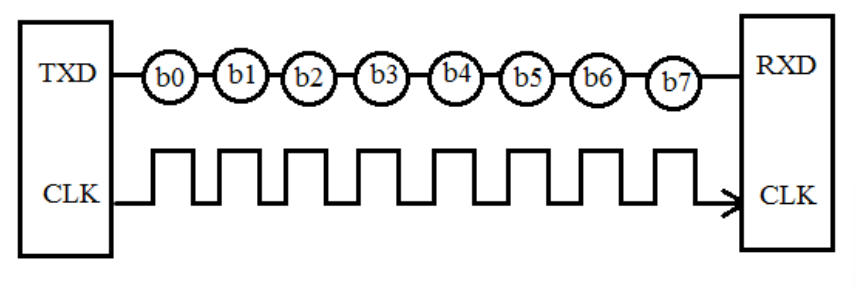
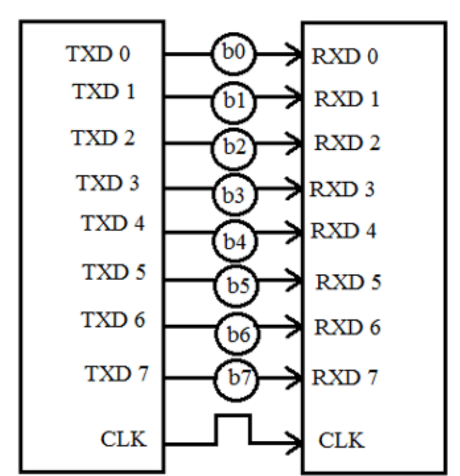
Communication: exchange of information between two microcontrollers in the form of bits

Communication protocols: set of defined rules. two types: serial and parallel

 Serial Communication Protocol

Parallel Communication Protocol

# SERIAL COMMUNICATION PROTOCOL

## Simplex Method

One-way communication. If sender is sending, the receiver can only accept. Eg: Television, Radio

## Half Duplex Method

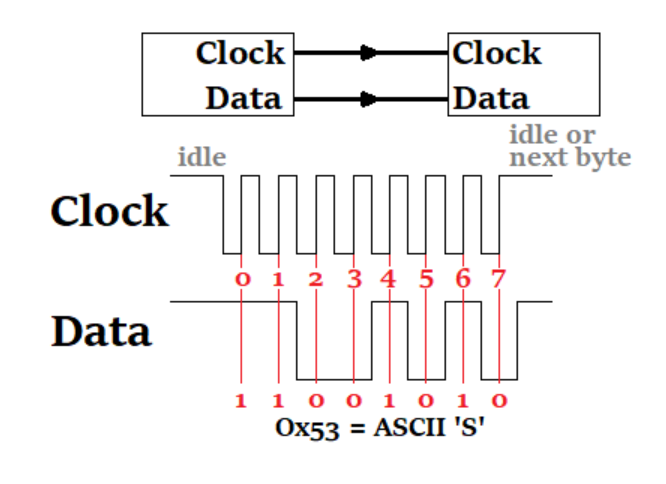
Both sender and receiver can be active but not at the same time. If sender is sending, receiver can accept but cannot send simultaneously. Eg: Internet

## Full Duplex Method

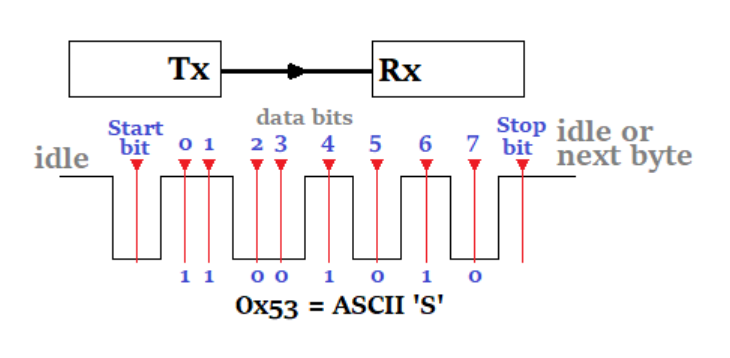
Both receiver and transmitter can send data to each other at the same time. Eg: mobile phone

## CLOCK SYNCHRONIZATION

## Synchronous Serial Interface

* all the devices use single CPU bus to share data and clock
* data transmission becomes faster
* no mismatch in baud rate
* no start, stop and parity bits are added to data
* Transmitter: shift of the data onto serial line; clock => separate signal
* Receiver: data extract using clock, converts serial data back to parallel form

### Asynchronous Serial Interface

* no external clock signal, suitable for long distance communication
* relies on Data Flow Control, Error Control, Baud Rate Control, Transmission Control and Reception Control
* Transmitter: shifts parallel data onto serial line using its own clock and adds start, stop, parity check bits.
* Receiver: extracts data using its own clock, strips start, stop, parity bits, converts back to parallel form

## TERMINOLOGY

### Baud Rate

Rate at which the data is transferred between the transmitter and receiver in the form of bits per second (bps). Most commonly 9600. Needs to be same for transmitter and receiver.

### Framing

Number of data bits to be sent from transmitter to receiver. 8 bits is standard.

### Synchronization

Tells the start and end of the data bits. Transmitter sets start and stop bits to the data frame for the receiver to identify.

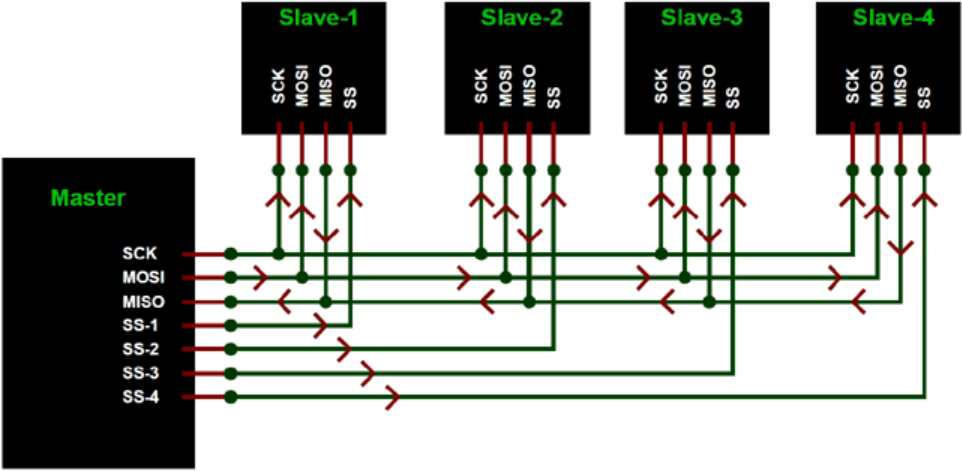
### Error Control

Parity bits added. Even no of 1s in data frame (even parity) --> parity bit set to 1. Odd no of 1s (odd parity) --> odd parity bit cleared

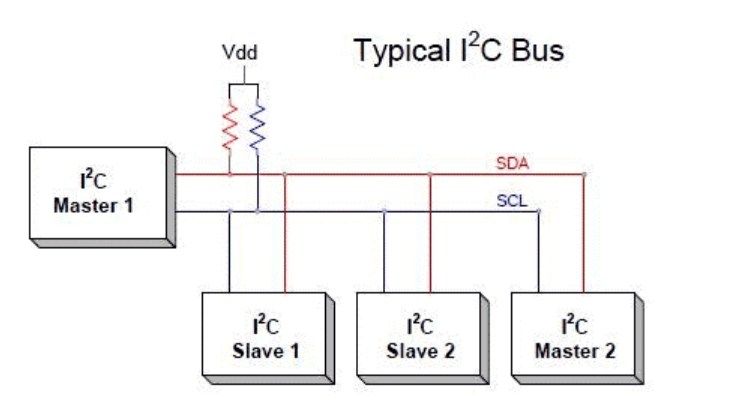
## SYNCHRONOUS SERIAL PROTOCOLS

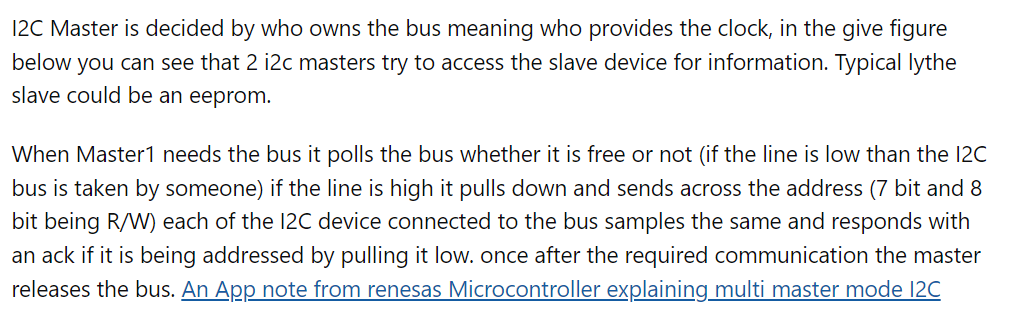
### Serial Peripheral Interface (SPI) Protocol

* Allows several SPI microcontrollers to be interconnected
* The SPI may be configured either as master or as a slave
* The four basic SPI signals (MISO, MOSI, SCK and SS), VCC and Ground are the part of data communication, so it needs 6 wires to send and receive data from slave or master.
* Theoretically, the SPI can have unlimited number of slaves.
* The SPI can deliver up to 10Mbps of speed and is ideal for high-speed data communication.



### Inter Integrated Circuit (I2C) Serial Communication

* Two-line communication between different ICs or modules: SDA (Serial Data Line) and SCL (Serial Clock Line)
* Both the lines must be connected to a positive supply using a pull up resistor
* Speed up to 400Kbps; limited length communication, ideal for onboard communication
* Uses 10bit or 7bit addressing system to target a specific device, so it can connect upto 1024 devices.



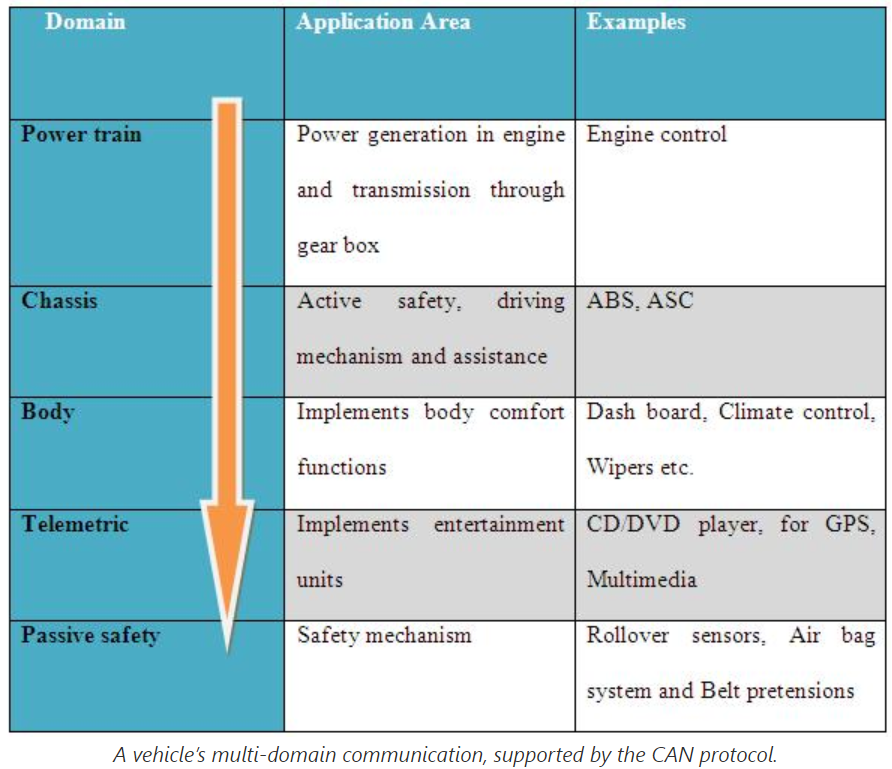
### Controller Area Network (CAN)

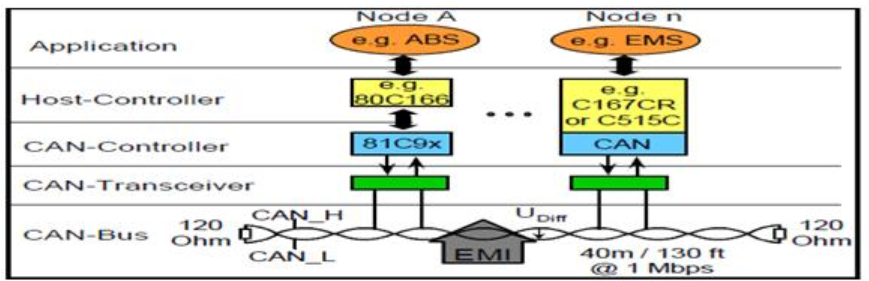
* Communication between Engine Control Units (ECUs) and sensors.
* The CAN bus system allows for central error diagnosis and configuration across all ECUs.
* Each ECU contains a chip for receiving all transmitted messages, decide relevance and act accordingly
* CAN messages are prioritized via IDs so that the highest priority IDs are non-interrupted.
* Speed up to 1Mbps; robust, low-cost

**CAN protocol: Understanding the controller area network**

Before CAN: point-to-point wiring; each device connected to another using wires. Good enough for basic functions but not for exchange of real-time info between ECUs. No multi-domain communication possible (A domain is a group of electronic devices that have similar requirements to work properly in the system. For example, a CD/DVD player, GPS, and monitors and displays form a single domain. Similarly, the dashboard, air-conditioning system (or climate control), wipers, lights, and door locks form another domain.)

CAN facilitates multi-domain communication.



Every electronic device (or node) that communicates via the CAN protocol is connected with one another through a common serial bus, which allows for the transfer of messages.

Host Controller: ECU/MCU responsible for function of the node

CAN-Controller: chip embedded in host-controller or separate, converts messages from the nodes per the CAN protocols

**No master-slave relation in CAN. Every node can read/write data on CAN bus.**

When the node is ready to send data, it checks the availability of the bus and writes a CAN frame onto the network. A frame is a structure that carries a meaningful sequence of bit or bytes of data within the network.

**Address-based protocol:** Data packets contain address of target device

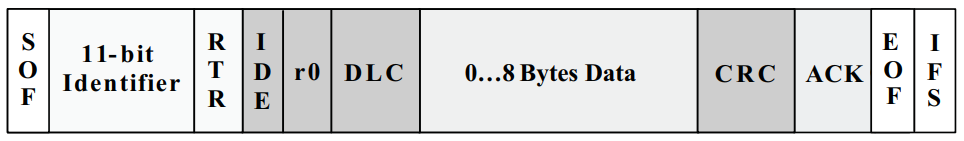
**Message-based protocol:** Every message has a pre-defined ID, typically used

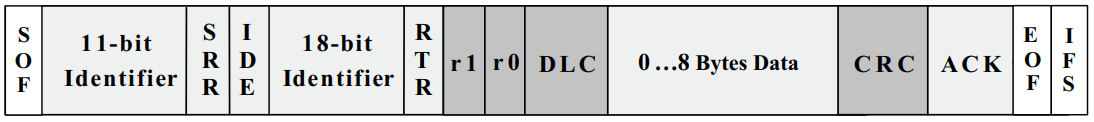
Message: data packet, 10 bytes of data organized in a frame. The data carried in each byte is defined in the CAN protocol. All nodes using the CAN protocol receive a frame and depending on the node’s ID, the CAN “decides” whether or not to accept it. If multiple nodes send the message at the same time, the node with the highest priority (so, the lowest arbitration ID) receives the bus access. Lower priority nodes must wait until the bus is available.

**Binary values in CAN protocol are termed as dominant and recessive bits.** CAN define the logic “0” as dominant bit and logic “1” as recessive bit.In the CAN system dominant bit always overwrites the recessive bit.

## MESSAGE FRAME

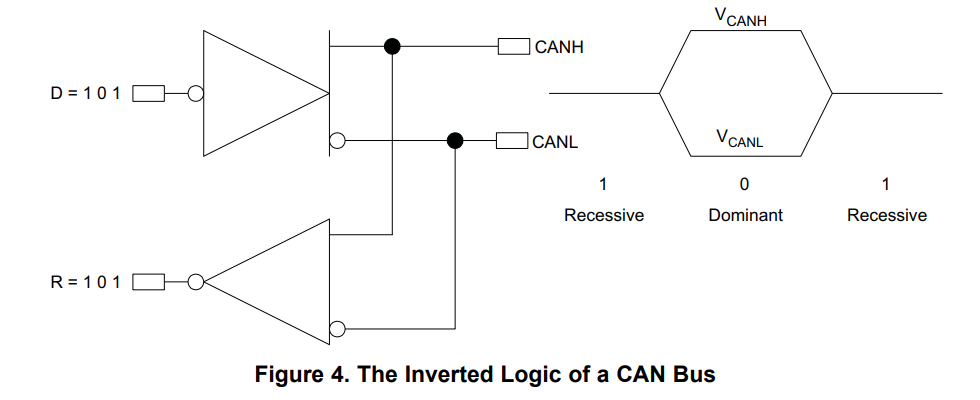
CAN is a carrier-sense, multiple-access protocol with collision detection and arbitration on message priority (CSMA/CD+AMP). CSMA means that each node on a bus must wait for a prescribed period of inactivity before attempting to send a message. CD+AMP means that collisions are resolved through a bit-wise arbitration, based on a preprogrammed priority of each message in the identifier field of a message.



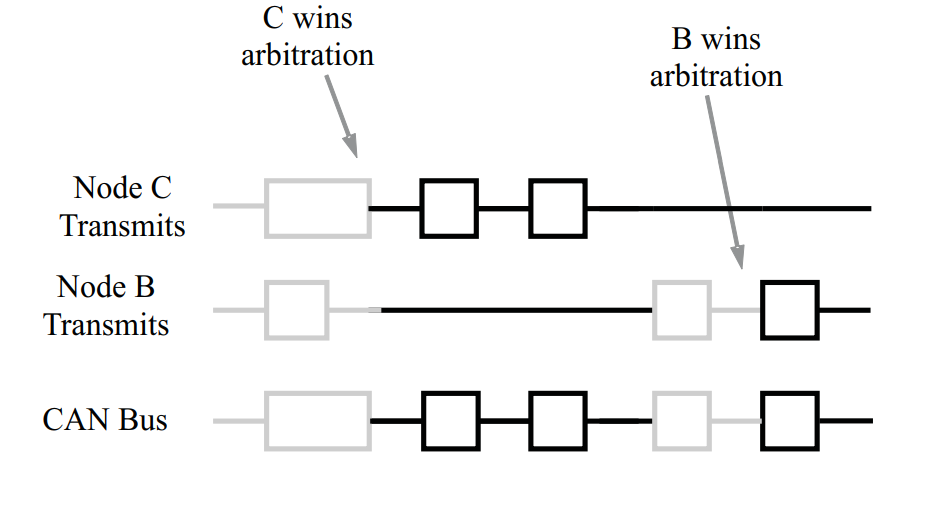


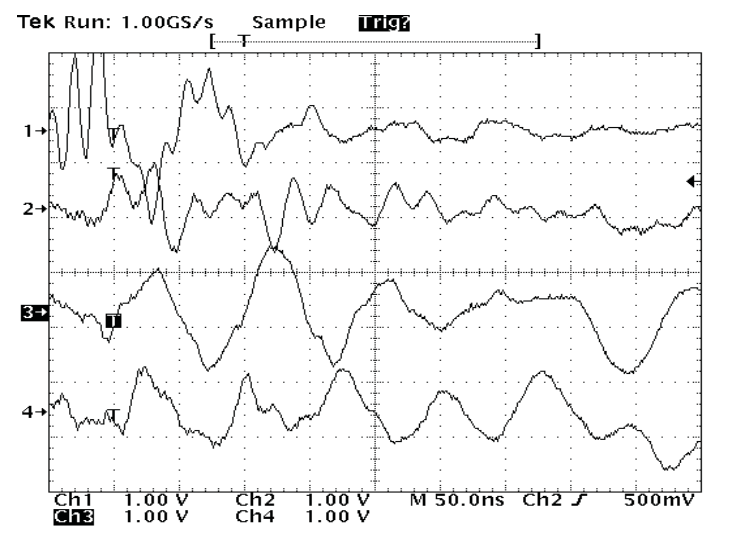
* SOF: start of frame, single dominant bit, marks start of message, used to synchronize nodes after being idle
* Identifier: establishes priority of the message
* IDE: identifier extension, single bit, dominant --> standard CAN with no extension
* DLC: data length code, 4 bits, no. of bytes of data being transmitted
* Data: upto 64 bits may be transmitted

## ARBITRATION



If two nodes try to occupy the bus simultaneously, access is implemented with a non-destructive, bit-wise arbitration. The lower the binary message identifier number, the higher its priority. A dominant bit always overwrites a recessive bit on a CAN bus. The node that sends a last identifier bit as a zero (dominant) while the other nodes send a one (recessive) retains control of the CAN bus and goes on to complete its message. Note that a transmitting node constantly monitors each bit of its own transmission

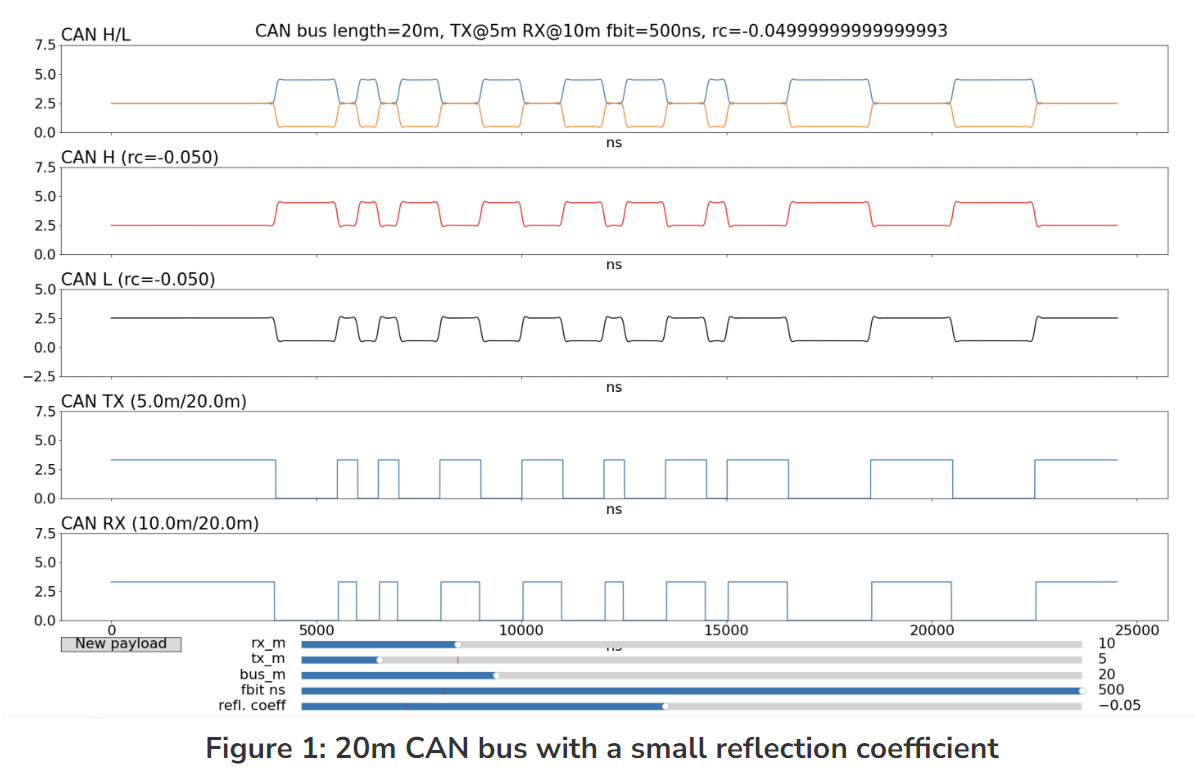
**Q. The main data processing is done by the microcontroller. Then why do we need the can transceiver?**

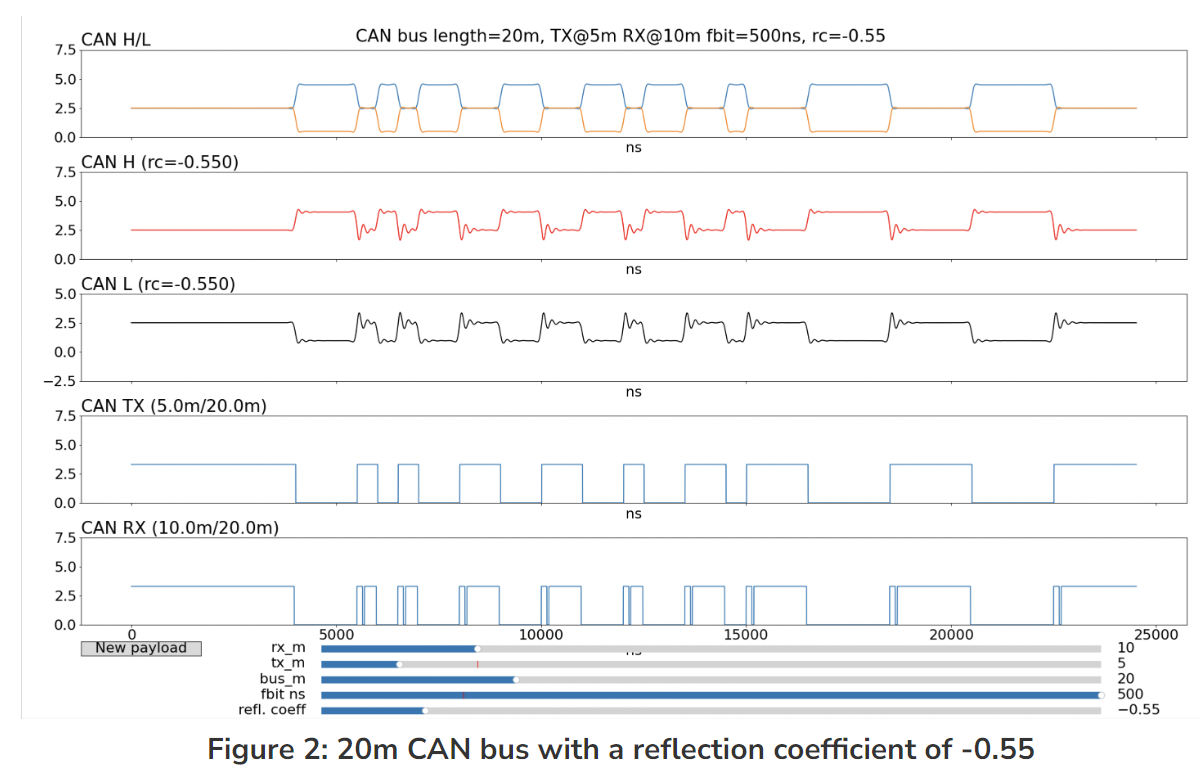
The role of the transceiver is simply to drive and detect data to and from the bus. It converts the single-ended logic used by the controller to the differential signal transmitted over the bus. It also determines the bus logic state from the differential voltage, rejects the common-mode noise, and outputs a single-ended logic signal to the controller. Additionally, many transceivers provide features to isolate the controller from bus conditions such as electrostatic discharge (ESD) or electrical over-stress (EOS) damage, so that a node would be protected if an extreme bus condition were to destroy the transceiver.

Voltage that is common to both lines is referred to as common-mode voltage. The use of twisted-pair cable and differential data transmission causes external noise from neighbouring signal lines or noise sources to be coupled onto both lines (common-mode) and rejected by the transceiver. CAN transceivers are specifically designed and tested for their ability to reject this common-mode noise.

The CAN transceiver determines if the signal is dominant (logic 0) or recessive (logic 1) by measuring the difference between the CAN H and CAN L lines, with a difference of more than 700mV equal to a dominant bit. Transceivers also apply a hysteresis function to prevent rapid state changes (typically 50mV).

**Q. Why do we need 120 ohms resistors at both the ends of the can bus?**

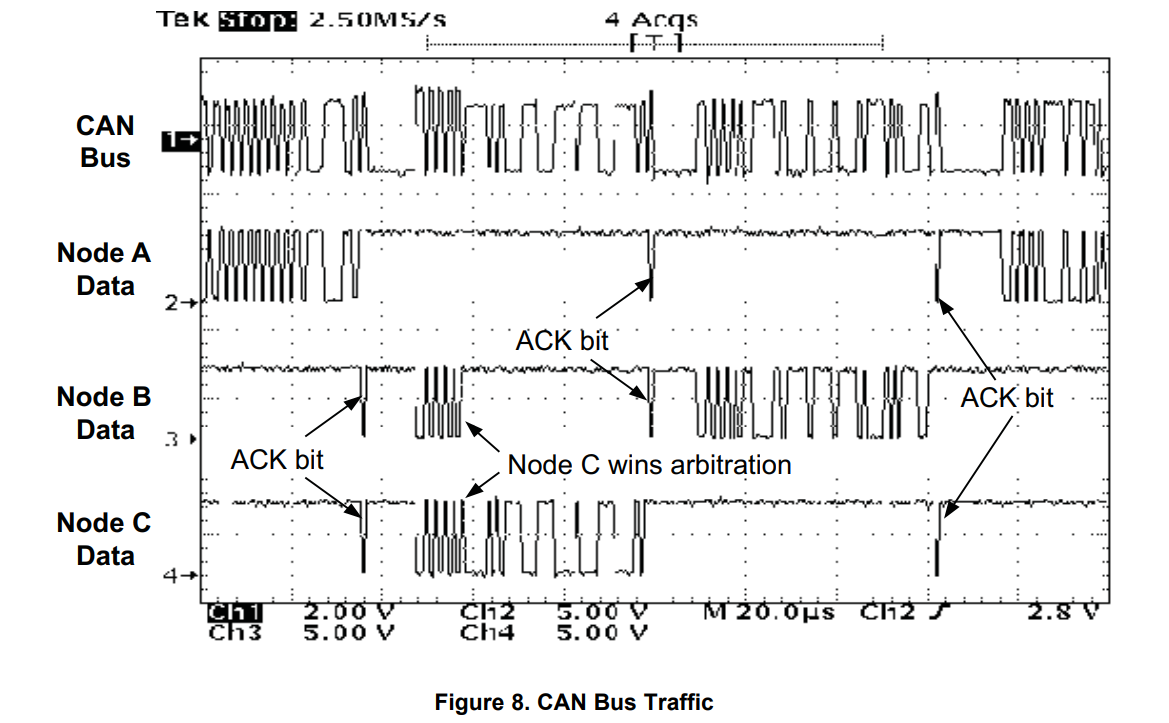
The cable is specified to be a shielded or unshielded twisted-pair with a 120-Ω characteristic impedance (Zo). The ISO 11898 Standard defines a single line of twisted-pair cable as the network topology, terminated at both ends with 120-Ω resistors, which match the characteristic impedance of the line to prevent **signal reflections**. A reflected pulse moves back up the bus until it hits the other end, then reflects again. The reflection can be characterised by the reflection coefficient: a value of 0.5 means that the pulse reflected back is the same sign as the pulse hitting an end, but half the amplitude. These reflections add together as an infinite geometric sum, giving the resulting signal for the transceiver.



What we can see here is the classic distortion we call ringing: the reflections flip sign when they bounce of an end of the bus and so the overall sum of these waves crashing into each other is the distinctive ‘wobbling’ with a diminishing amplitude. The differential signal (CAN H minus CAN L) has little spikes due to the ringing and if the amplitude is large enough then the hysteresis of the receiving transceiver is overwhelmed and the digital state is flipped back for a short time. This shows up as glitches on the CAN RX signal. Where the glitches appear and how wide they are depends on the exact placement of the receiver and transmitter on the bus.

It’s worth emphasising that this distortion of the CAN bus signals is deterministic: it is not random noise and the same pattern of transmitted data will produce the same pattern of received data.

**Q. What happens when multiple nodes try to send data on the bus at the same time? How is it decided which one sends first?**

****

Arbitration as explained before.

**Bonus Q - If 2 CAN nodes start transmitting messages with ID 120 at the same time what will happen?**

If the transmitting messages have the same ID, the Remote Transmission Request (RTR) bit decides the message that will be transmitted first. RTR: dominant --> Data Frame, recessive --> Remote Frame. In case the RTR bits differ in the two messages, the one with the dominant RTR gets the priority.