Can Introdcution

Owner	A Angel David Sanchez
Tags	Embedded software

Controller Area Network

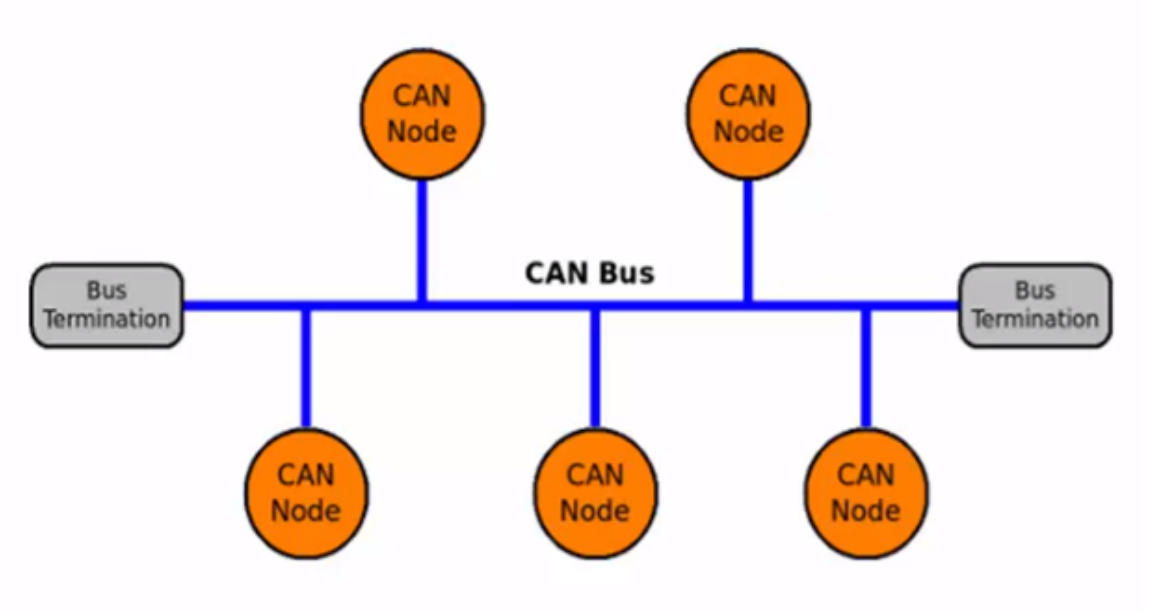
▼ CAN Bus Protocol and its features (Fundamentals)

- The CAN protocol was originally design in the late 1980’s by German Company Robert Bosch.
- CAN is a mullti-master serial communication bus whose basic design specification called for high speed, high noise-immunity and error detection features and iit can offers data communication up to 1 Mbit/sec.
- The error confinement and the error detection features make it more reliable in noise-critical environments (being in the automotive industry).
- Using the CAN protocol me can communicate modules like the Powertrain Module, to the body control module to the GPS module.
- An automovile is a dynamic system and with Can we are looking for reliable communicatiopn and which is more robust in nature.

▼ CAN Most Attractive features

- CAN is low cost
- it has extreme robustness
- it has high data transmission speeds (up to 1 Mbit/sec)
- Reliability. Excellent error handling and Error Confinenment abilities
- Automatic re-transmission of faulty messages
- Automatic Bus disconnection of nodes that are suspected to be physically faulty
- Functional addressing - data messages do not contain source or destination addresses, only identifiers relating to their function and /or priority.

- CAN is a broadcast type of Bus (unlike traditional network such as USB or Ethernet, or I2C, CAN does not send data point-to-point from node A to node B under the supervision of a central Bus master), meaning that all the devices can hear the transmission.



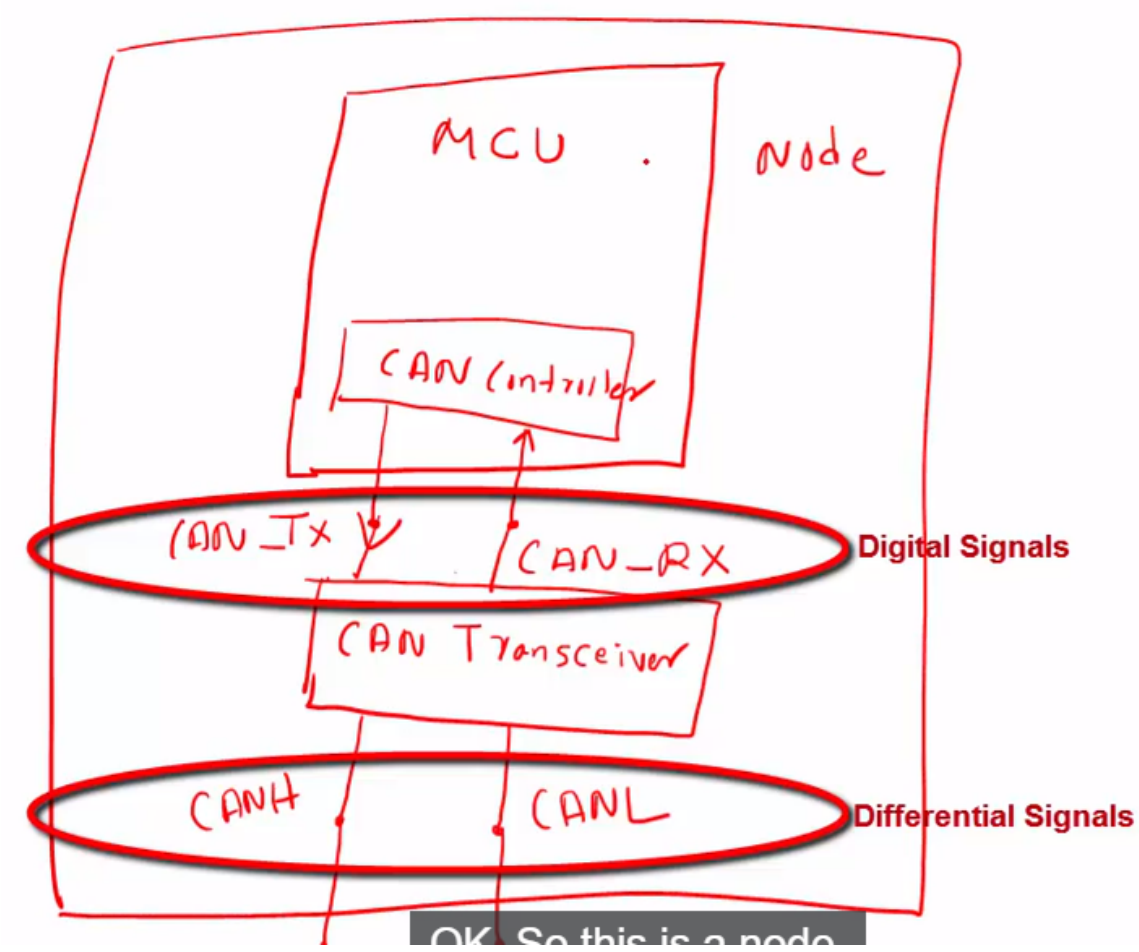
- Also there is now way top send data specifically to a node by its address or something
- All nodes will pick up the traffic on the bus
- The CAN standard defines a communication network that links all the nodes connected to a bus and enables them to talk with one another. There may or may not be a central control nofe, and nodes may be added at any time, even while the network is operating (hot-plugging).

▼ Operating Principles of the CAN Bus

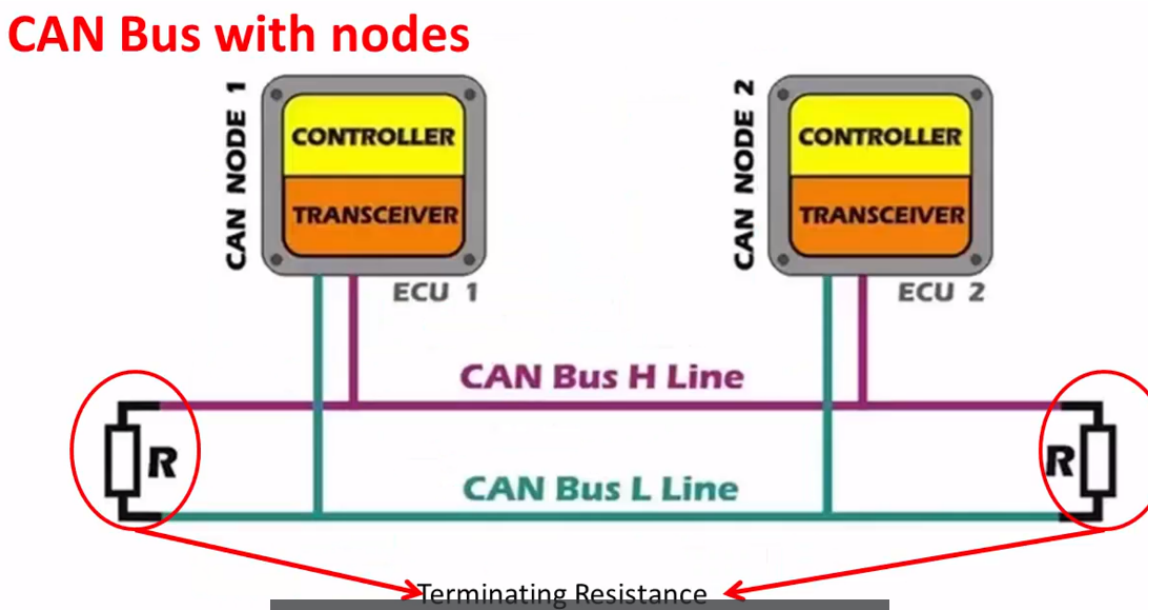
- What we normally have is an MCU (Microcontroller) and also a CAN controller that has 2 digital pins in order to communicate, which are CAN TX and CAN RX, one to transmit the data and the other to receive it (the pins are digital, sending a digital sign), also called as single ended pins, what follows is a CAN transciever.
- Now, according to the specifications, we cannot use these digital signals in order to communicate with another node (is not allowed), instead with use something called differential signals, but the MCU does not produce these. these signals are the ones used in order to communicate with another node and it’s because these signals are good for a vey short range of

communication, but in the automobile industry w cannot use them since in the industry the standard is used for long range communication.

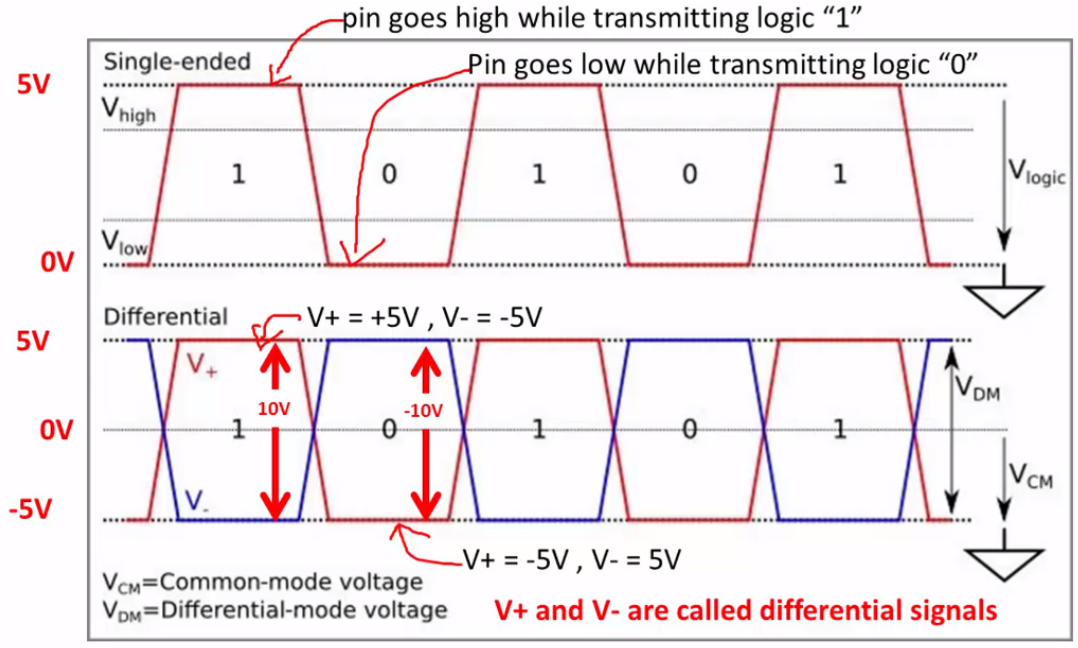
- The differential signals give more immunity to the noise and data can be transmitted more reliably, that is why, there is a need to convert the single ended signals to differential signals, and in order to achive that signals we need a CAN transciever and what this does is it takes single ended signals and conver them into differential signals and we call them as CANH (CAN High) and CANL (CAN Low). After having all that what we have is what we called a node.



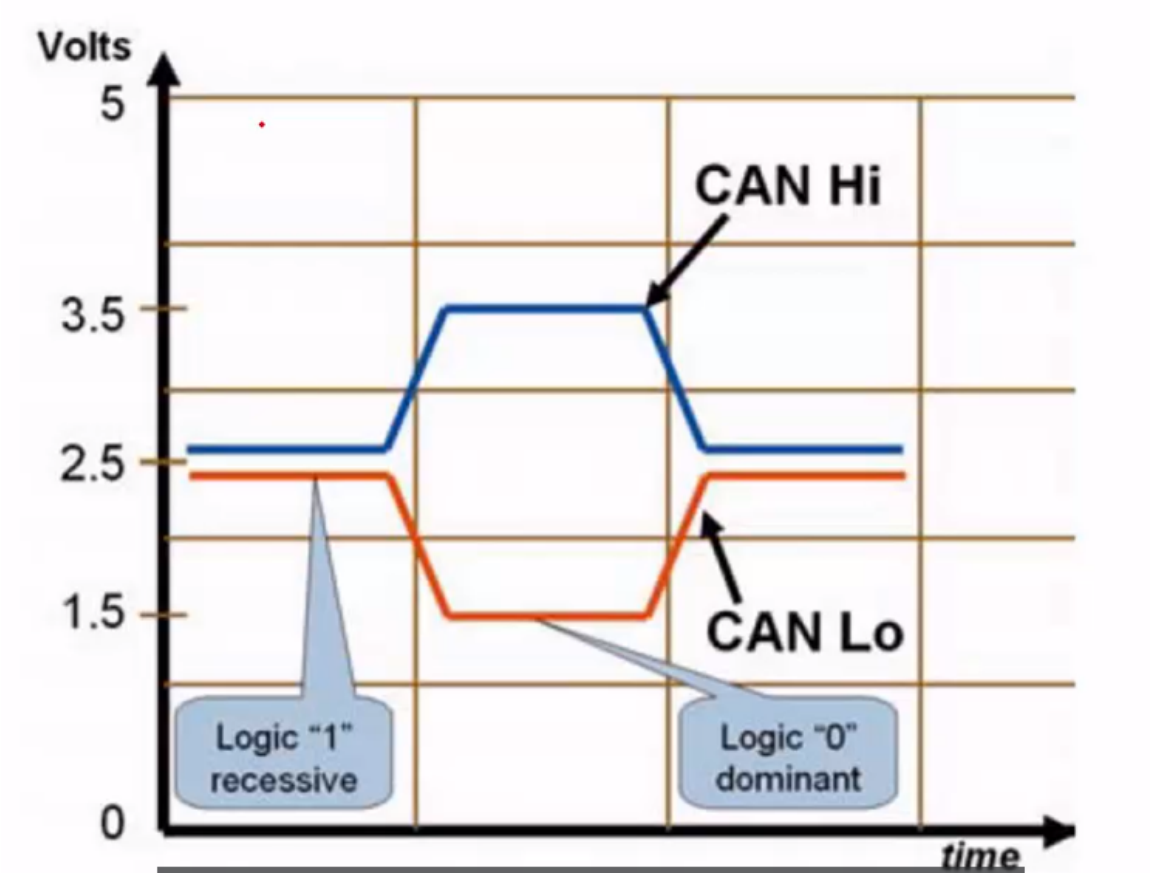
- So basically a CAN bus will be having 2 lines (CANH and CANL), both lines are terminated by a termination register (RL). So if we want to attach a node to the bus, we have to connect CANH terminal to our node to the CANH line of the bus and the CANL terminal to our node to the CANL line of the bus.



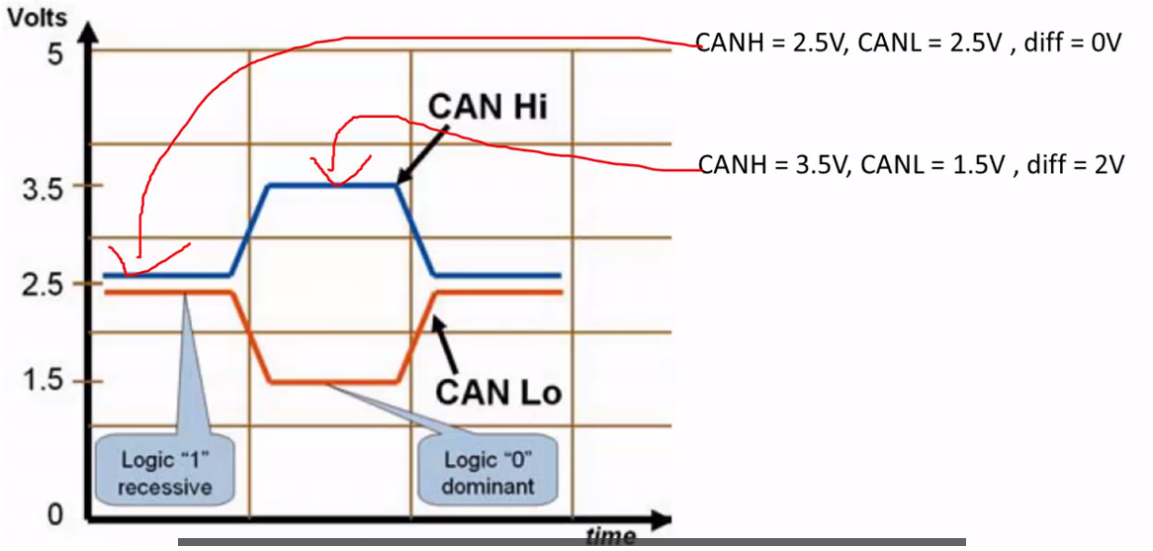
- **SINGLE ENDED VS DIFFERENTIAL SIGNALS**
  - Differential signaling is a method of data communication, in a single ended communication, when the pin goes high while transmittins we can interpret it as a logic 1 and when the pin goes low while trasnmmiting we can interpret it as a logic 0 (0v). But in the differential signals, it implies of 2 complementary voltage signals in order to trasnmit one information (called differential signals), now, in order to send a logical one signal, one becomes +5V and signals 2 becomes complementary of that -5v and the reciever considers the difference between thes 2 signals voltage levels and that is 10v, which in this case is consider as logical 1.
  - So even if the noise is added to this signaling, the noise will get added equaly to the signal and so, the reciever will consider the difference of that and that is why noise cancelling will happen automatically and that is yhe biggest advantage of the differential signal.
  - The reciever extract the information by detecting the potential difference between invertor and non invertor signals, and to send a 0, signal 1 becomes -5v and signal 2 becomes +5v and the difference is -10v



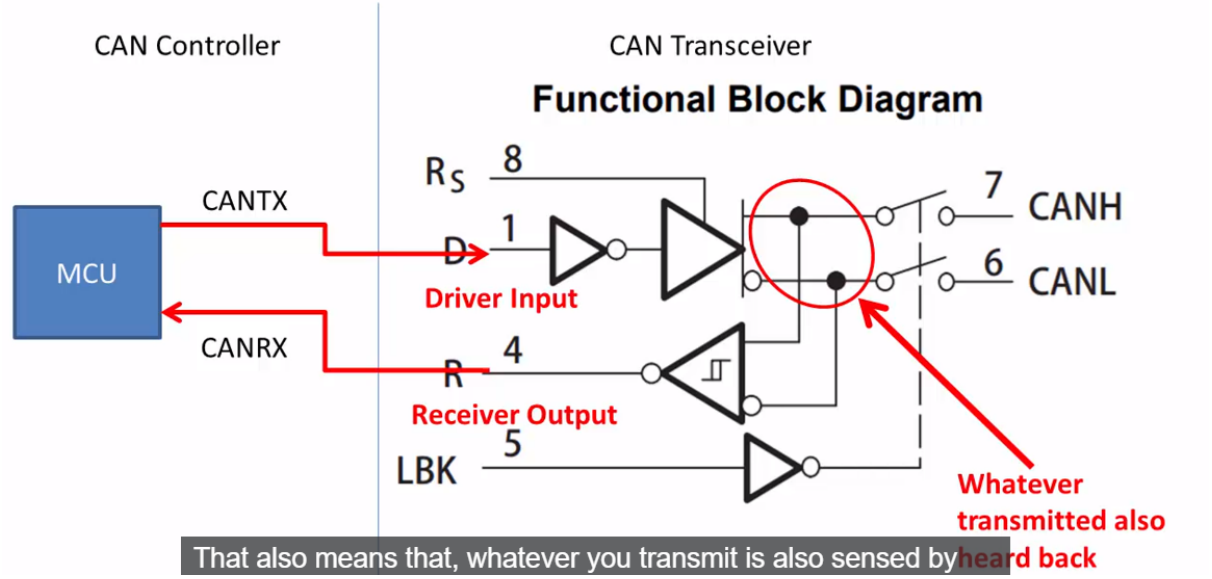
- **UNDERSTANDING CAN DIFFERENTIAL SIGNALS**
- As before, a transceiver produces differential signals (CANH and CANL), these are complementary signas (CANL is actually complementary signal of CANH)



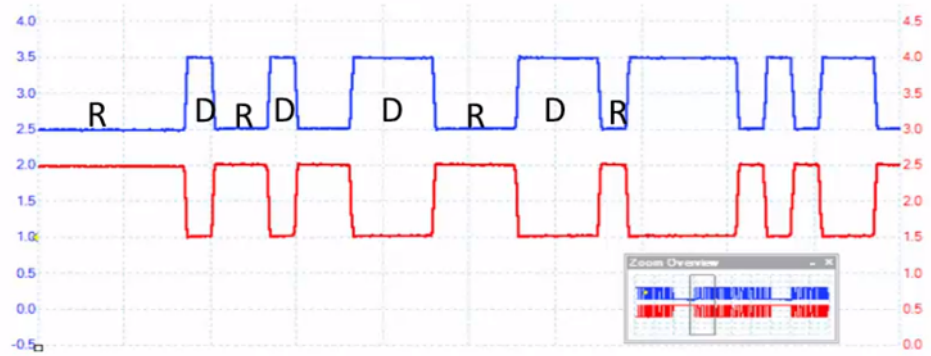
- When we are trying to send a logic 1, the CAN bus stae will be recessive, the recessive state is a place on the CAN bus where potential difference between CANH and CANL is 0V. and when we are trying to send a logic 0, the CAN bus state will be dominant, the dominant state is a place on the CAn bus where potential difference between CANH and CANL is ~2v.

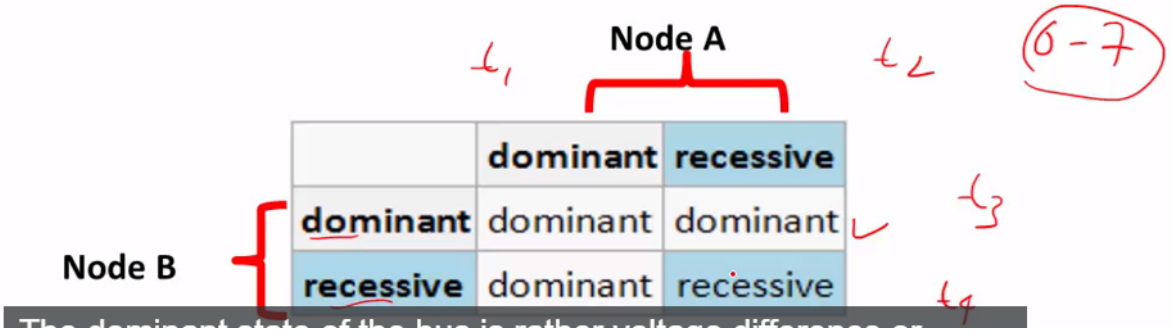


- When we use a transceiver, we have a Diver Input and a Receiver Output, these 2 are connected to the single ended pins (CAN_TX to Dirver Input and CAN_RX to Receiver Output), the Driver Input goes connected to the CANH and the Receiver Output with the CANL, also in the transceiver, we have a loop back, meaning that whatever we are trasnmitting the receiver can also heard it. So, if the Driver Input recieve a 1, the CANH and CANL will go into recessive state and if we input a 0, the state will be dominant.



- Since Dominant state of the bus means voltage difference of 2V, logic 0 (dominant) always overrides logic 1 (recessive) during data transfer.





CAN signaling

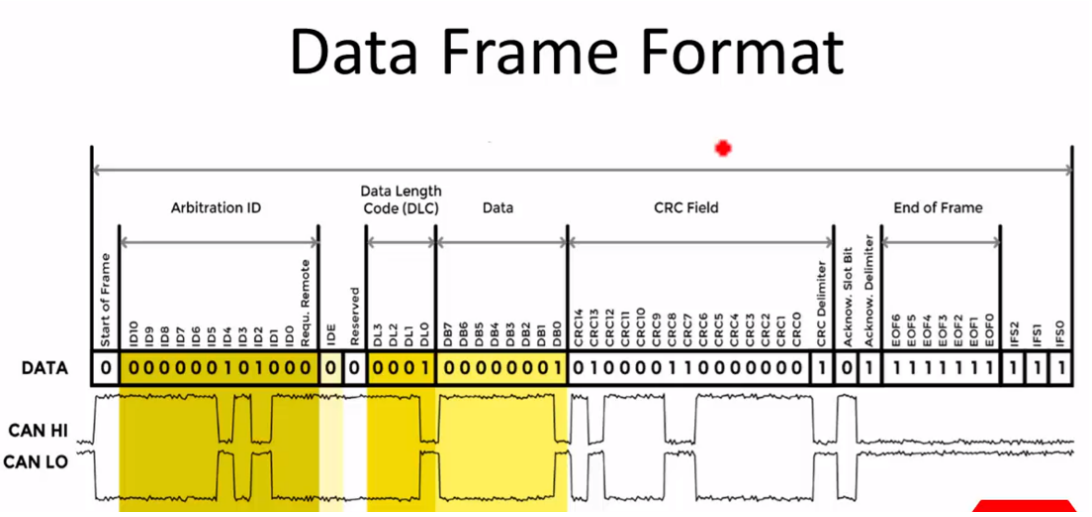
- Signalling is differential which is where CAN derives its robust noise immunity and fault tolerance
- Balanced differential signalling reduces noise coupling and allows for high signalling rates over twisted-pair cable
- Balanced means that the current flowing in each signal line is equal but opposite in direction, resulting in a field-cancelling effect that is a key to low noise emissions
- The use of balanced differential receivers and twisted-pair cabling enhance the common-mode rejection and high noise immunity of a 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 as shown in, , terminated at both ends with 120-Ω resistors. which match the characteristic impedance of the line to prevent signal reflections

▼ Defferent Message Formats

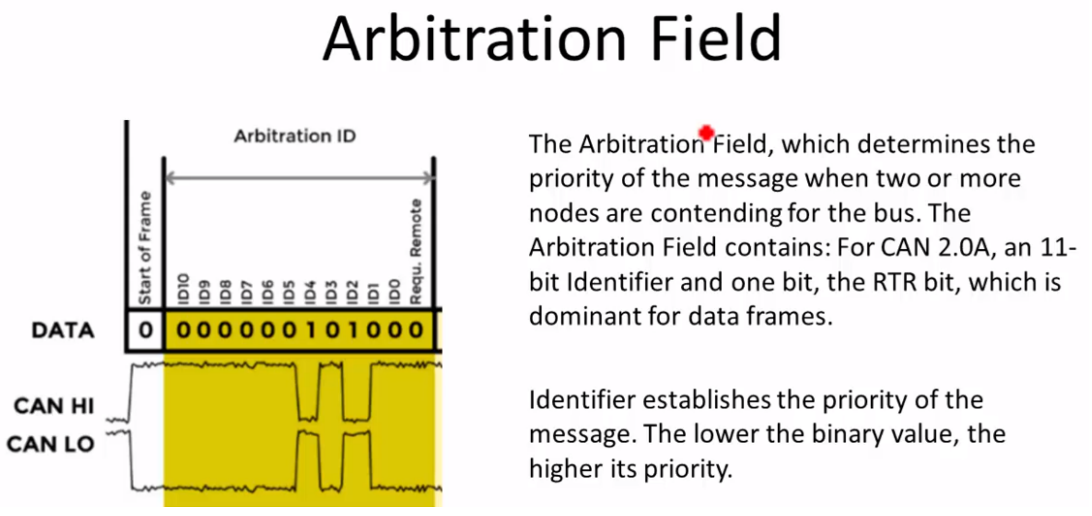
- There are 4 different message types (or frames) in CAN protocol: Data frame, Remote frame, Error frame, Overload frame. The data frame and Remote frame are the ones we are going to us in our applications more frequenty, but the error frame and overload frame are actually used by the

MCU automatically when they detect errors or any violation of signaling rules in the CAN protocol.

- When we program we basically use Data Frame more frequently than Remote frame.
- **DATA FRAME**
  - Data Frame is the most common message types in CAN communication
  - We use it very frequently
  - A node uses this frame to send a message to other nodes on the CAN bus.
  - It’s like broadcast a message.
  - In any digital communication or in any specification we can find that a frame has many fields (payload, length, CRC, the field that marks the beginning and the end of the frame, etc), there are varios segments in the data frame (SOF (Start of Frame), next is the arbitration field, which consumes 12 bits, next some control and length field, next is the data field, where we put our payload and can have a max zise of 8 Bytes, what follows is the CRC field, in this field we do not need to calculate and put any CRC code there, the controller has the ability to do that, after is a very important field which is the Ack (acknowledge) and is set to dominant by the receiving node, if it finds a frame is good, so if a transmitter finds a dominant state in this position the trasnmitter concludes that message sent was successfully, but if the trasnmitter finds a recessive state, then it undesrtand that message has an error, and it is manage by the receiving node, what follows is the end of frame which is actually a collection of 7 recessive states and then comes 3 bits inter frame spacing, which is another 3 recessive state).



- **The albitration Field:** this field actually contains an arbitration ID, in which, 11 bits is called as an identifier and there is another bit called RTR bit, that means request mode, so collectively we call it as arbitration field or abitration ID.
- This field determines the priority of the message when 2 or more nodes are contending for the bus. The arbitration field contains: For CAN 2.0A, an start of frame (SOF), an 11-bit identifier and one bit, the RTR bit, which is dominant for data frames. Identifiers stablishes the priority of the message. The lower the binary value, the higher its priority.



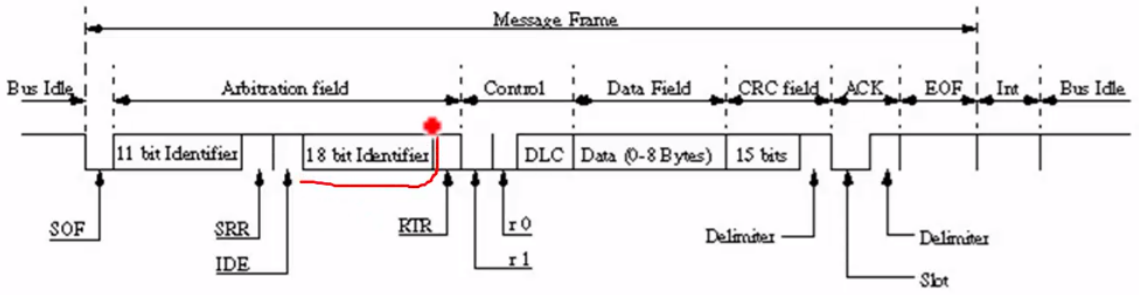
- When a frame contains a 11-bit identifier then it is called as Standar CAN frame, meaning that there are 2 types of CAN frames or controllers: the **Standar CAN** and the **Extendent CAN**

Standard CAN Vs Extended CAN

- The original specification is the Bosch specification Version 2.0
- Version 2.0 specification is divided into two parts
  - **Standard CAN (Version 2.0A). Uses 11 bit identifiers.**
  - **Extended CAN (Version 2.0B). Uses 29 bit identifiers.**
- The difference between these two formats is that the length of bits, i.e., the standard CAN Frame format supports 11-bits length for the identifier, whereas the extended frame supports 29-bits length for the identifier, which is made up of 18-bit extension and an 11-bit identifier.
- Most 2.0A controllers transmit and receive only Standard format messages.
- 2.0B controllers can send and receive messages in both formats.

- **STANDARD CAN VS EXTENDEN CAN**

Extended Frame Format or CAN 2.0 B



The IDE bit differs CAN extended frame format and the CAN standard frame format wherein IDE is transmitted as dominant in an 11-bit frame case and recessive in a 29-bit frame case.

- Standar CAN (Version 2.0A). Uses 11 bit identifier
- Extended CAN (Version 2.0B). Uses 29 bit identifier (it has an extension of 18 bits)
- Most 2.0A controllers transmit and recieve only Standard format messages.
- 2.0B controllers can send and recieve messages in both formats.

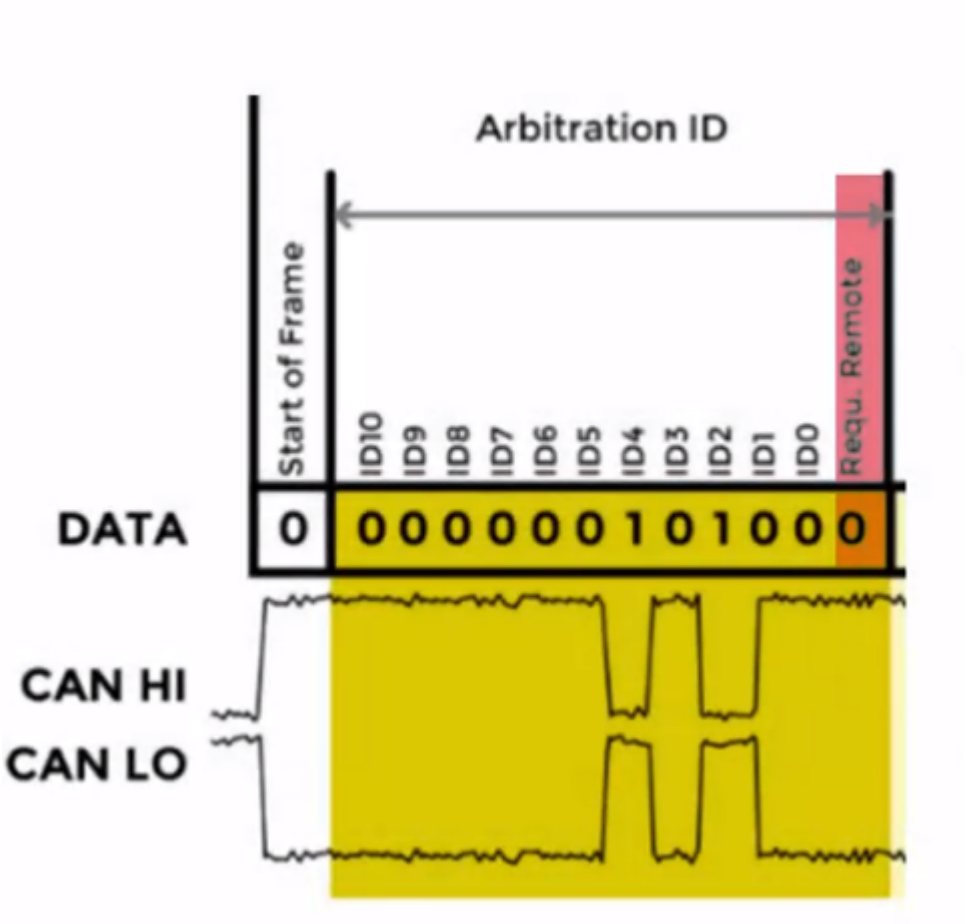
Standard CAN Vs Extended CAN

- If you have a CAN network which Consists of both 2.0A and 2.0B based CAN devices and if you use extended frame format (29 bit identifiers), then your network will not work, because 2.0A devices will generate an error

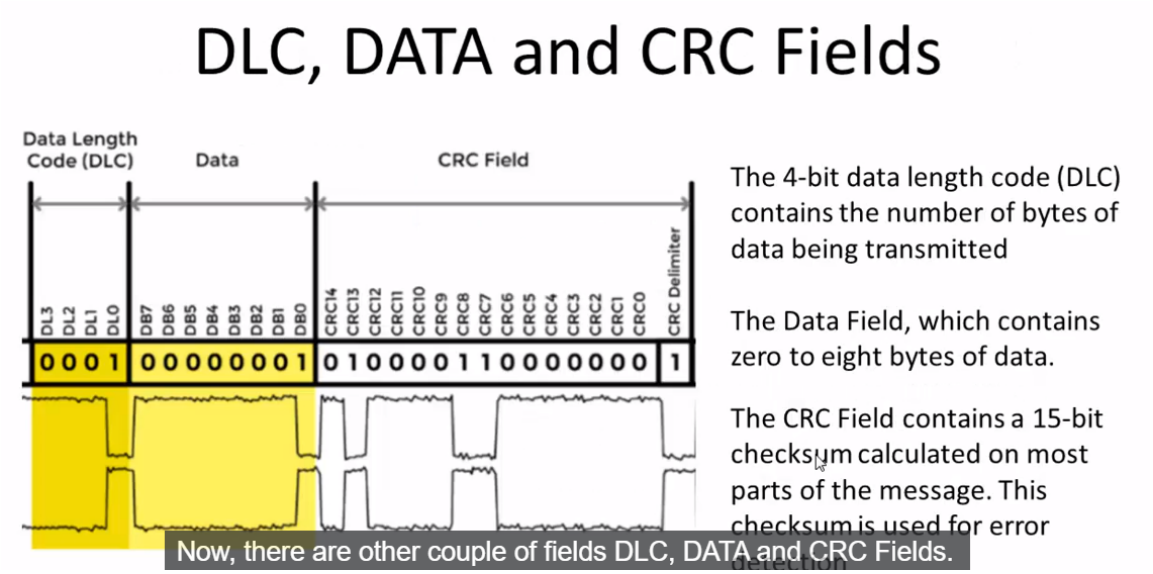
- 2.0B controllers are completely backward compatible with 2.0A controllers and can transmit and receive messages in either format.

- 2.0A controller based devices are capable of transmitting and receiving only messages in 2.0A format (standard format). With this type of controller, reception of any 2.0B message will flag an error.

- **CAN MESSAGE FORMAT EXPLANATION: ACK bit**
- RTR (Remote Transmission Request)
- A dominant (logic 0), RTR bit indicates that the message is a data frame.
- A recessive (logic 1) value indicates that the message is a Remote Transmission Request (known as a Remote Frame). A remote Frame is a request by one node for data from some other node on the bus. Remote frames does not contain a Data field.

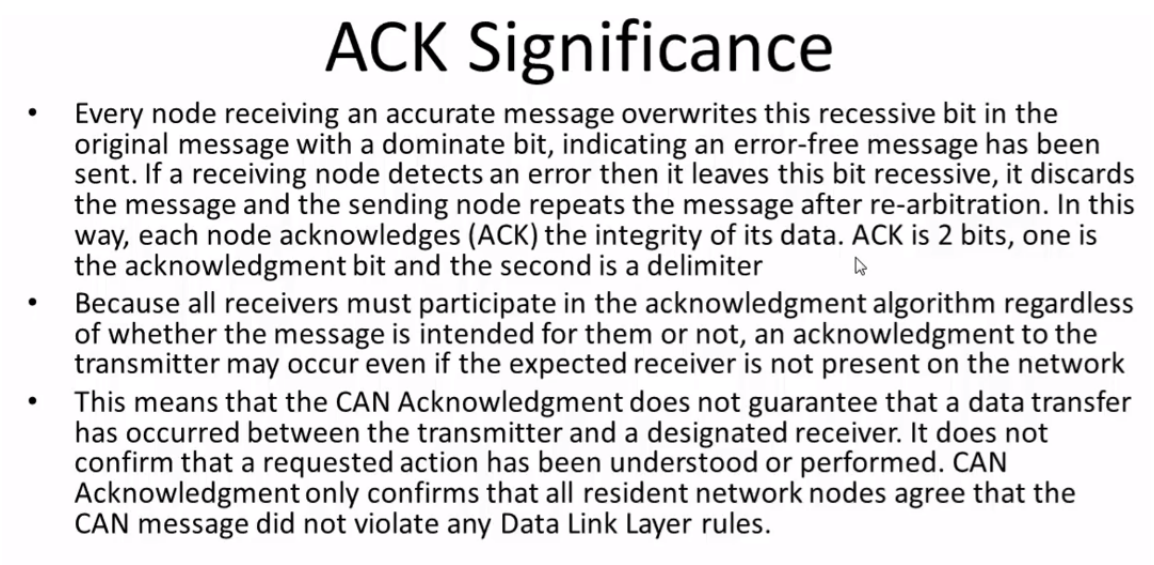


- **DLC, DATA and CRC fields**
- The 4-bit data length code (DLC) contains the number of bytes of data being trasnmitted.
- The data field, which contains zero to eight bytes of data.
- The CRC field contains a 15-bit cheksum calculated on most parts of the message. This cheksum is used for error an detection.



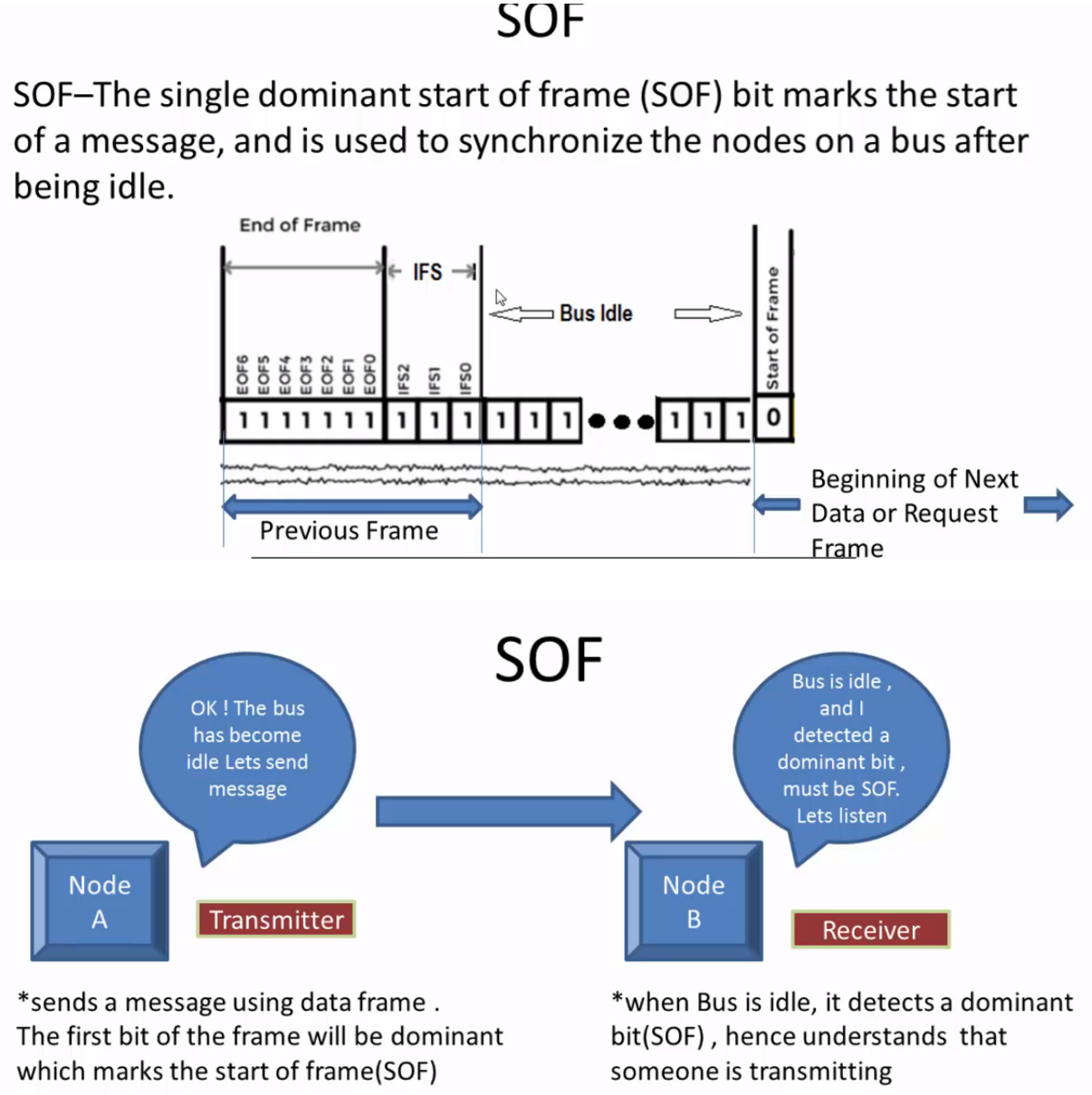
- As soon as CAN reciever starts receiving a frame it also calculates the CRC and then it maches with the CRC sent by the trasnmitter.
- If everything is fine, if a frame does not violate any CAN specifications rules
- If CRC match is successful then receiving node make bus state as dominant exactly at the ack slot which indicates the transmitter that at least one node received the frame correctly. If the trasnmitter sees recessive state of the bus at ack slot, then it understands that it’s an error and retransmits the message.

It is worth noting that the presence of an Acknowledgement Bit on the bus does not mean that any of the intended devices has received the message.
It just means that one or more nodes on the bus has received it correctly and Transmitter concludes that message sent successfully .
If Transmitter sees recessive state at the ACK slot , then it retransmits the message until it sees dominant state. That's the reason when there is only one node on the bus, transmitter keep sending the same message since no one is there to ack it.



- **END of Frame and IFS**
- After the ack bit, what follows is the end of frame, the EOF is actually a collection of 7 recessive states. And after that comes the IFS (Inter Frame Spacing) which is a collection of 3 recessive states.
- 10 recessive states (EOF + IFS) forms tail of a CAN frame, after this only a node whi is listening to bus understand that bus is idle. Any dominant bit heard after IFS is considered as SOF (Start of Frame).

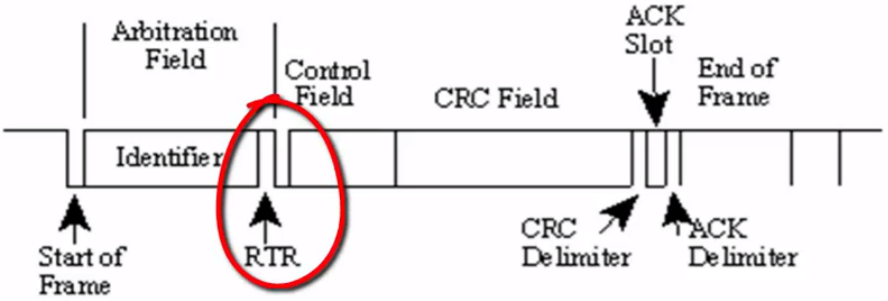
- After the 10 recessive states the bus becomes idle and then it can recieve or put data and in order to do that, it has to put SOF first, since is the single dominant start of frame bit which marks the start of a message, and is used to synchronize the nodes on a bus after being idle



- **REMOTE FRAME**
- The intended purpose of the remote frame is to solicit the transmission of data from another node.
- The remote frame is similar to the data frame, with 2 important differences. First, this type of message is explicitly marked as a remote frame by a recessive RTR bit in the arbitration field, and secondly, there is no data.

Remote Frame

RTR bit is Recessive and there is no Data



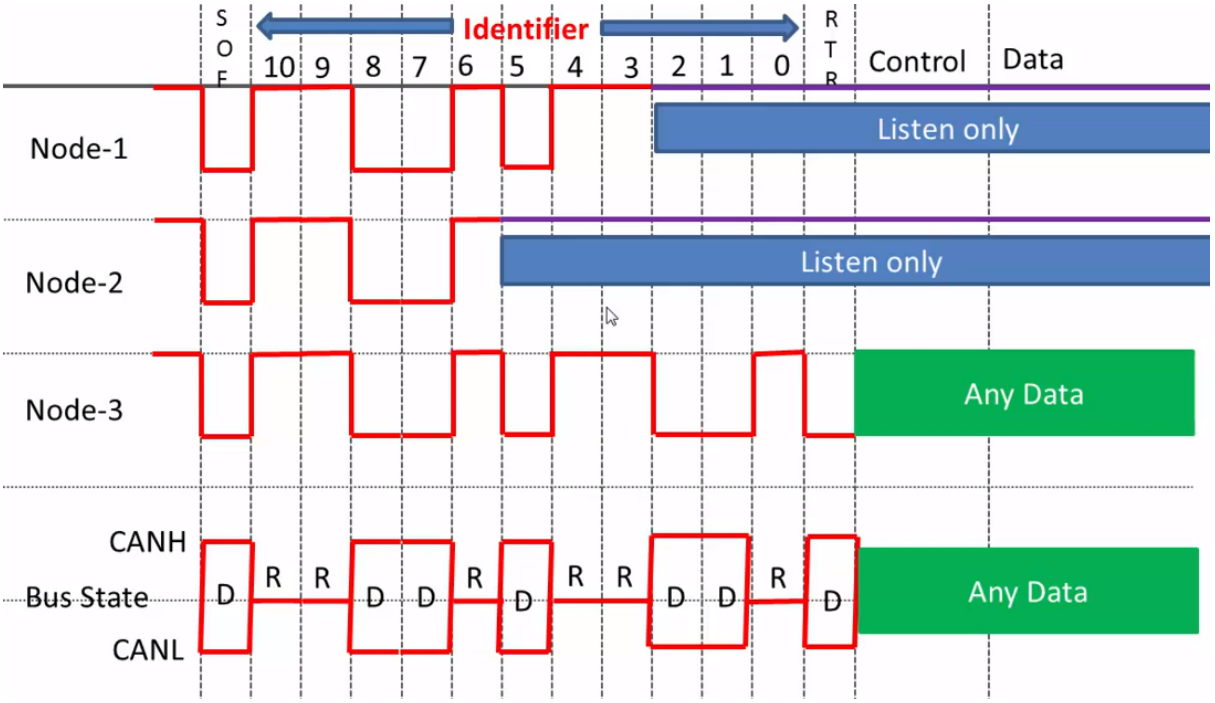
- **CAN BUS ARBITRATION**

BUS access from Multiple Nodes

The CAN communication protocol 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 preprogramed priority of each message in the identifier field of a message

The higher priority identifier always wins bus access.



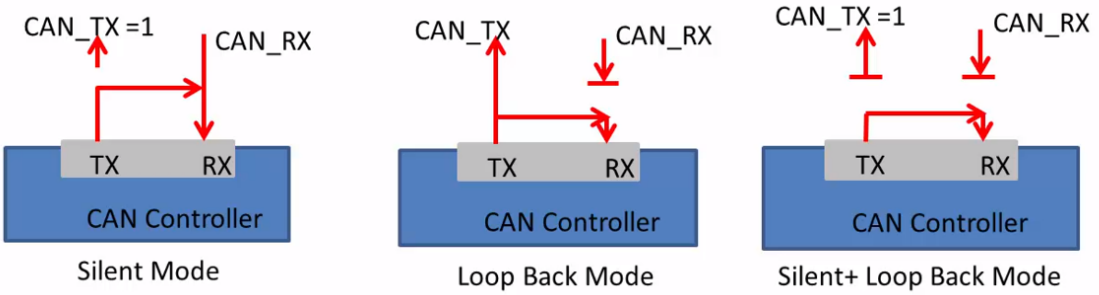
▼ Working with STM32 bxCAN Peripherals

- When we are testing CAN Tx and Rx fiunctioanlity in loopback mode, it means that there will only be one CAN peripheral on the bus
- The bxCAN (Basic Xtendedn CAN) modules handles the trasnmission and the reception of CAN messages fully autonomously. Standard identifiers (11-bit) and extended indentifiers (29-bits) are fully supported by the hardware.

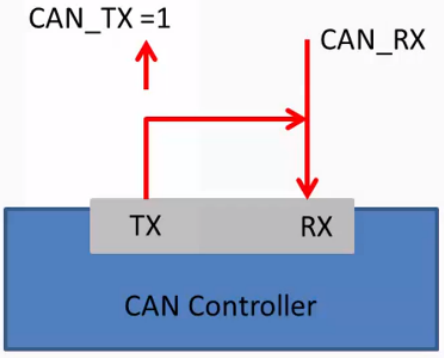
What application can do with bxCAN ?

- Configure CAN parameters, e.g. bit rate, bit timings , etc
- Transmissions
- Handle receptions
- Manage interrupts
- Get diagnostic information

BxCAN Test Modes

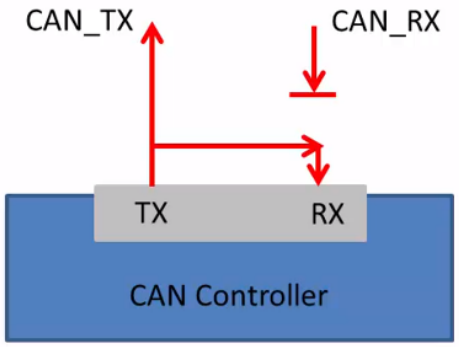


Silent Mode

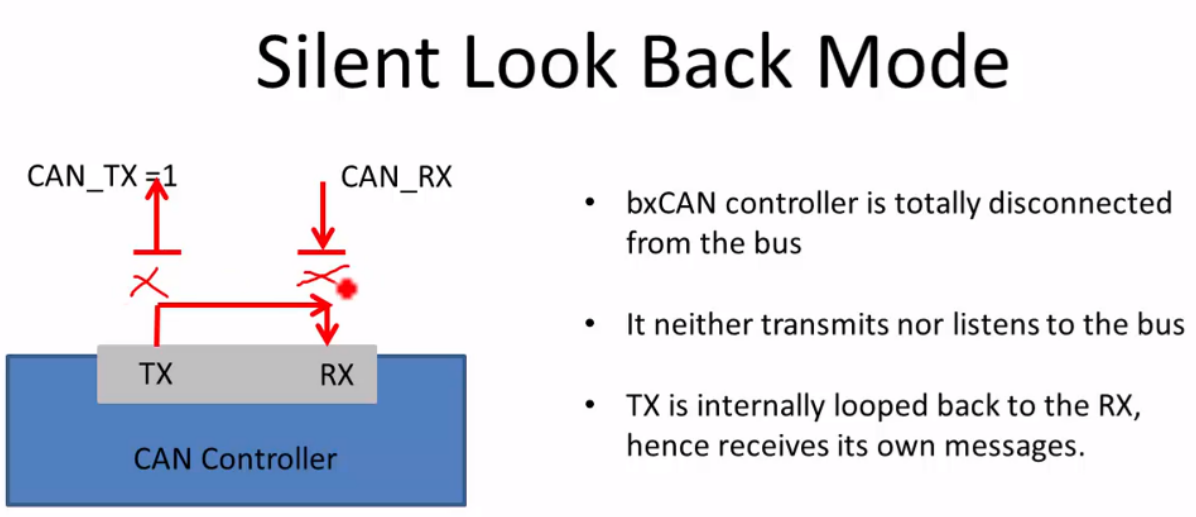


- Tx line is internally looped back to RX line
- CAN TX is held at recessive state
- bxCAN is able to receive valid frames
- It just listens and doesn't change the bus state by putting dominant bit
- Can be used as a sniffer which just analyzes the traffic on the bus.

Loop Back Mode



- bxCAN can Transmit frames on the bus .
- Also the frames are looped back to the RX line internally
- bxCAN will not listen to the bus, but just receives its own message which is looped back.
- loop back mode is provided for self test functions



Information from Datasheet about STM32F407VGT

- Some of the features that bxCAN (basic extended CAN) has are: it can support CAN protocol version 2.0A and B, it can have a speed rate up to 1Mbit/s and it supports the Time Triggered Communication option.
- For the trasmission we have 3 mailbox, it can be configurable for transmit prority and the Time Stamps on SOF transmission.
- For the reception we have 2 receive FIFOs with 3 stages each, 28 scalable filter banks (shared between CAN1(master) and CAN2 (salve)).
- Dual CAN. CAN1: Master bxCAN for managing the communication between a Slave bxCAN and the 512-byte SRAM memory.
- CAN2: Slace bxCAN, with no direct access to the SRAM memory.
- The 2 bxCAN cells share the 512-byte SRAM memory

Figure 334. CAN network topology

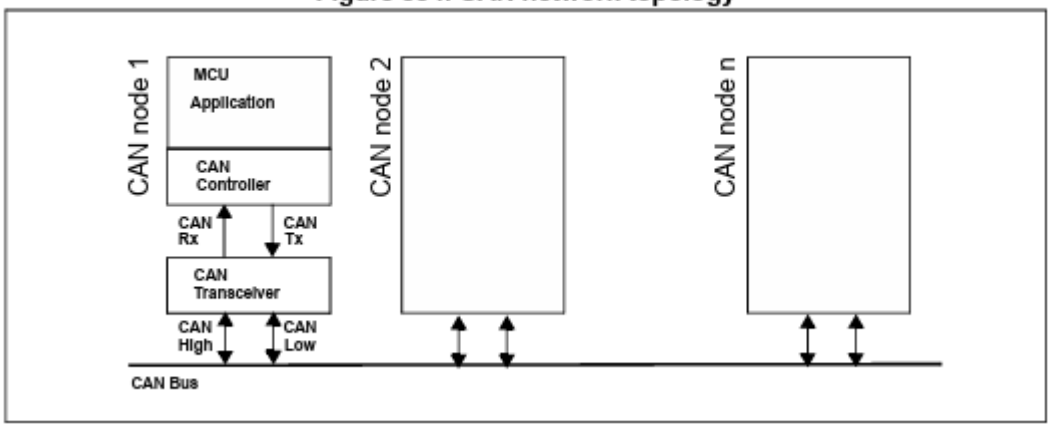
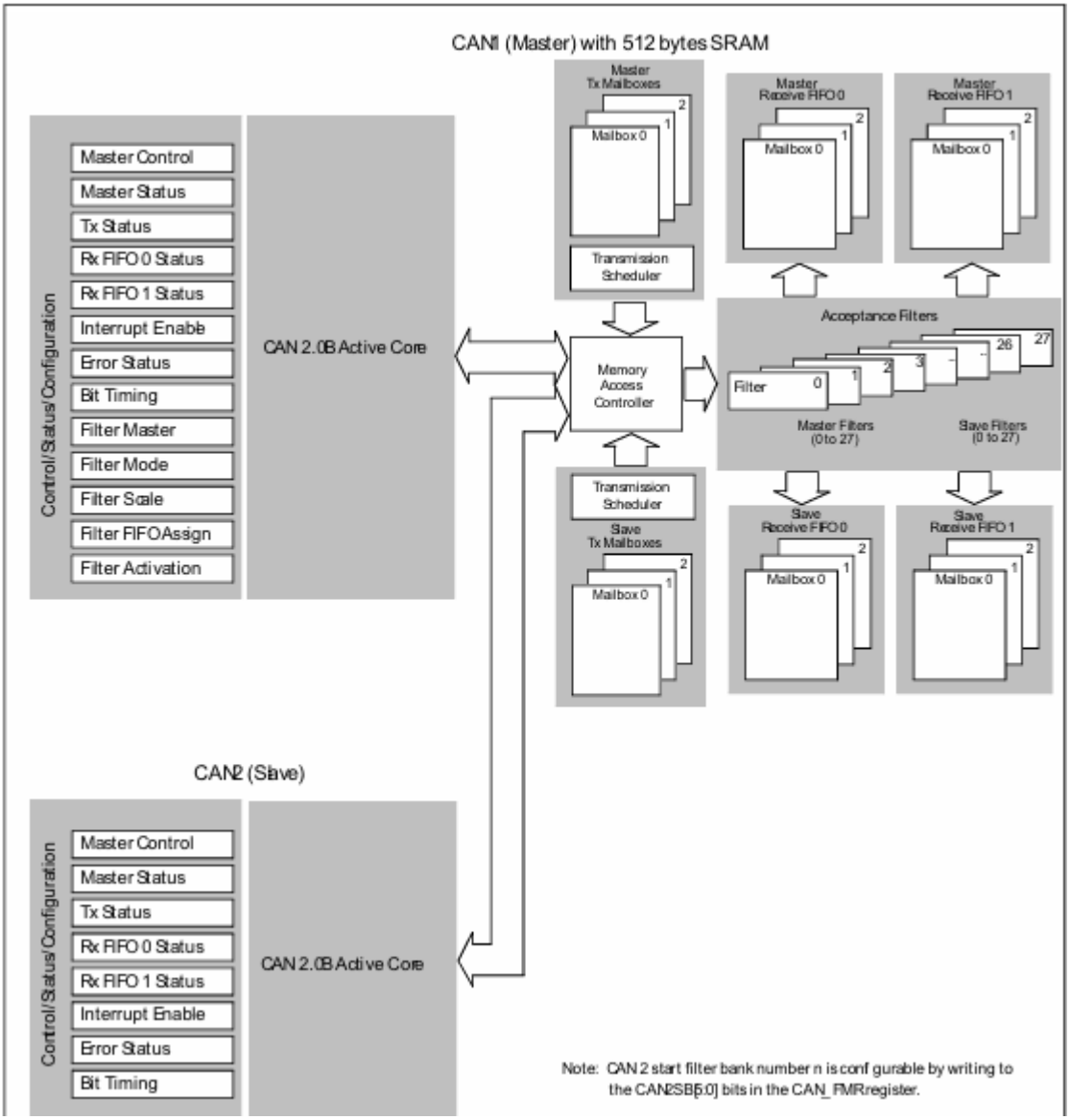


Figure 335. Dual CAN block diagram



- In the block diagram above we can see how the control status and configuration are divided between the CAN1 (master) and CAN2 (slave), an important thing to consider is that if we want to use any of the control/status configuration of the Slave CAN, we first need to activate Master, also, these are controled by the MCU, this alsohas access to the trasnmitter schedule that has the mailbox boxes, this schedule will send first the message with the highes priority (the one with the lowes ID), we also see the FIFOs that are connected to the filter banks.
- CAN1 being the master for managing the communication between salve bxCAN and the 512 bytes of SRAM.
- **OPERATING MODES**
- bXCAN has 3 operating modes: **initialization, normal and sleep** after a hardware reset, bcCan enter in sleep mode, for power consumption and an interal pull-up is active on Tx. Then the software request bxCAN to enter **initialization** or **sleep** mode by setting the INRQ or SLEEP bits in the CAN_MCR register. Once the mode has been entered, bxCAN confirms it by setting the INAK or SLAK bit in the CAN_MSR register and the internal pull-up is disable. When neither INAK nor SLAK are set, bxCAN is in normal mode.
- But, before entering normal mode, bxCAN always has to synchronize on the CAN bus, to do that bxCAN waits untill the CAN bus is idle, this means 11 consecutive recessive bits have been monitored on CANRX.
- **INITIALIZATION MODE**

The software initialization can be done while the hardware is in Initialization mode. To enter
this mode the software sets the INRQ bit in the CAN_MCR register and waits until the
hardware has confirmed the request by setting the INAK bit in the CAN_MSR register.

To leave Initialization mode, the software clears the INRQ bit. bxCAN has left Initialization
mode once the INAK bit has been cleared by hardware.

While in Initialization Mode, all message transfers to and from the CAN bus are stopped and
the status of the CAN bus output CANTX is recessive (high).
Entering Initialization Mode does not change any of the configuration registers.

To initialize the CAN Controller, software has to set up the Bit Timing (CAN_BTR) and CAN
options (CAN_MCR) registers.

To initialize the registers associated with the CAN filter banks (mode, scale, FIFO assignment, activation and filter values), software has to set the FINIT bit (CAN_FMR). Filter
initialization also can be done outside the initialization mode.

- **NORMAL MODE**

Once the initialization is complete, the software must request the hardware to enter Normal
mode to be able to synchronize on the CAN bus and start reception and transmission.

The request to enter Normal mode is issued by clearing the INRQ bit in the CAN_MCR
register. The bxCAN enters Normal mode and is ready to take part in bus activities when it
has synchronized with the data transfer on the CAN bus. This is done by waiting for the
occurrence of a sequence of 11 consecutive recessive bits (Bus Idle state). The switch to
Normal mode is confirmed by the hardware by clearing the INAK bit in the CAN_MSR
register.

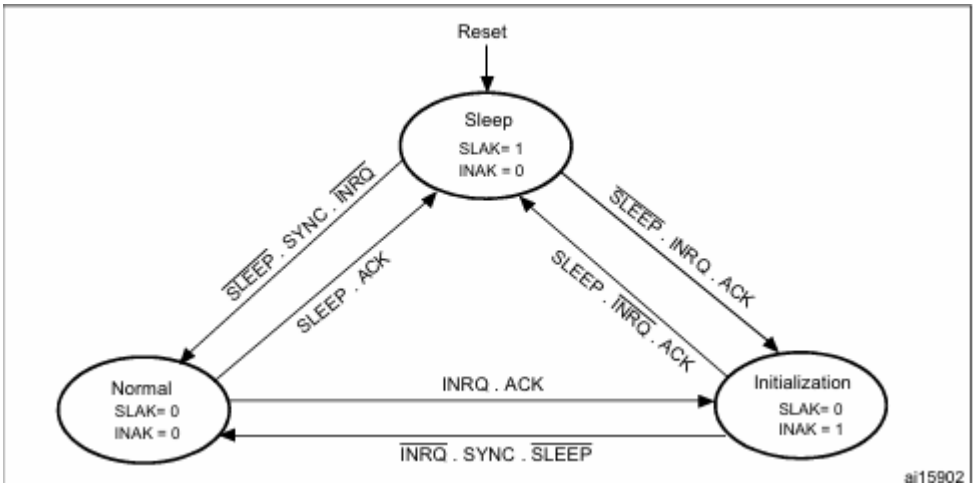
The initialization of the filter values is independent from Initialization Mode but must be done
while the filter is not active (corresponding FACTx bit cleared). The filter scale and mode
configuration must be configured before entering Normal Mode

- **SLEEP MODE (LOW POWER)**

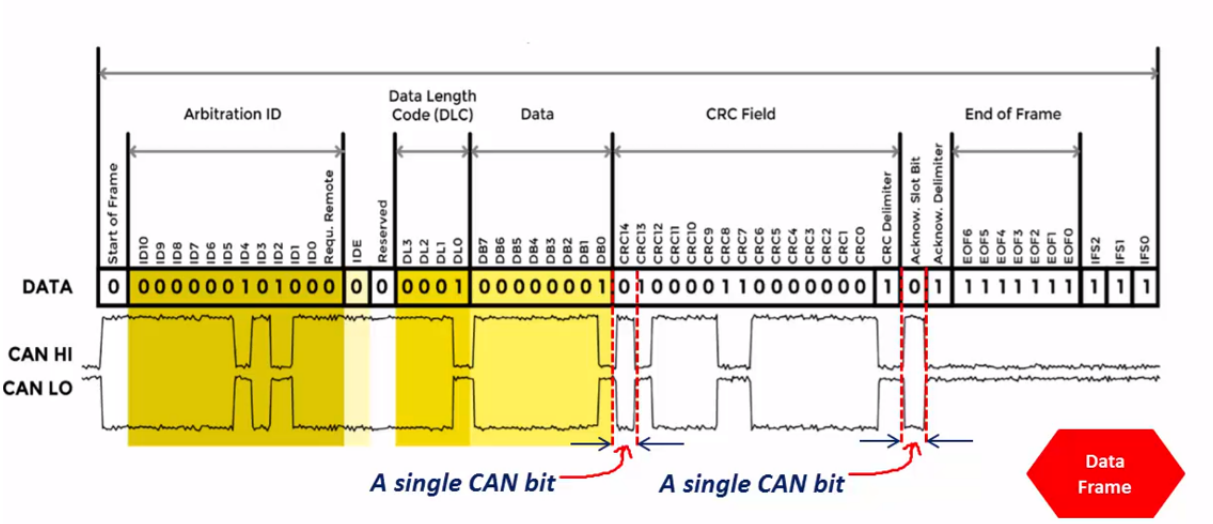
To reduce power consumption, bxCAN has a low-power mode called Sleep mode. This
mode is entered on software request by setting the SLEEP bit in the CAN_MCR register. In

this mode, the bxCAN clock is stopped, however software can still access the bxCAN mailboxes.
If software requests entry to initialization mode by setting the INRQ bit while bxCAN is in
Sleep mode, it must also clear the SLEEP bit

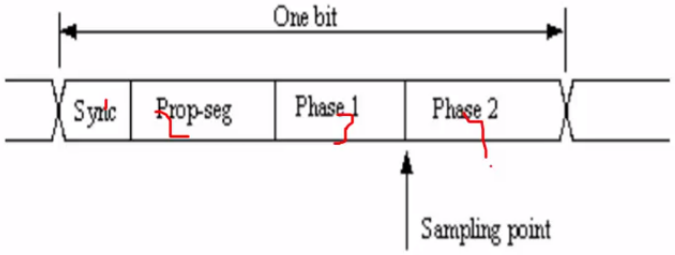
bxCAN can be woken up (exit Sleep mode) either by software clearing the SLEEP bit or on
detection of CAN bus activity.
On CAN bus activity detection, hardware automatically performs the wakeup sequence by
clearing the SLEEP bit if the AWUM bit in the CAN_MCR register is set. If the AWUM bit is
cleared, software has to clear the SLEEP bit when a wakeup interrupt occurs, in order to exit
from Sleep mode.



1. ACK = The wait state during which hardware confirms a request by setting the INAK or SLAK bits in the
   CAN_MSR register
2. SYNC = The state during which bxCAN waits until the CAN bus is idle, meaning 11 consecutive recessive
   bits have been monitored on CANRX



CAN Bit Timings Configuration



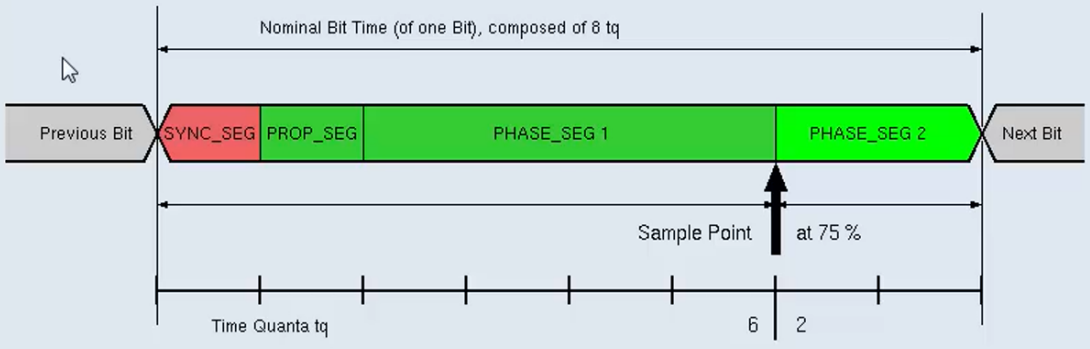
Each bit on the CAN bus is, for timing purposes, divided in to 4 segments

1) Synchronization Segment
2) Propagation Segment
3) Phase Segment 1
4) Phase Segment 2

Width of these segments have to be adjusted properly to get desired bit rate on the CAN bus. Width of each segment is mentioned in terms of time quanta

The Time Quanta is the smallest time unit for all configuration values.

The question arises: what exactly is the **sample point**? The picture shows you the time segments of a CAN-Bit as defined by ISO-11898.



- There are single CAN bits between, theses bist can be either recessive or dominant, to find the width of that we need to know the clock frequency we are going to use.
- That single or one bit duration of the CAN message/protocol is divided into 4 segments (for timing porpuses):
  - Synchronization Segment
  - Propagation Segment
  - Phase Segment 1
  - Phase Segment 2
- Thw width of these segments need to be adjust properly to get the desired bit rate on the CAN bus (each in terms of time quanta —> Tq)
- **Time Quanta is the smalles time unit for the all configuration values and is required for the correct operation of the CAN bus (for both trasmission and reception)**
- So a bit is actually a collection of all these segments or number of time quanta.
- Note: If we increase the bit duration by including more and more Tqs for each segment, we may minimize the trasnmission error and signal can be carried over longer distances but but throughput decreases.
- Note2: If your shrink the bit duration by including lesser Tqs for each segment, then messages may not be carried over longer distances but bus throughputt increases.

What is CAN?

- CAN stands for Controller Area Network
- CAN basically is a multi-master, message broadcast system and is a standar of communication
- The ISO_11898: 2003 standardized the CAN communication protocol and fenied it as: “Serial communication but to replace the complex wiring harmess with a two-wire bus”.
- It has 3 layers,