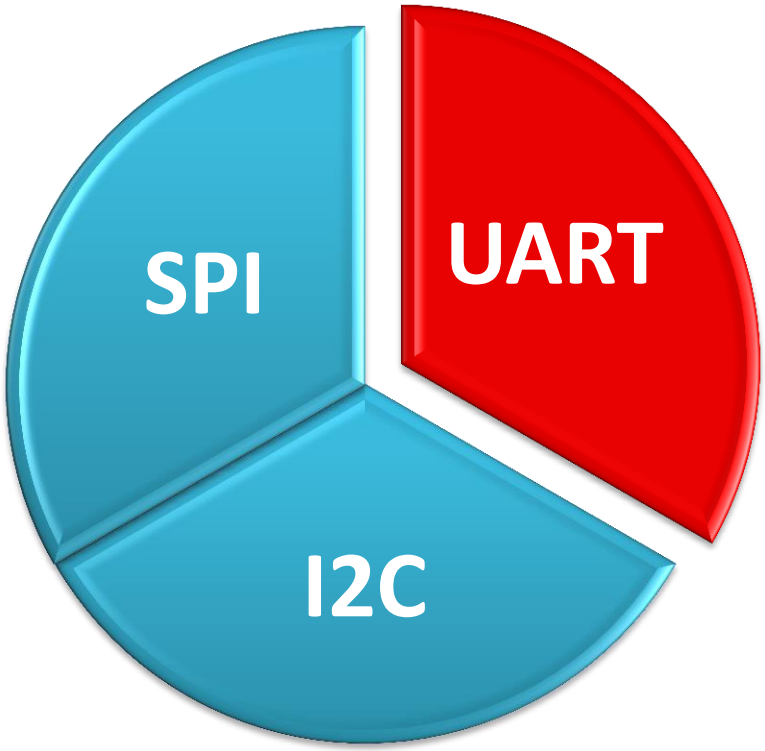
- Asynchronous serial communication is a form serial communication in which the communicating endpoints' interfaces are not continuously synchronized by a common clock signal.
- Instead of a common synchronization signal, the data stream contains synchronization information in form of start and stop signals, before and after each unit of transmission, respectively. The start signal prepares the receiver for arrival of data and the stop signal resets its state to enable triggering of a new sequence.



Protocols examples



UART



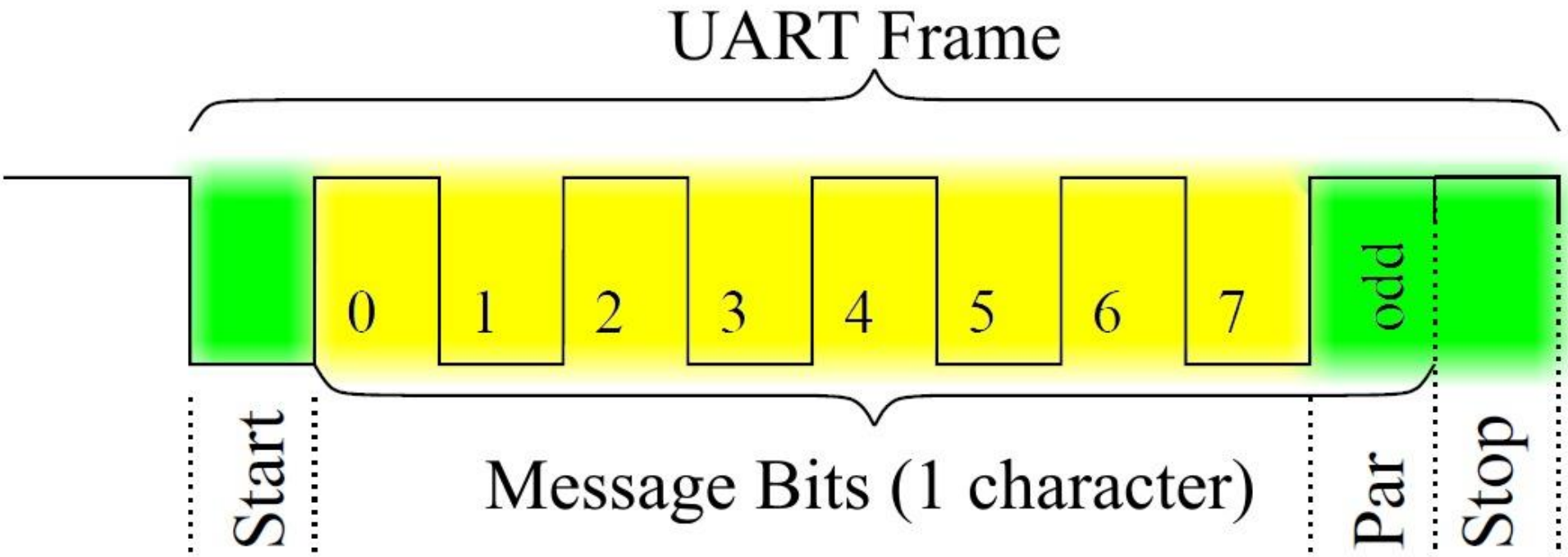
UART

What is UART ?

- UART stands for Universal Asynchronous Receiver Transmitter. There is one wire for transmitting data, and one wire to receive data. A common parameter is the baud rate known as "bps" which stands for bits per second. If a transmitter is configured with 9600bps, then the receiver must be listening on the other end at the same speed.
- UART is a serial communication, so bits must travel on a single wire. If you wish to send a char (8-bits) over UART, the char is enclosed within a start and a stop bit, so to send 8-bits of char data, it would require 2-bit overhead; this 10-bit of information is called a UART frame.

UART

- **UART frame format**



UART

➢ UART frame format

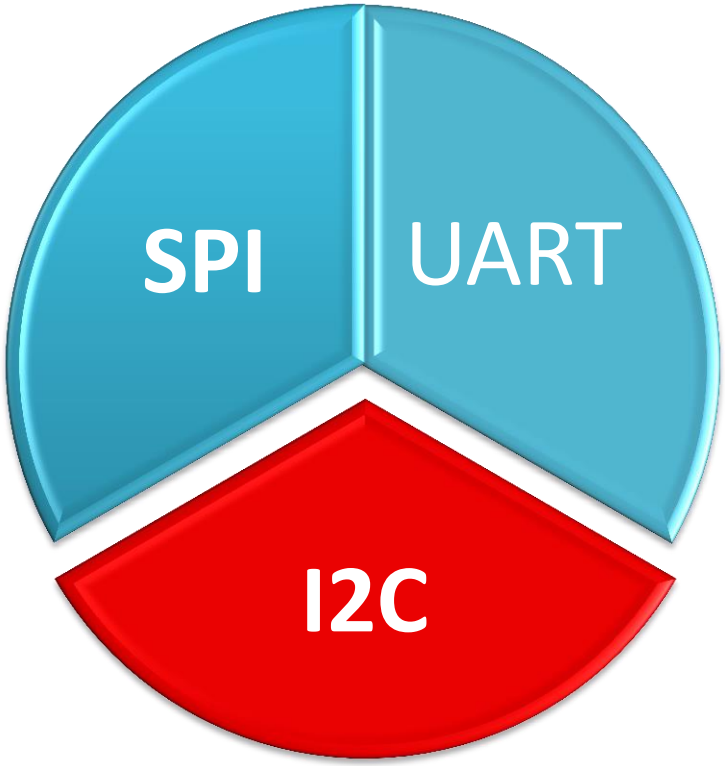
- Start bit: 1 bit indicates the start of a new frame, always logic low.
- Data: 5 to 8 bits of sent data.
- Parity bit: 1 bit for error checking
  - Even parity: clear parity bit if number of 1s sent is even.
  - Odd parity: clear parity bit if number of 1s sent is odd.
- Stop bit: 1 or 2 bits indicate end of frame, always logic high.

UART

➢ UART notes

- UART supports PC communication through RS232.
- UART standard baud rate examples (110, 300, 600, 1200, 4800, 9600, ...).
- UART is a full duplex communication protocol.
- UART is point to point communication protocol, which means communication can be between 2 devices only at the same time.

I2C



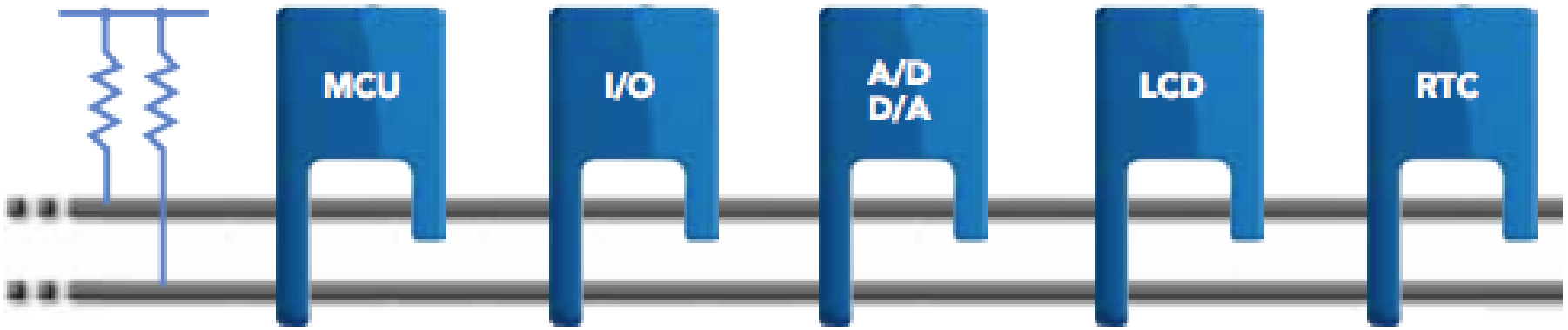
I2C

What is I2C?

- I²C (Inter-Integrated Circuit), is a multi-master, multi-slave, serial computer bus invented by Philips Semiconductor (now NXP Semiconductors). It is typically used for attaching lower-speed peripheral ICs to processors and microcontrollers.
- I2C is a synchronous serial communication using two wires, one wire for data (SDA) and the other wire for clock (SCL).

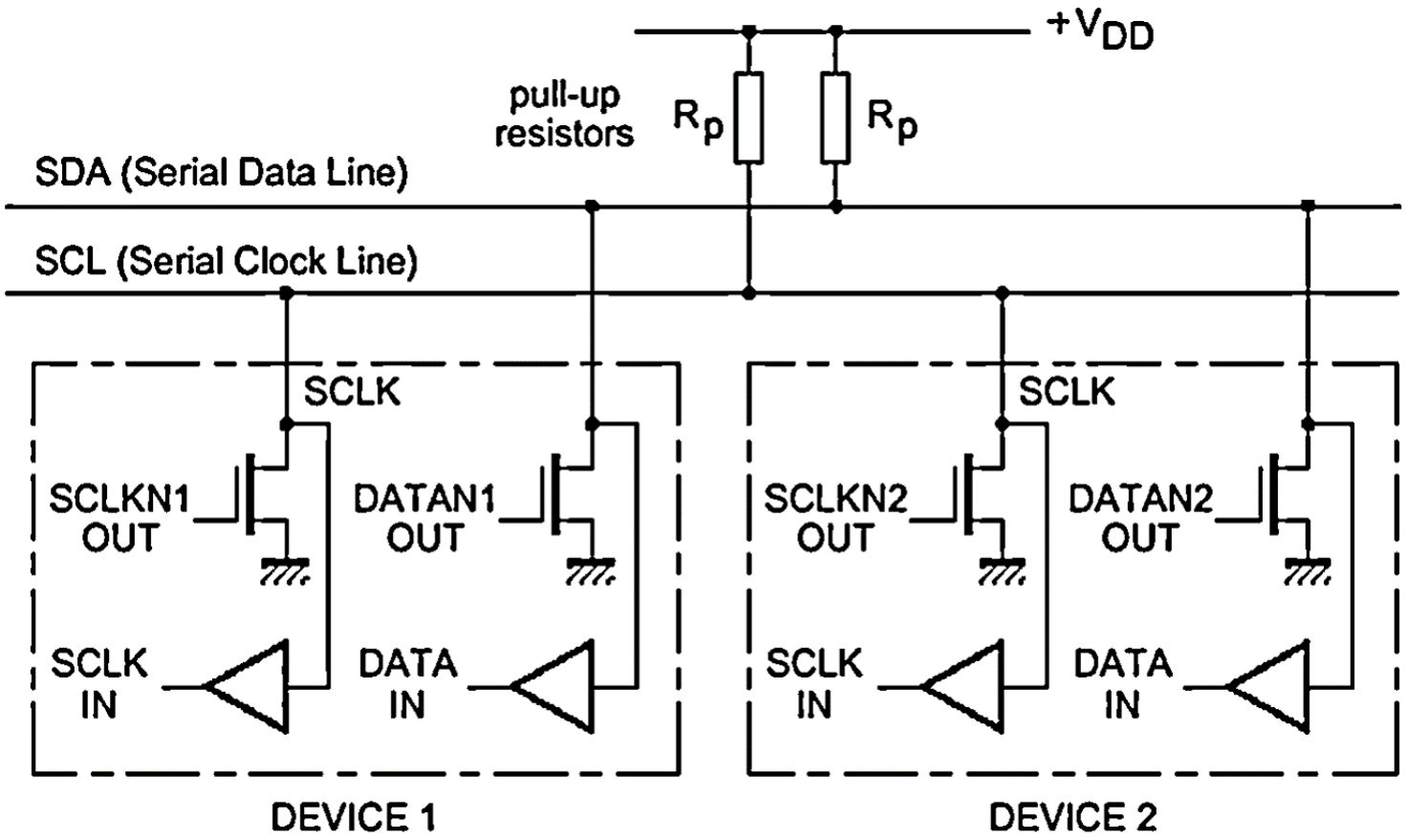
I2C

- **I2C bus**



I2C

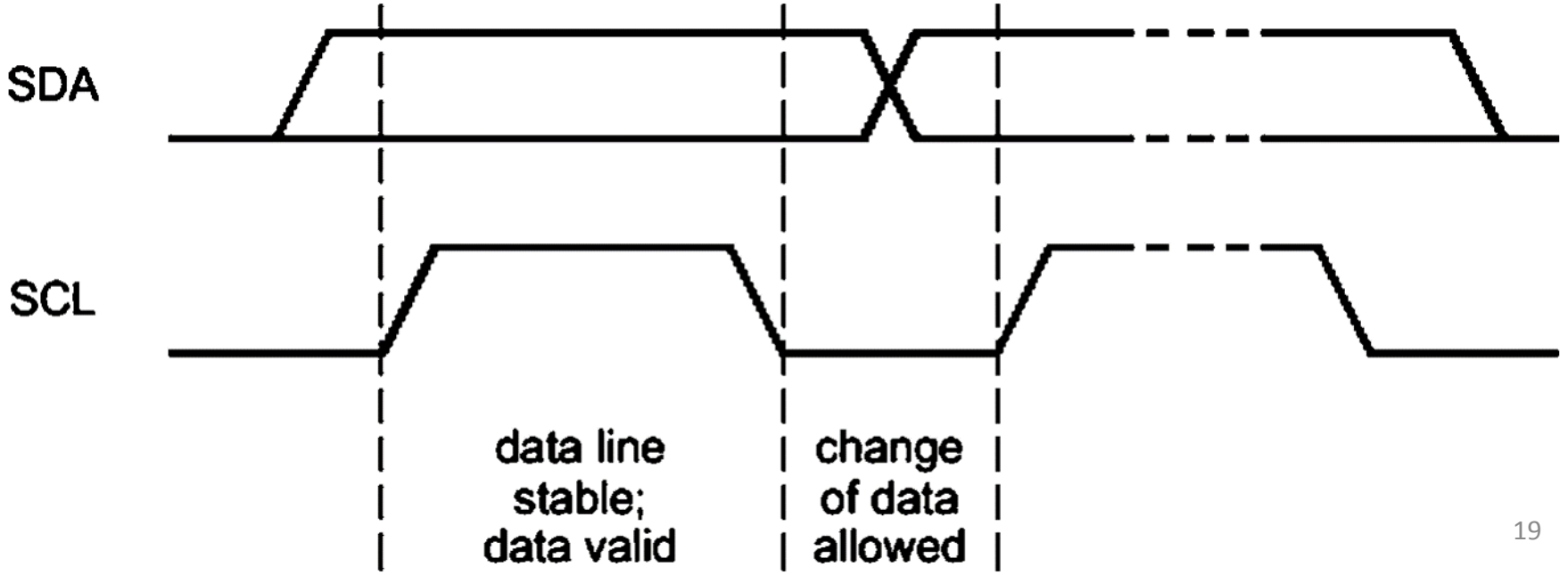
- **I2C bus**


+VDD
pull-up resistors
Rp
Rp
SDA (Serial Data Line)
SCL (Serial Clock Line)
SCLK
SCLKN1 OUT
DATAN1 OUT
SCLK IN
DATA IN
DEVICE 1
SCLK
SCLKN2 OUT
DATAN2 OUT
SCLK IN
DATA IN
DEVICE 2

I2C

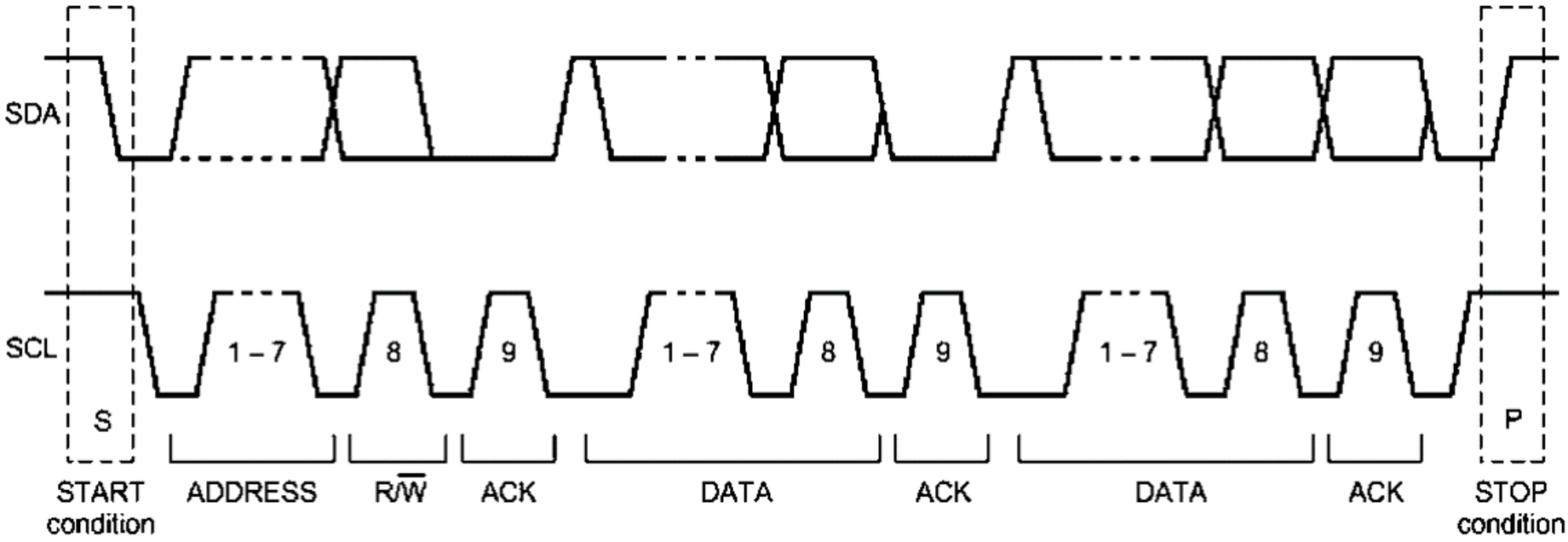
➢ I2C data validity

- For the data to be valid on the SDA line, it must not change while the SCL is high. The data on the SDA line should change only and only when the SCL line goes low.



I2C

- **I2C frame format**



I2C

I2C frame format

- Start bit: 1 bit indicates the start of a new frame, always logic low.
- Address: 7 bits that decides slave address which is the device that will process the upcoming sent data by the master.
- R/$\overline{W}$: 1 bit to decide type of operation, logic high for read, logic low for write.
- ACK: 1 bit sent by the slave as acknowledge by polling line low.
- Data: each 8-bits of data sent is followed by ACK bit by the receiver.
- Stop bit: 1 bit indicates the end of the frame.

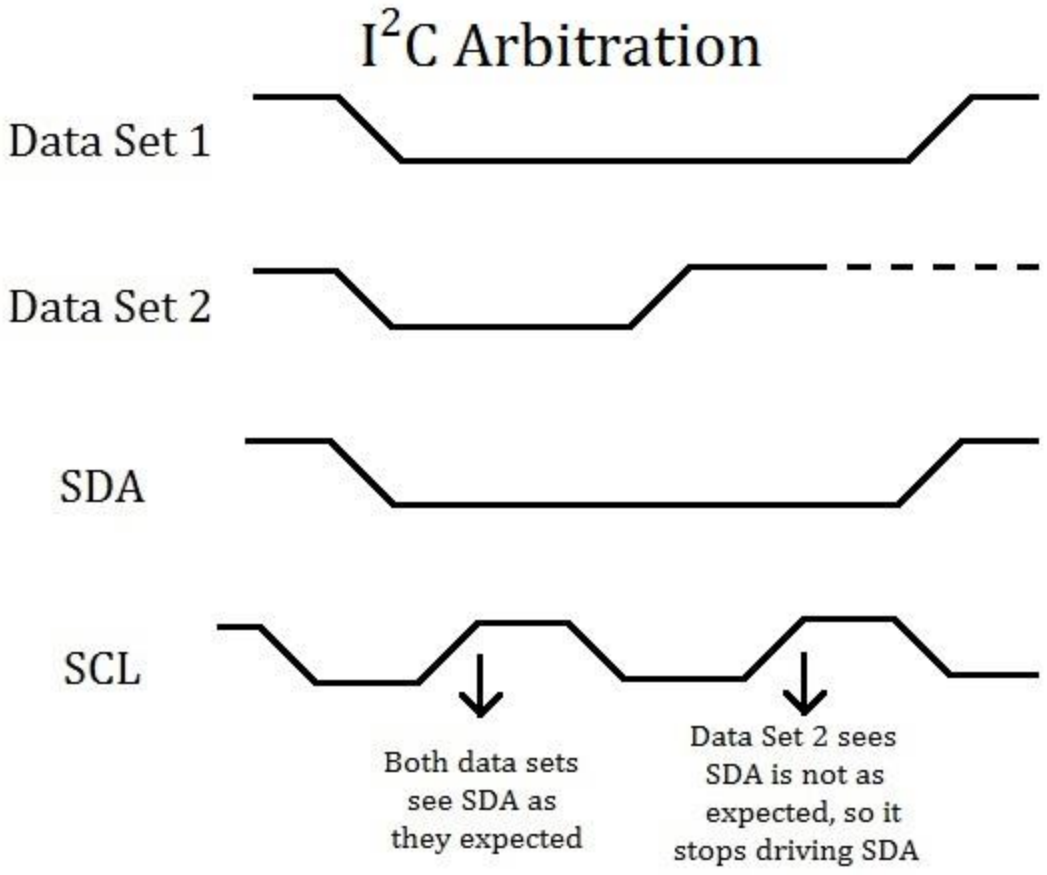
I2C

I2C bus arbitration mechanism

- Since the bus structure is a wired AND (if one device pulls a line low it stays low), you can test if the bus is idle or occupied.
- When a master changes the state of a line to HIGH, it MUST always check that the line really has gone to HIGH. If it stays low then this is an indication that the bus is occupied and some other device is pulling the line low.
- Therefore the general rule is: If a master can't get a certain line to go high, it lost arbitration and needs to back off and wait until a stop condition is seen before making another attempt to start transmitting.

I2C

- **I2C bus arbitration mechanism**



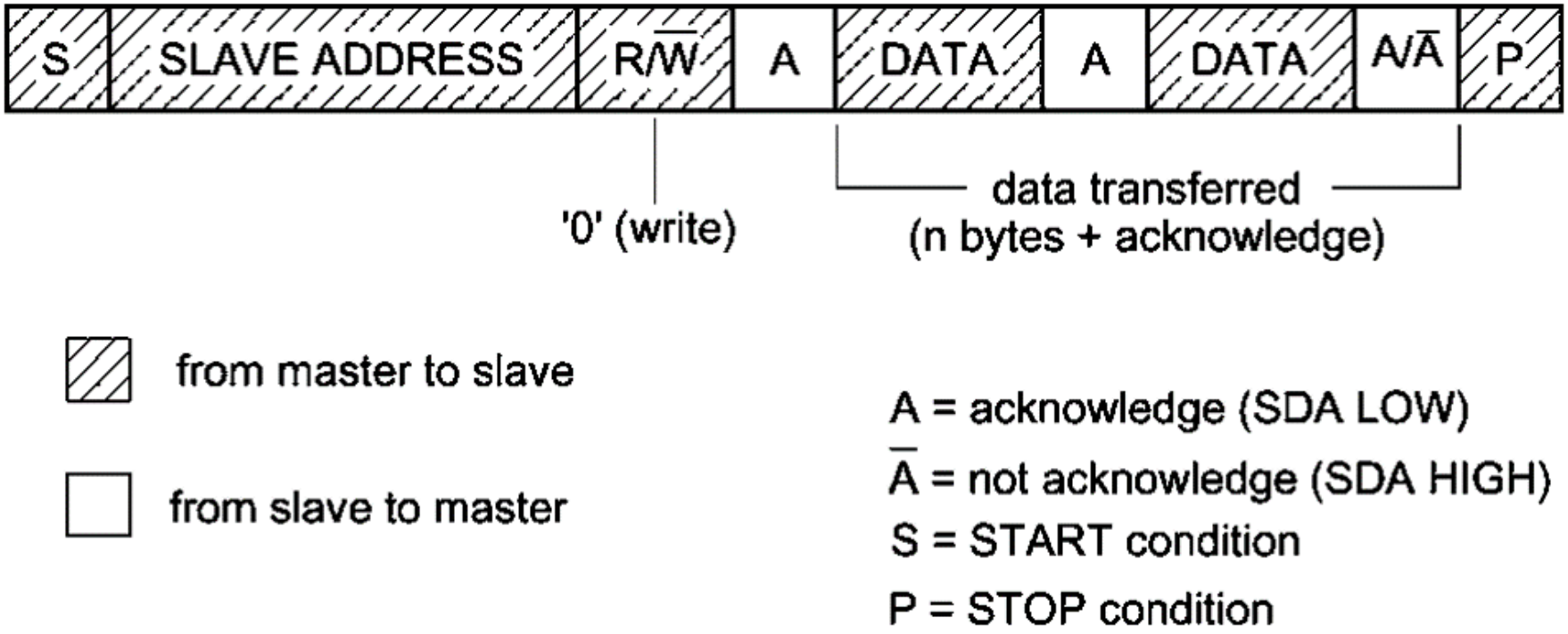
I2C

➢ Case 1: Master (Transmitter) to Slave (Receiver) Data Transfer

- The Master sends the START sequence to begin the transaction.
- It is followed by Master sending 7-bit Slave address and the R/$\overline{W}$ bit set to zero. We set it to zero because the Master is writing to the Slave.
- The Slave acknowledges by pulling the ACK bit low.
- Once the Slave acknowledges the address, Master can now send data to the Slave byte-by-byte. The Slave has to send the ACK bit after every byte it receives.
- This goes on till Slave can no longer receive data and does NOT send the ACK bit.
- This is when the Master realizes that the Slave is not accepting anymore and then STOPs the transaction (or RE-START).
- An example of this case would be like performing page write operations on a EEPROM chip.

I2C

- **Case 1: Master (Transmitter) to Slave (Receiver) Data Transfer**



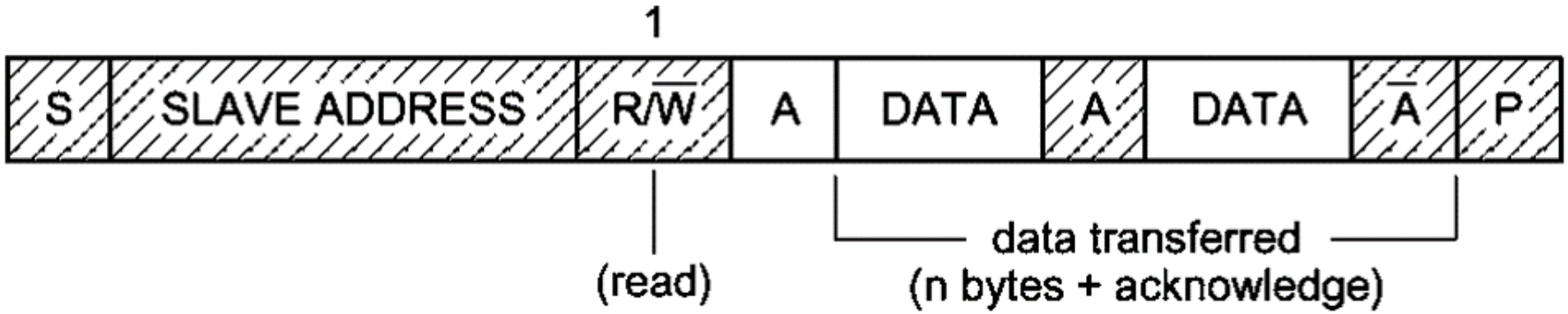
I2C

➢ Case 2: Slave (Transmitter) to Master (Receiver) Data Transfer

- The Master sends the START sequence, followed by the 7-bit Slave address and the R/$\overline{W}$ bit set to 1.
- We set R/$\overline{W}$ bit to 1 because the Master is reading from the Slave.
- The Slave acknowledges the address, thus ready to send data now.
- Slave keeps on sending data to the Master, and the Master keeps on sending ACK to the Slave after each byte until it can no longer accept any more data.
- When the Master feels like ending the transaction, it does not send the ACK, thus ending with the STOP sequence.
- An example of this case could be an Analog to Digital Converter (ADC) sending data to the microcontroller continuously. The microcontroller accepts data as long as it wants to, after which it stops/finishes execution.

I2C

- **Case 2: Slave (Transmitter) to Master (Receiver) Data Transfer**



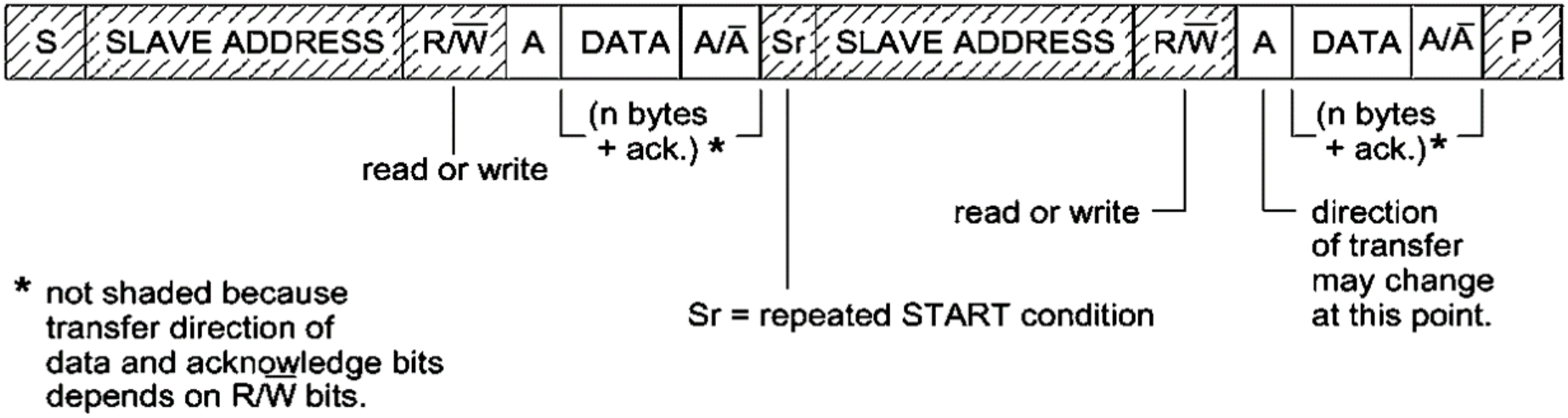
I2C

➢ Case 3: Bi-directional Read and Write in same Data Transfer

- The Master sends out the START sequence, followed by the 7-bit Slave address and the R/$\overline{W}$ bit, then the Slave acknowledges the address.
- Depending upon the value of the R/$\overline{W}$ bit, read/write operations are performed (like the previous two cases).
- Whatever the case it may be, it always ends with the receiver not sending the ACK.
- At the end the Master attempts a repeated START and the entire process repeats again, until the Master decides to STOP.
- An example of this case could be performing sequential read from a EEPROM chip. It is bi-directional because the CPU first writes the address from where it would like to start reading, followed by reading from the device.

I2C

- **Case 3: Bi-directional Read and Write in same Data Transfer**



I2C

I2C notes

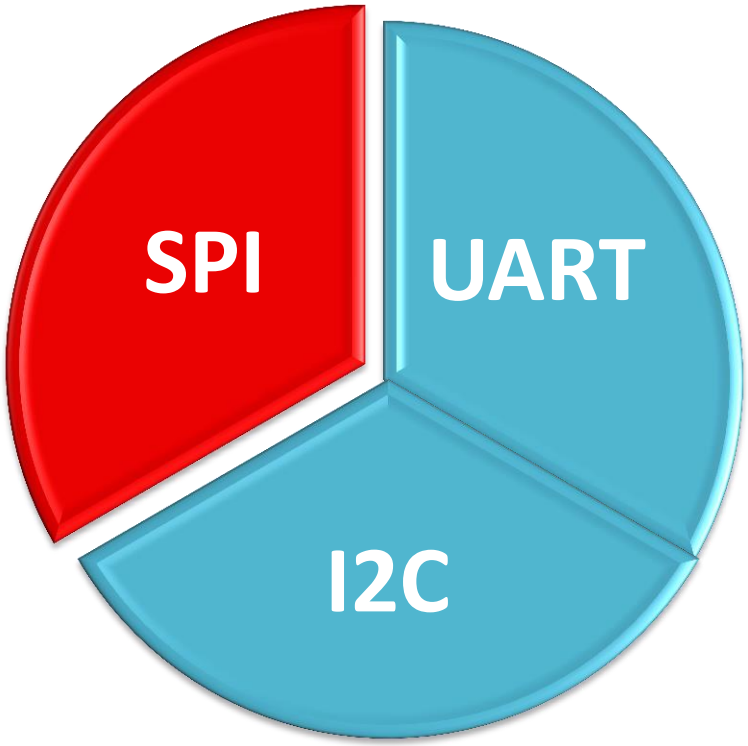
- I2C supports a wide range of voltage levels, hence you can provide +5 volts, or +3.3 volts as Vcc easily, and other lower/higher voltages as well.
- This gives us a wide range of choices for the values of the pull-up resistors Anything within the range of 1k to 47k should be fine, however values lower than 10k are usually preferred.
- I2C is a half duplex communication protocol.
- I2C supports serial 8-bit data transfers up to a speed of 100 kbps, which is the standard clock speed of SCL. However, I2C can also operate at higher speeds – Fast Mode (400 kbps) and High Speed Mode (3.4 Mbps).

I2C

I2C notes

- Maximum number of nodes is 112, as address is 7-bits and there are 16 nodes reserved.
- I2C is used for short distance communication.
- The Slave is allowed to hold the clock line low until it is ready to give out the result. The Master must wait for the clock to go back to high before continuing with its work, this is called clock stretching.

SPI



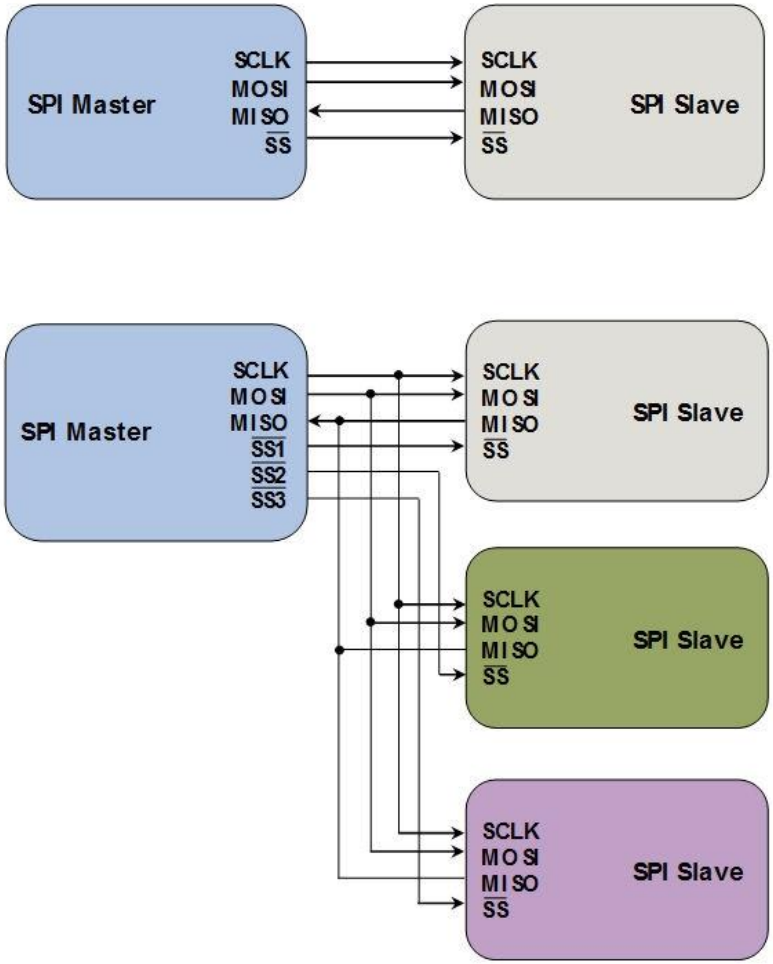
SPI

What is SPI?

- Serial Peripheral Interface (SPI) bus is a synchronous serial communication interface specification used for short distance communication.
- The SPI bus can operate with a single master device and with one or more slave devices.

SPI

- **SPI connection**



SPI

- **SPI connection**
  - SCLK : Serial Clock (output from master).
  - MOSI : Master Output, Slave Input (output from master).
  - MISO : Master Input, Slave Output (output from slave).
  - SS : Slave Select (active low, output from master).

SPI

- **SPI notes**
  - SPI is a full duplex communication protocol.
  - SPI is single master multiple slaves communication protocol.
  - SPI protocol has complete flexibility for the bits transferred as there is no limit for 8-bit word or message size or purpose.