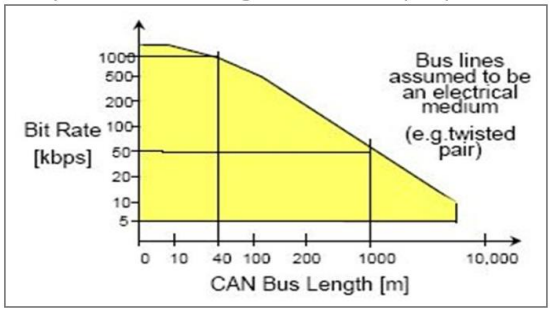
Controller Area Network

Table des matières

CAN Protocol

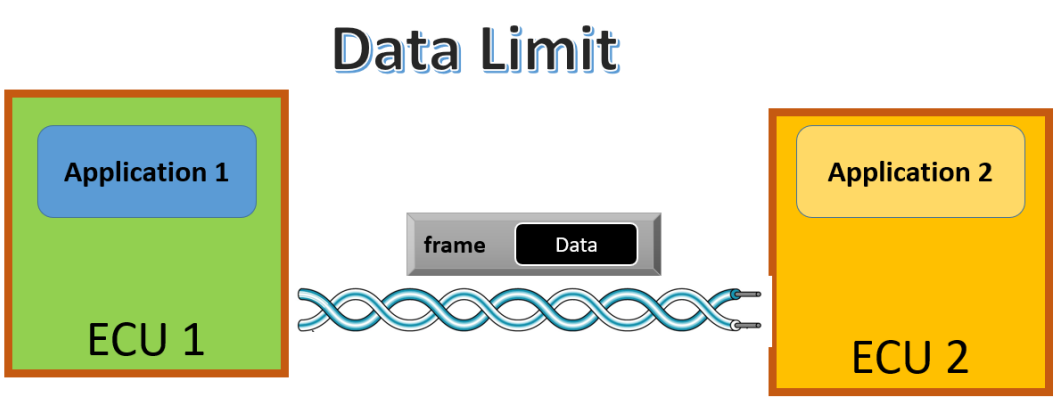
1. CAN SPEED

❑CAN frames ca go up to 1mbps speed with the bus length at max of 40 mts.
❑As the bus length increases above the maximum speed reduces.
❑In industry usually the CAN is configured at 500kbps speed



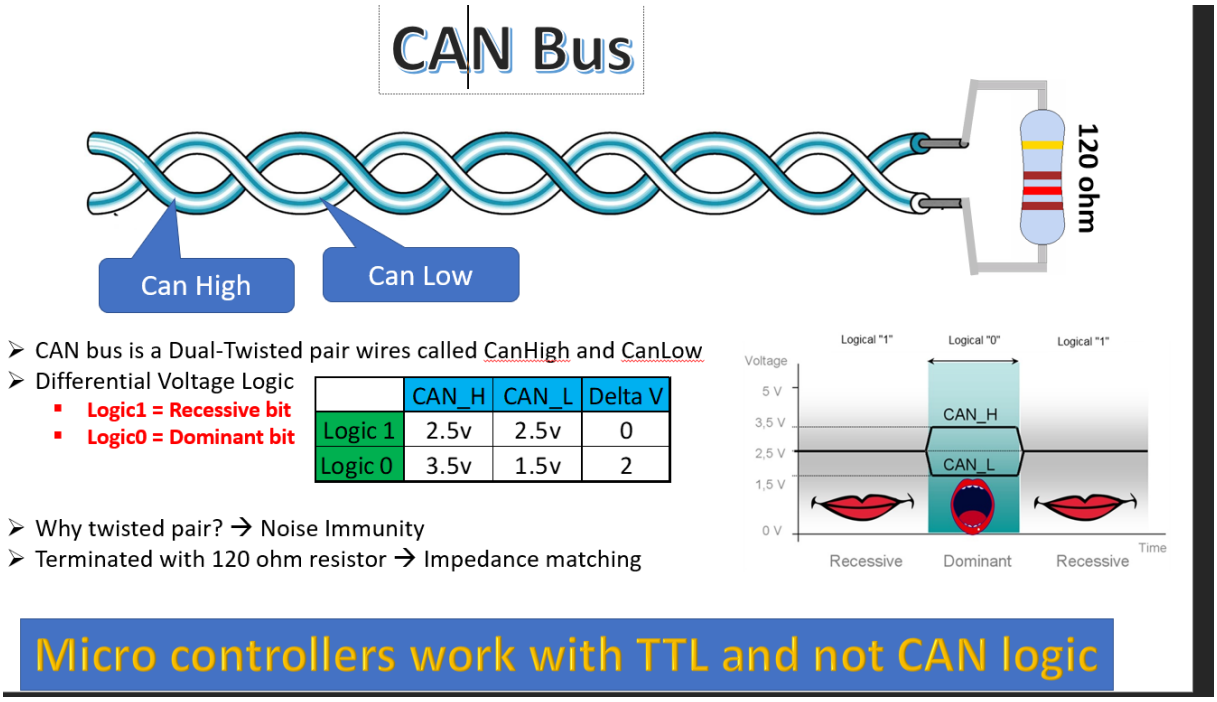
❑Lets say the baud rate is 500kbps → In 1 second, 500 *1000 bits can be transmitted
❑Lets say a single frame takes about 125 bits for example (without stuffing 111 bits)
❑So in a second the bus can fit around 2000 (500k /125) frames

2. Data limité



❑Frame has 2 parts: PCI and Data
❑As per CAN protocol How much data can be sent in each frame? → 8 Bytes
❑Is the data size variable? → Yes we can have 0 bytes or 1 byte or 2 bytes etc. Max is 8 bytes.
❑Is it possible to have data in bits? → No. Its in bytes. We cant have 2 bit or 27 bits etc its always in bytes.

3. CAN Bus



	CAN_H	CAN_L	Delta V
Logic 1	2.5v	2.5v	0
Logic 0	3.5v	1.5v	2

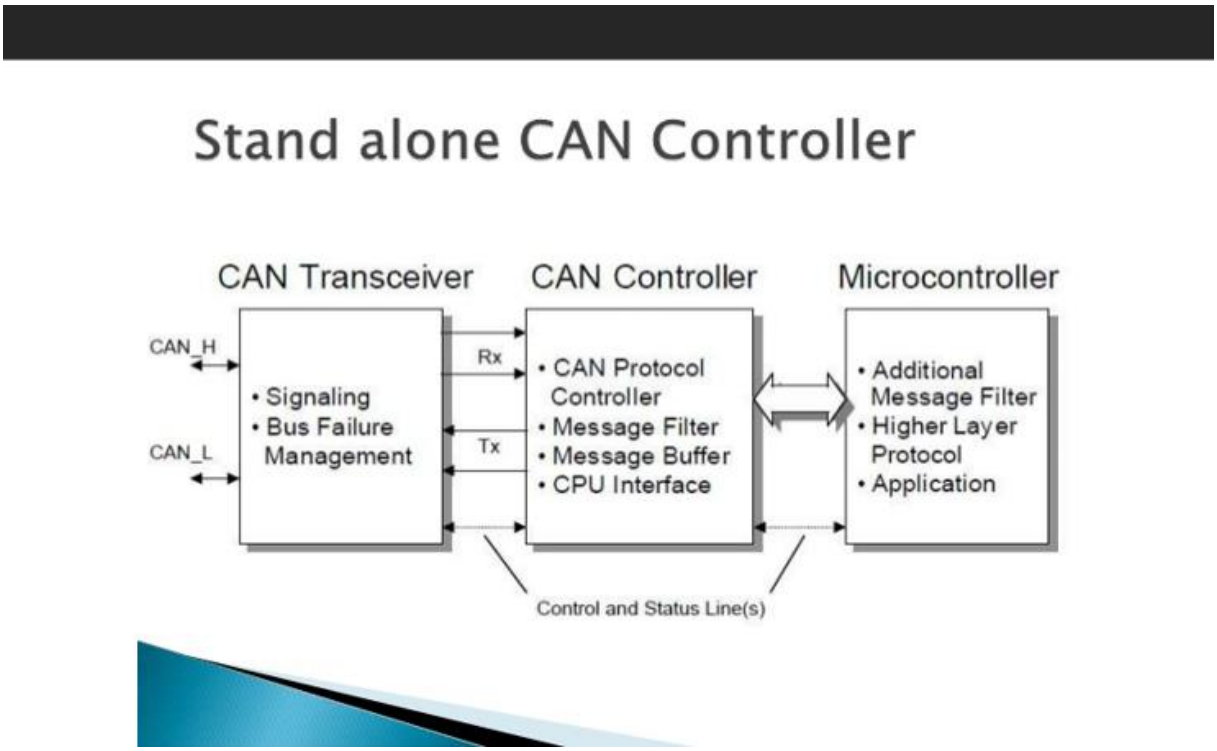
Notes : the two twisted buses always end with resistor=120 OHM

1- Delta V is the difference between the CAN_H and CAN_L
2- Logic0 is dominate because it has the priority
3- CAN is not sensibale to the noises because:

Supposed the we have a voltage noise Vn: so CAN_H=VH+Vn and CAN_L=VL+Vn

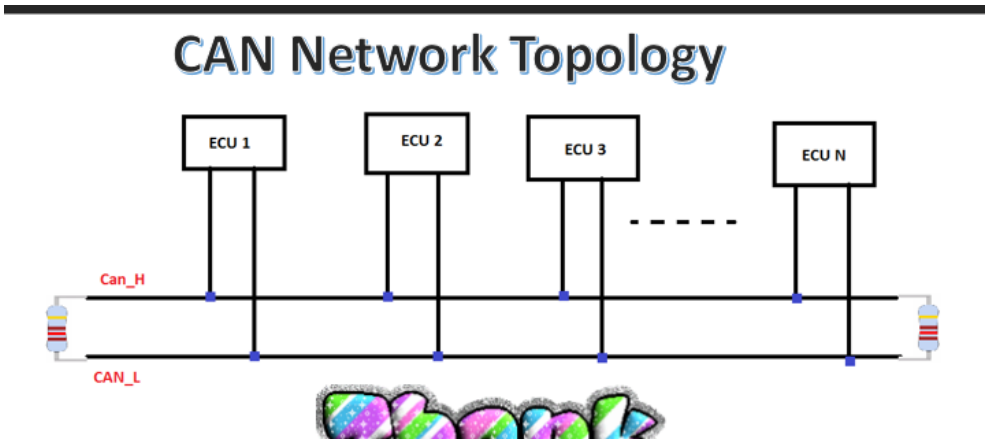
Which means that the difference between them nothing but Delt_V

4- The 120 ohm is to optimize the signal reflection.
5- The ECU work with TTL(0-5V) Logic and not the Can Logic that why we will use the can transceiver



6- The CAN transceiver(physiq la,yer) convert from the CAN logic to th TTL logic, and the Can controller(Hardwarem, Network layer) form the Can Frame but in the TTL bit

4. CAN Network Topology



1- It's a bus topology
2- The CAN busses and the ECUS forme a CAN Network
3- If the speed of the CAN busses is equal to X Kbps all the ECUS must configure on the same value which is x Kbps

5. CAN priorities

1- CAN is Wire Protocol

2- CAN is serial communication
3- CAN is a Asynchrone communication protocol, the difference between the Synch and the Asyn : the first one if the node(ECU) has the data ready it cannot send it till it has the permission it could be in a Front Descendent or Front Montant, The Asynchron the node can send the data at any time it wants.
4- CAN is a Multicast protocol, Broadcast

Message Based Protocol

❑CAN protocol is message based protocol and not Node based protocol.

❑Each CAN frame is identified with Message ID and no info about the Tx or Rx node.

❑Node based Can be understood with mobile phones. Usually Peer-Peer are node based.

❑Since CAN is multicast, It is message based protocol.

❑Message ID has 3 purpose:
- Identify the Data carried by the frame
- Decides the message Priority and hence decides which frame gets the bus if clash happens.
- Used in Message Filtering

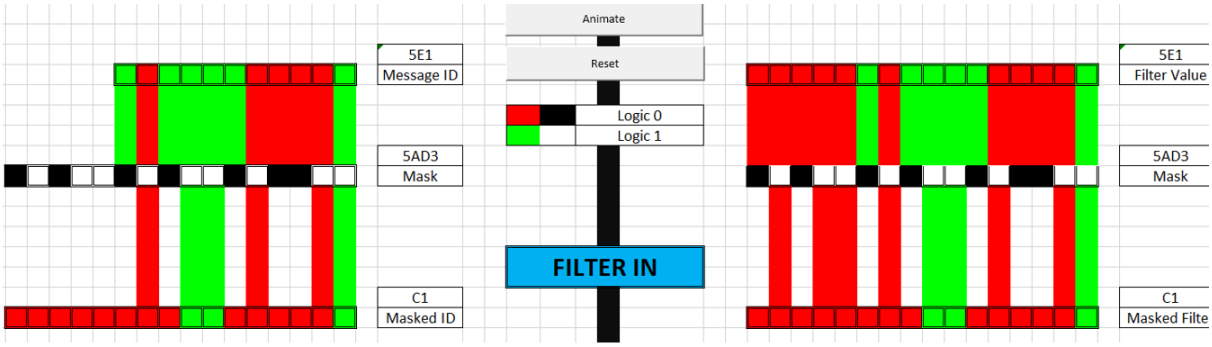
5- CAN is a message based protocol, because in the frame we don't know who is the receiver.
6- Each msg has something called msg ID and the msg ID has three porpuses :
- What kind of data carried or what is about
- The priority
- Filtring: if the id msg=X I wont let it pass

6. Message filtering

Message Filtering Logic

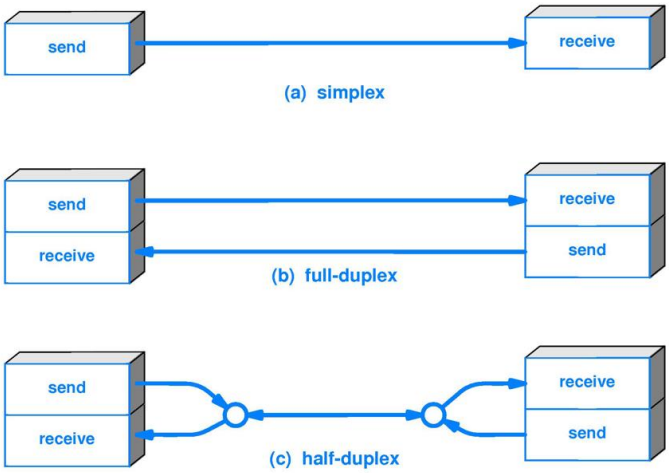
```
if (Message_ID & Mask) == (Filter_Val & Mask) {
        Let the Message IN
}
else {
        Ignore the Message
}
```

Msg ID : 5E1
Filter : 5E1
Mask : 5AD3



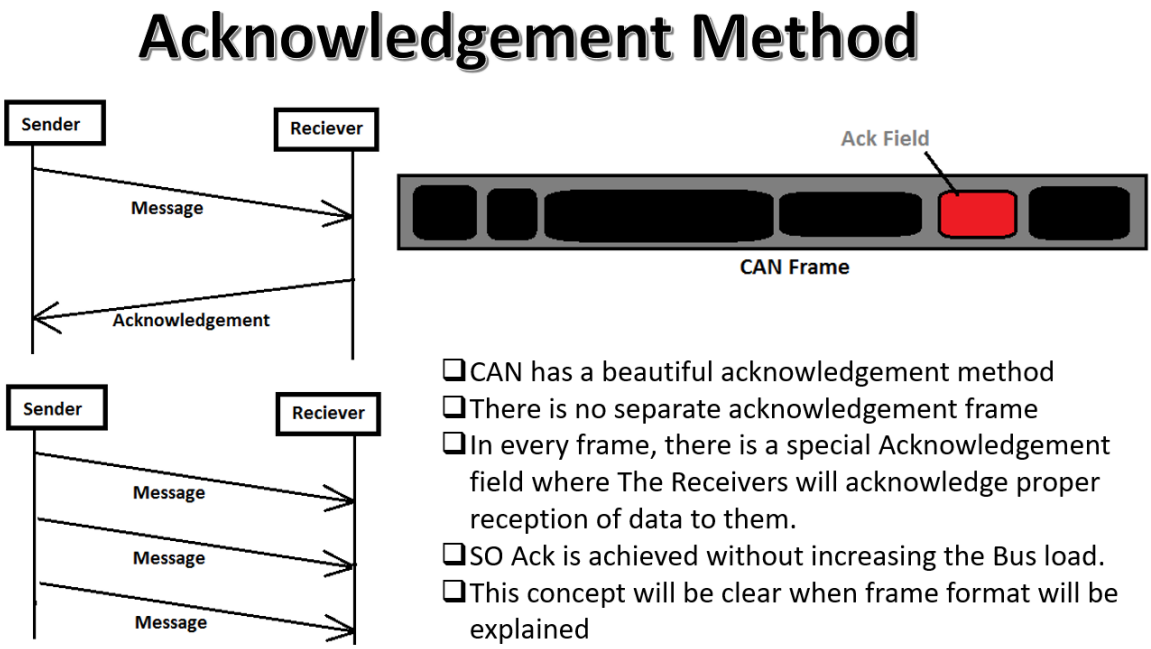
7- CAN is a Half-duplix,

Simplex – Half Duplex – Full Duplex



CAN Protocol is a Half-Duplex Communication.

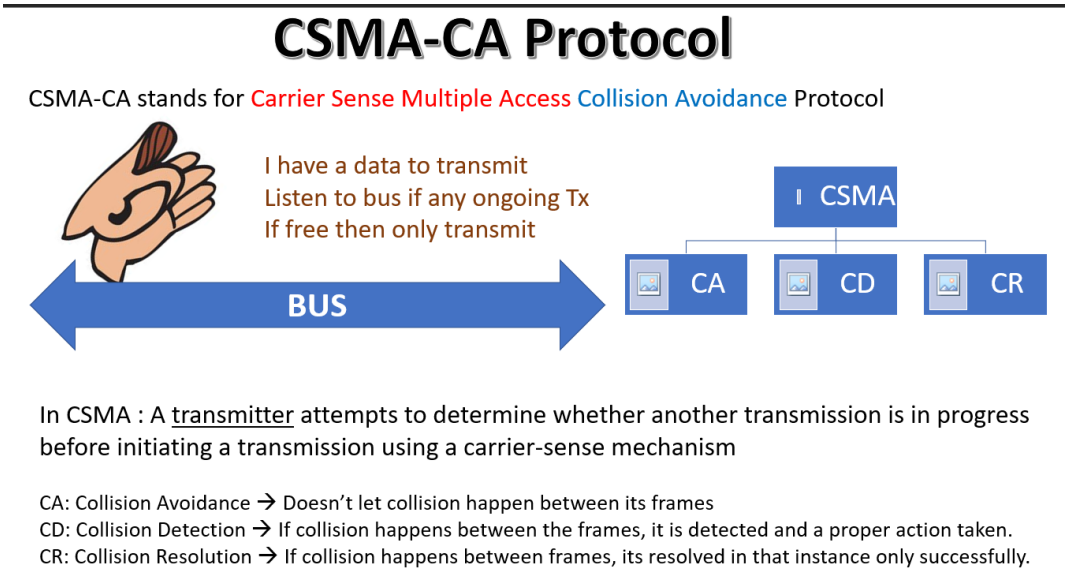
7. Acknowledgement method



- In the can frame we have the data and also the bit pf acknowledgement.

8. CSMA-CA Protocol

Carrier Sense Multiple Access Collision Avoidance Protocol



CAN is a CSMA-CA protocol

9. Type of Frames

They are 4 types of can Protocol,

- Data Frame
- Remote Frame
- Error Frame
- Overload Frame

1.2 Data frame

Data Frame

❑Frame carrying the data from transmitter node to all receiver nodes.

❑Serves the main purpose of CAN protocol

❑Two types of Data Frames
- ➢ Standard Frame : 11 bit message ID
- ➢ Extended Frame : 29 bit message ID {split as 11 bit Base ID + 18 bit ID Extension}

❑In CAN when we are normally talking of frames, we are referring Data frames only.

Frame carrying the data from the transmitter, noede tp all recieiver nodes. And ther is two types of data frame:

- Standards frame:11BIT msg Id
- Extended Frame:29 BIT MSG ID

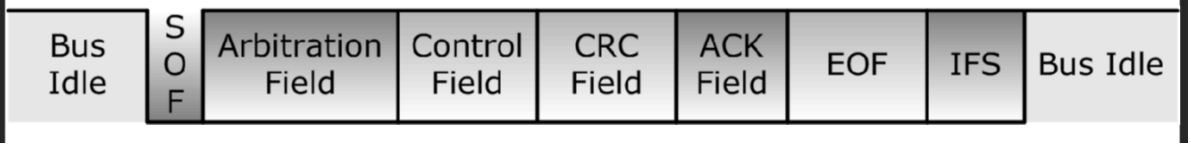
2.2 REMOTE frame

This frame requesting data frame,

IF a node wants a specific dataFrame, so it sends the Remote frame with same msg ID on the BUS

Remote Frame

- Frame Requesting for a Data Frame
- A node wants a particular Data frame, so It sends the Remote frame with same Message ID on the Bus.
- This remote frame is followed by the Data frame on the bus transmitted by the node who owns that data frame.
- Hence Remote frame sending node got the data it requested for.
- Now a days Remote frame is obsolete and not used in CAN protocol.



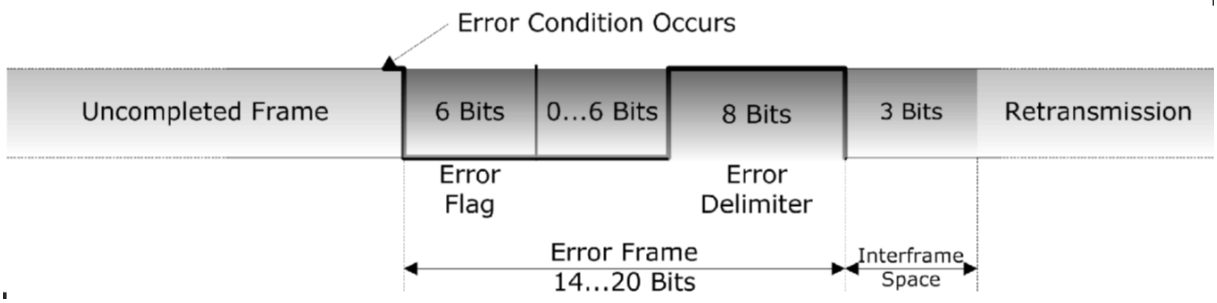
It s approximately just like data frame but only the data frame is missing

3.2 Error Frame

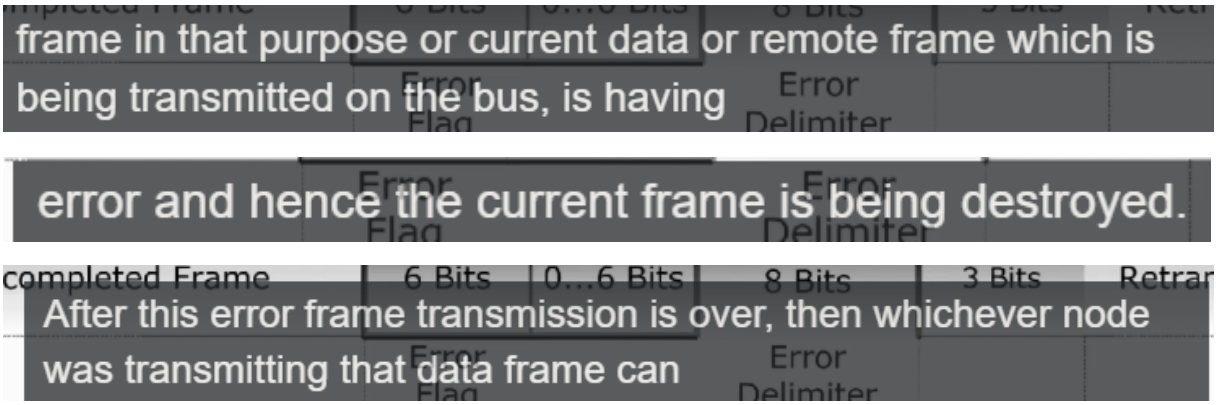
This frame signals an error condition in the data frame being transmitted currently.

Error Frame

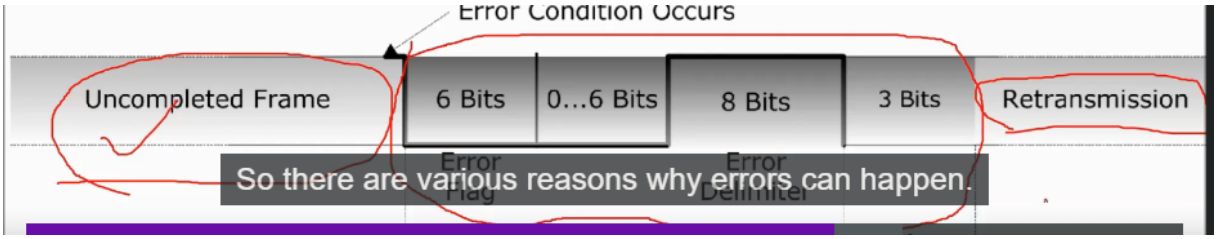
- Error Frame signals an error condition in the data frame being transmitted currently.
- If a node detects an error in the data frame being transmitted on the bus currently, then it destroys that data frame and signals all other nodes by transmitting a error frame.
- The error can be detected by any of the receiver nodes or the transmitter node.



Error frame is transmitted by a node to signal to all the nodes that the current data frame or remote



Retransmeted the same frame again



As u see in the pic upper the frame is detected as an error so the transmission of the frame in the bus stopped and after the destroying we have retransmission

4.2 Overload Frame

It's a frame transmitted by a node when it is overloaded and needs some time to process the data frame received previously.

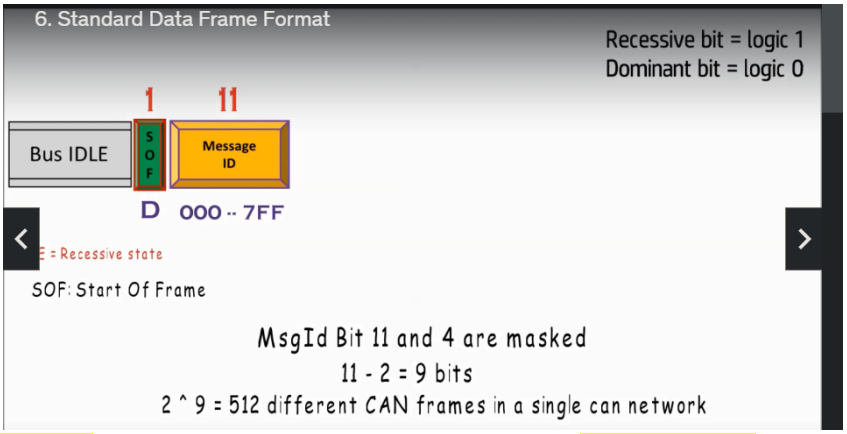
Overload Frame

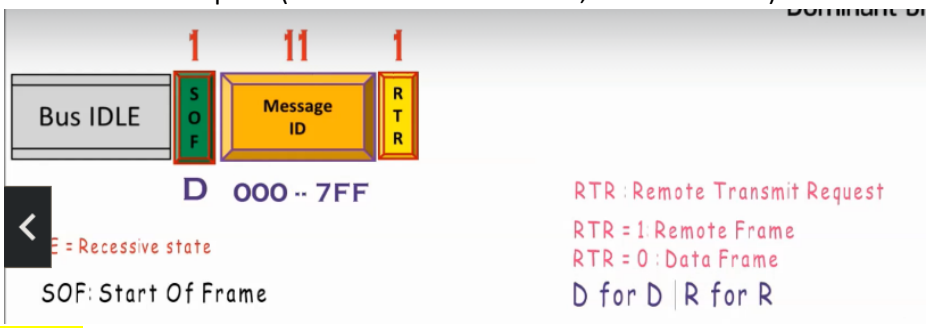
- ❑Overload frame is transmitted by a node when it is overloaded and needs some time to process the data frame received previously.
- ❑Overload frame format is same as that of an error frame.
- ❑Strategy is that since the overloaded node needs some time so it buys that time by keeping the bus busy with its overload frame preventing other nodes to start transmitting data frames.
- ❑Maximum of 3 consecutive overload frames can be transmitted by each node after a data frame.

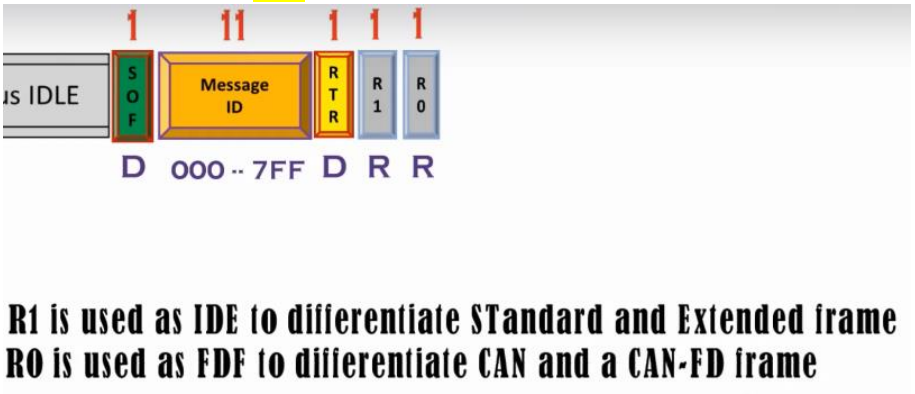
But now we have ecu that works in GH so this frame is very rare

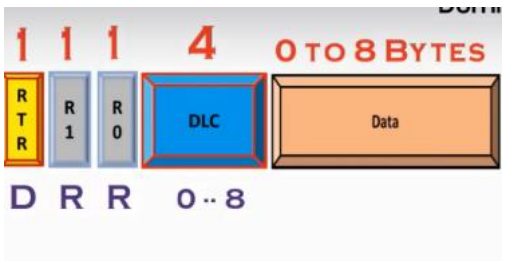
10. Can STANDART data frame format

- When there is no frame in the bus we say: BUS IDLE (Recessive state=logic 1)
- Every data frame start with a dominante bit SOF (start of frame), the important of this frame is to tell the nodes of a starting of frame on the bus and the bus is no longer IDLE

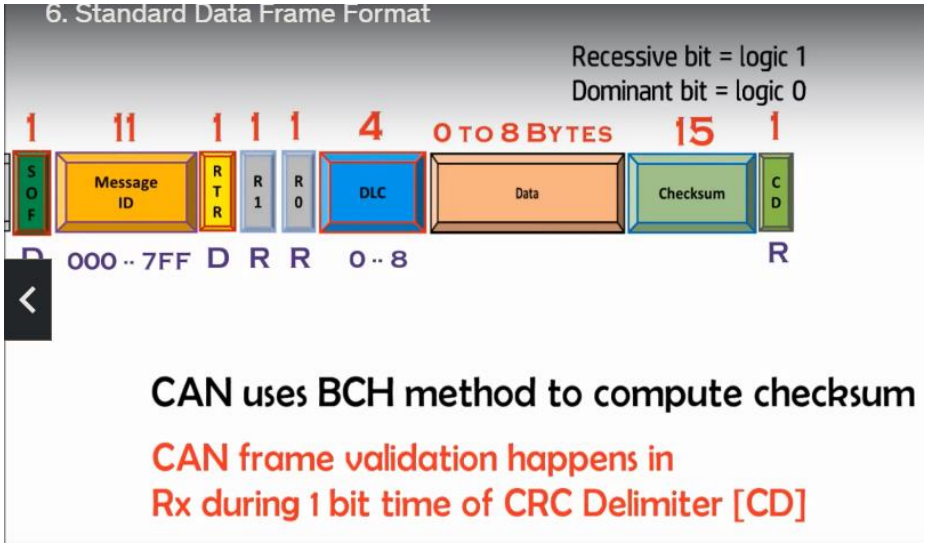
- THEN WE HAVE TO IDENTIFY THE FRAME BY Msg IDE (CAN is msg based Protocol), 11bits
- In the standar Can Frame we have 9 bits of the 11 Bits that can change and the are not fix, so we can have 2^9=512 Different Can Frame in the standard Can frame (000X to 7FFx)
- 

- after the msg ide we must tell the Bus if it is data frame or Remote Frame, becu zyou know the data frame and the remote frame have the same Msg ID for this we use RTR Bit
- RTR: remote transmut Request:(=1 means Remote frame, =0 Data Frame)
- 

- R1 and R0 bits:R1 used as IDE or difference between the Standard frame and the extended frame. R0 used as FDF to difference between a
- 

- DLC data length the Code, 4bits: tells how many bytes of data in the frame, u know the can frame contain 1 bits or 2 3 OR 4;;;;;8bits,
- Data: 0 to 8 bits
- 

- Checksum, 15 bits : detect if the bit changed due to noises for example
- CD 1bit: If the checksum number is the same we must validate the data frame for that we use the checksome Delimiter

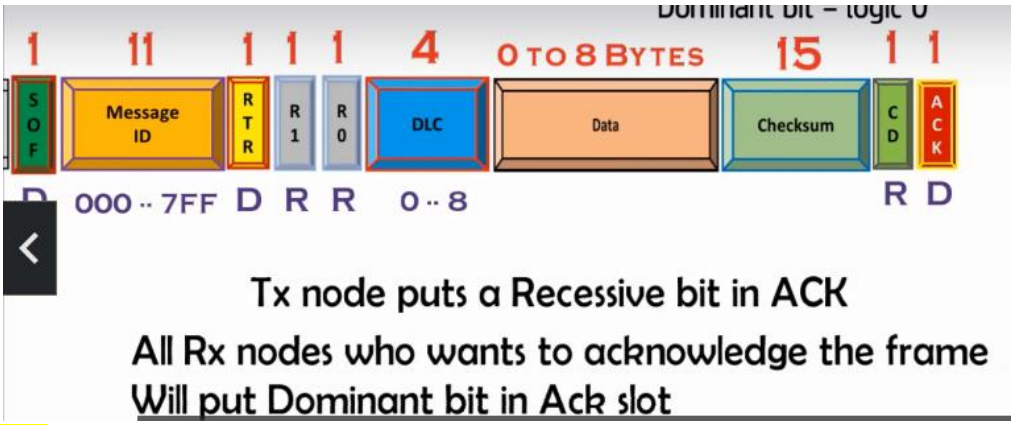


The only purpose of checksum delimiter bit is to provide some time for all the receiver nodes to validate

the received data and frame and be ready for the acknowledgement.

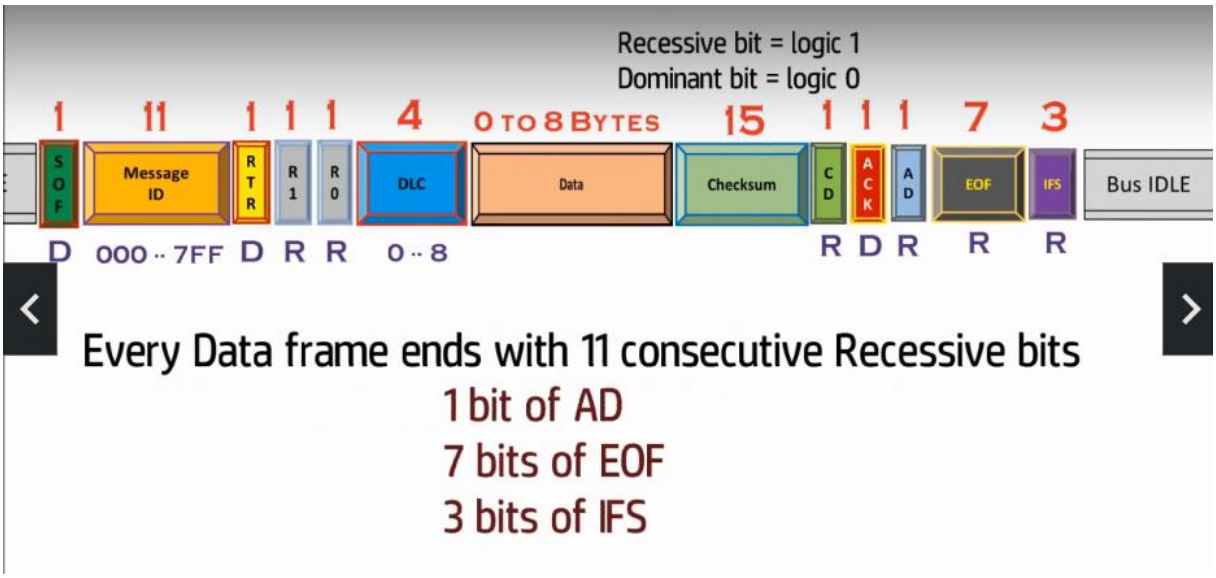
- ACK 1 bit: here where the nodes can tell that the frame is received and it is valid. Here how it is happened the Tx node puts a recessive bit in the ACK, so All th Rx who valide this frame and want to ACK it will put a dominant bit in the slot, so THE Tx if he find that the bit change to a dominant he will that it s ack by the receiver.

  If this bit got a 0 and 1 by two nodes in the same time the bus will take the dominant bit value 0

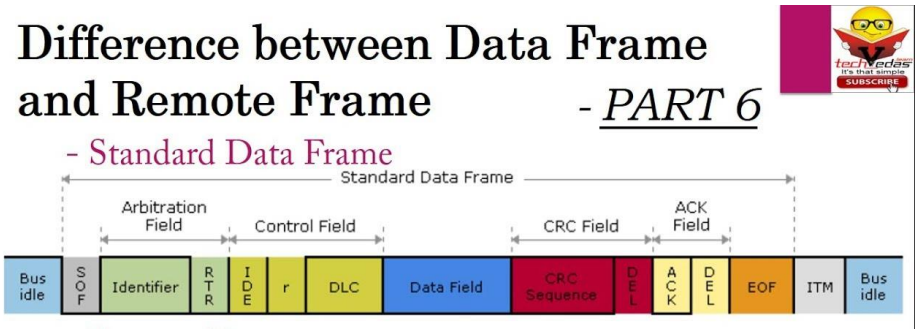


- AD ack delimiter 1bit: always Recessive
- EOF the end of data frame 7bit: marks the frame is ended
- IFS inter frame space 3bits:

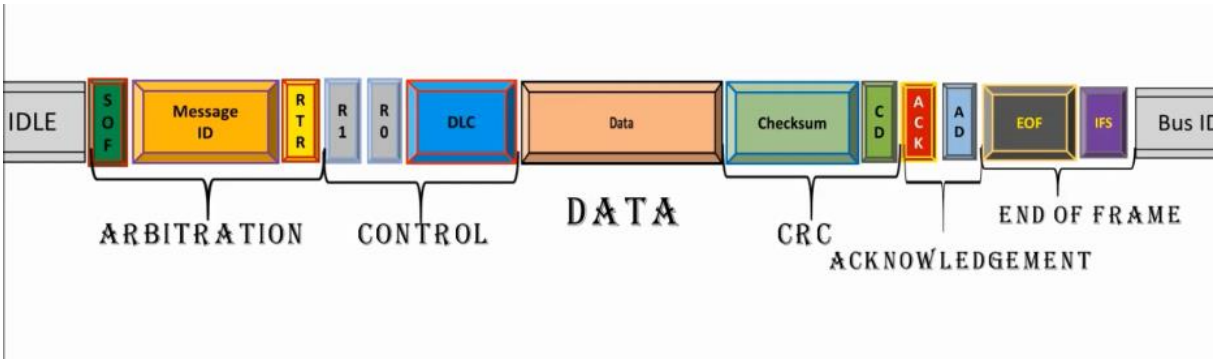
I guess we can argue if it is a part of frame or not, as every frame ends with at least three bits



- Every dat frame ends with 11 conecutive Recessive bits , AD, EOF, IFS just after those bits the bus consider IDLE again and every node can send data frame

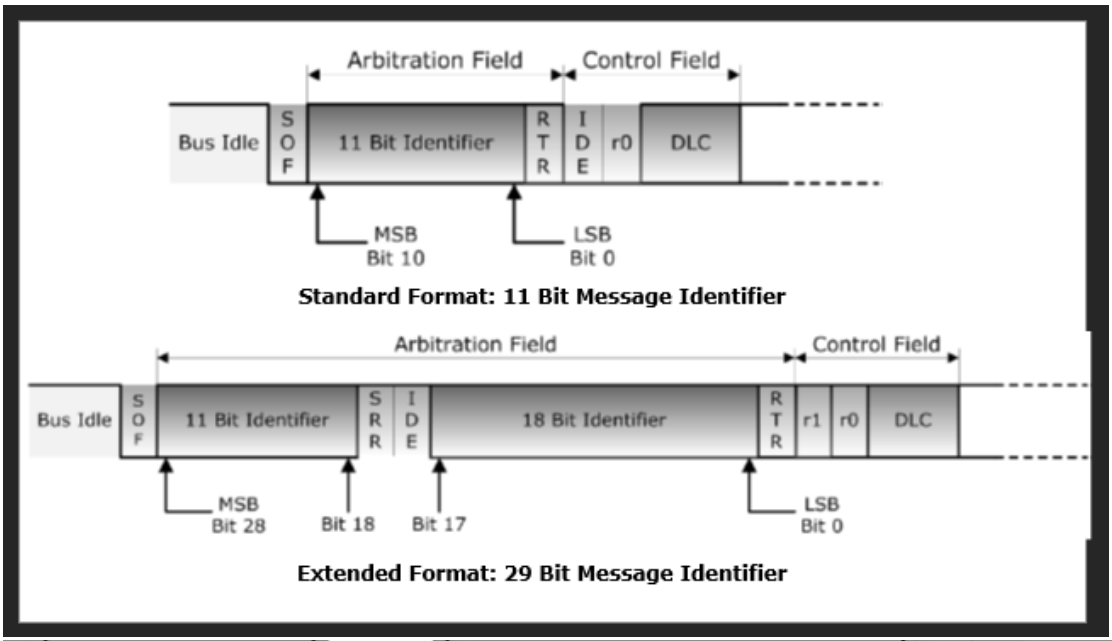


- SOF and msg ID and RTR: called Arbitration field, R0 (IDE) and R1 DLC: control field, checksum (CRC seaquence), CD (DEL) CRC field, ACK and DEL (AD) CK field



11. EXTENDED FRAME FORMAT

- In the extended bit the msg IDE 29 bits, means that 436 milion is Possibal
- Standar and extended can frame could co-exist, means mixt frame



-

So the arbitration field is where the difference exists between them.

You see here there is a 11 bit.

And here you might be thinking that instead of 11 you just put 29 bits straight and then put ID.

But what you have to remember is you have to maintain the backward compatibility, right?

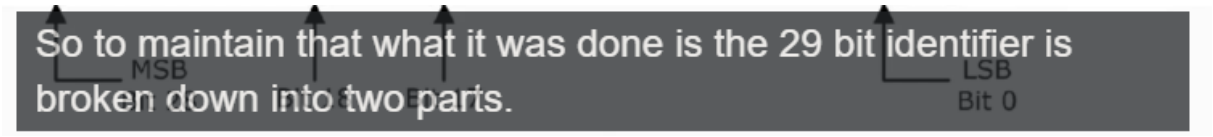
So when the cam controller is there, can network is there in can network and Node is monitoring.

So it usually sees that the first bit after the dominant comes, then that is the first bit dominant

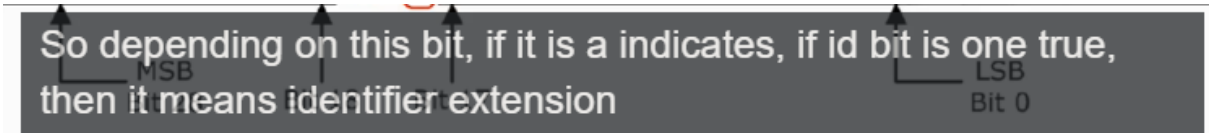
on the bus is safe and then 11 bits will be maintained and it always expects the first bit 11th bit

that is 12th, 13th bit to be RTR.

But if you just put 29 bit identifier here, then it would be comparing assuming the 13th bit of the



Identifier base and the second identifier extension



SRR 1bit: Subtitude remote requist in the place of RTR in standard data format. Always Recessive

IDE: Identifier Extension Indication Bit : indicate the presence of Extended identifier or not

Additional Bits

SRR: Substitute Remote Request → In place of RTR bit in Standard frame format.
Always Recessive and single bit

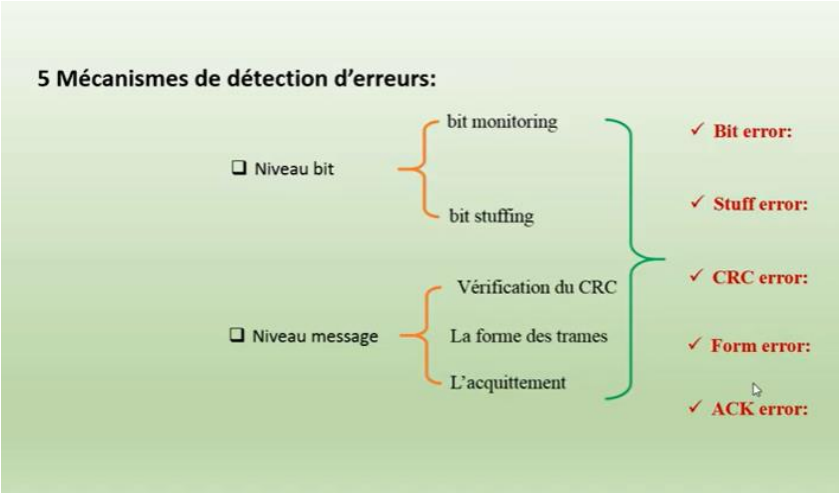
IDE: Identifier Extension Indication Bit → Indicates the presence of Extended Identifier or not and hence differentiates Standard frame and extended frame.
Dominant (0): Standard frame
Recessive (1): Extended frame

12. Priority

The fra;e who had the low ID has the priority

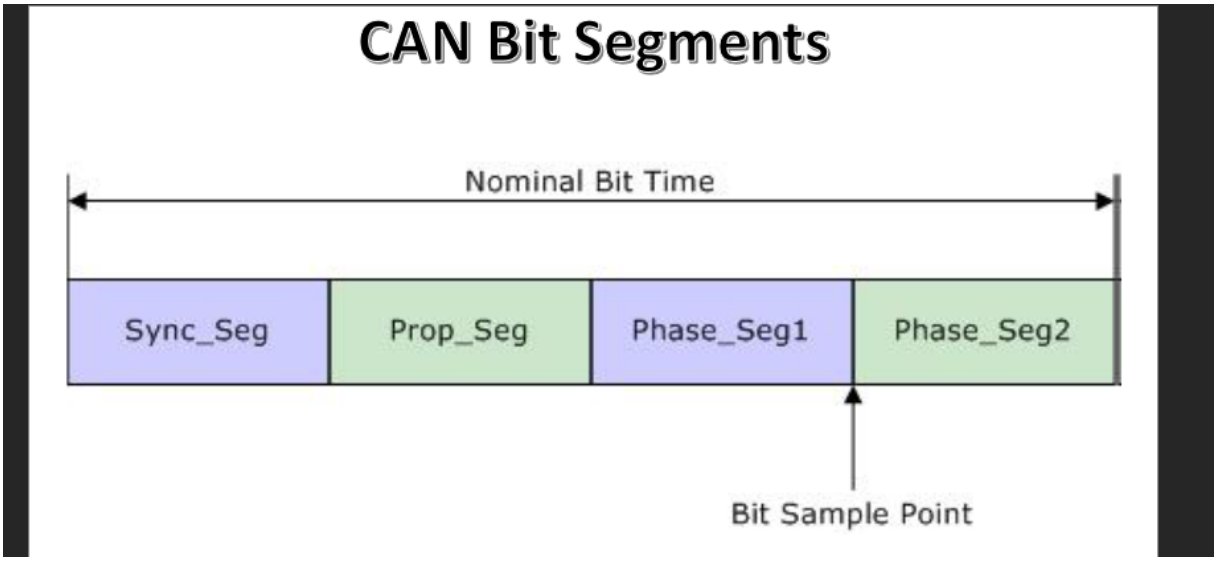
5.2 Mechanism de detection d erreur

1- Au niveau de bit : bit monotoring , bit shuffling
2- :Niveau msg : Verification de crc m la for ; des trame , l ack



13. Bit seg segmentation

In CAN bus every bit divide into 4 parties , this segmentations are not equal



So right after the phase segment one is ended and phase segment two is started.

That point of time, which marks the junction between these two segments is where a sampling is happening.

But the each segments, the sync segment size is not equal to proper segment size, even though in this

pic they look equal

Sync_seg:

As the name indicates, it is a segment used for synchronization of various nodes.

We know that the Can protocol is asynchronous.

That is, there is no separate clock signal to synchronize the node to nodes, right?

So the frame itself will be acting as the source for the synchronization.

So the each bit, the bit which will be responsible for synchronization will be the sync segment.

Basically you remember a sync segment is a segment where the nodes is allowed to put the bit on the

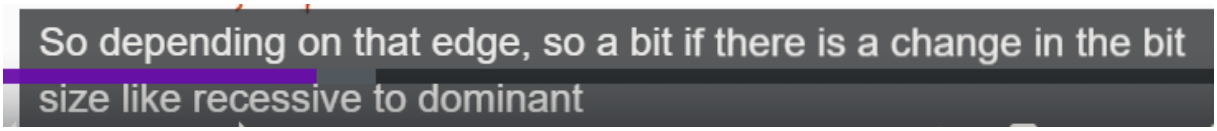
Bus

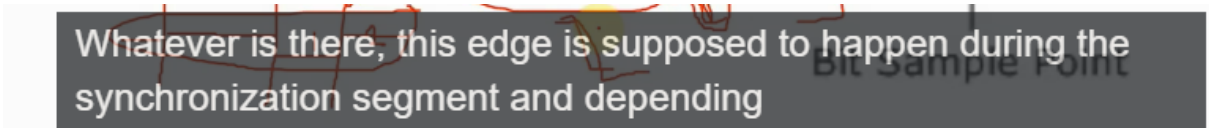
So in a bid time where this node is supposed to put its bit on the bus, is it allowed to put not any

time in the any random time during the whole bit time.

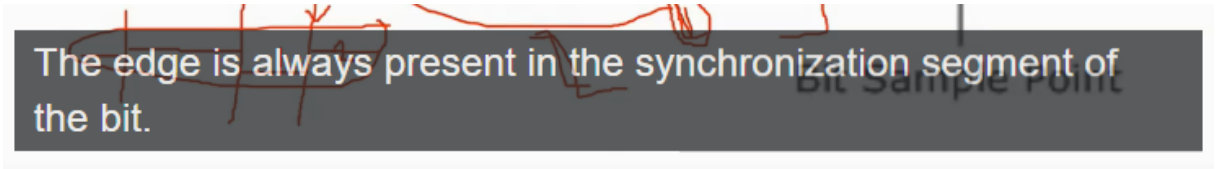
It is supposed to put the bit value on the bus during the synchronization segment period.

Okay, so once this bit value is got received by various nodes where the change is happening.

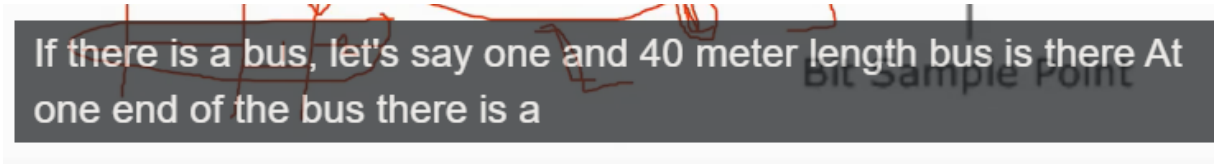


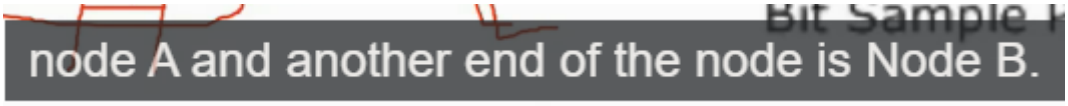


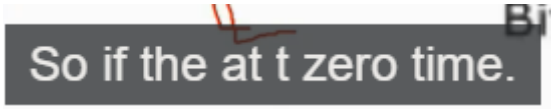
The RX node will be synchoronising to Tx. So

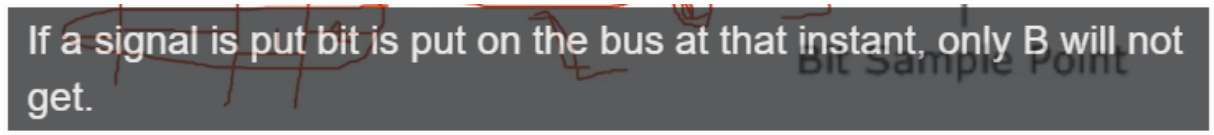


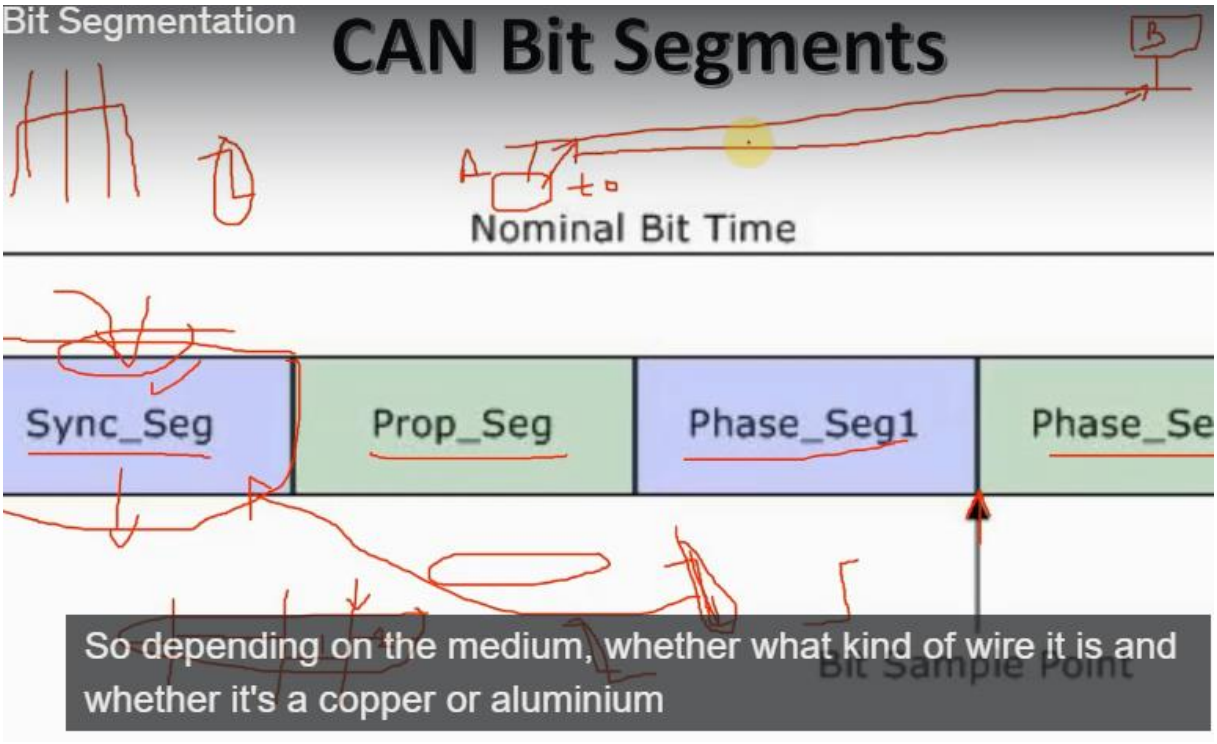
Pro_Seg:





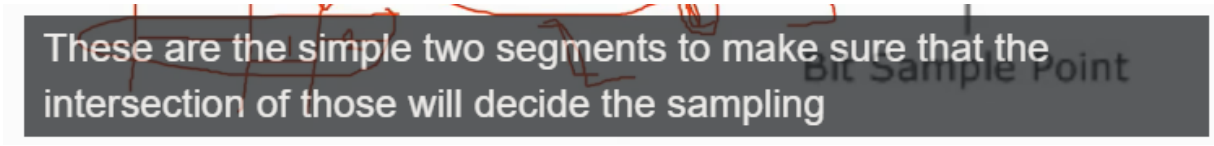




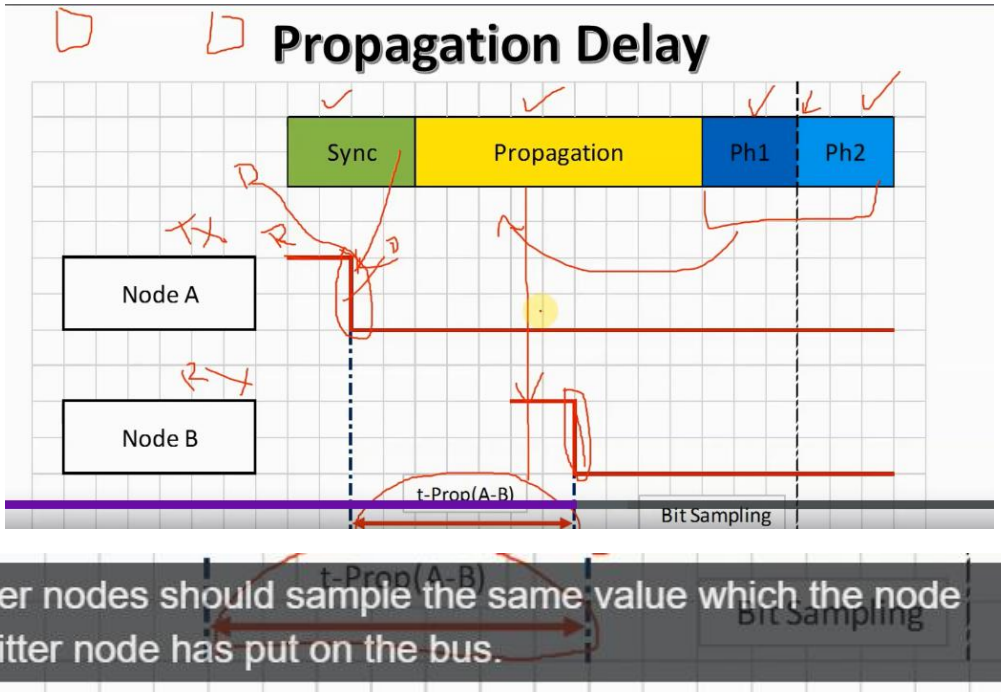


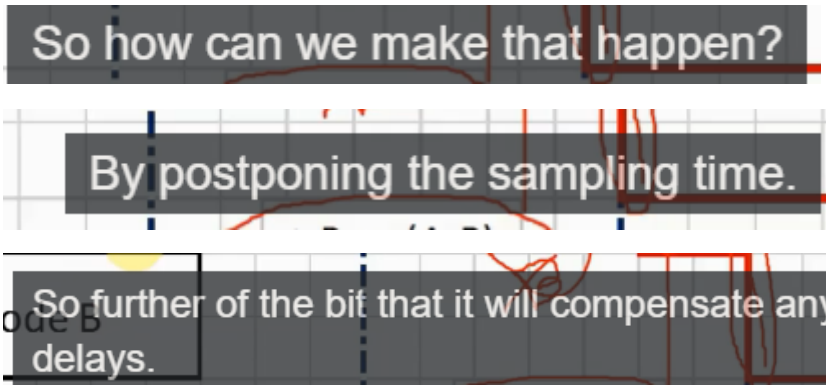
… used to compense the delay of propagation

Phase_seg1 , Phase_seg2:

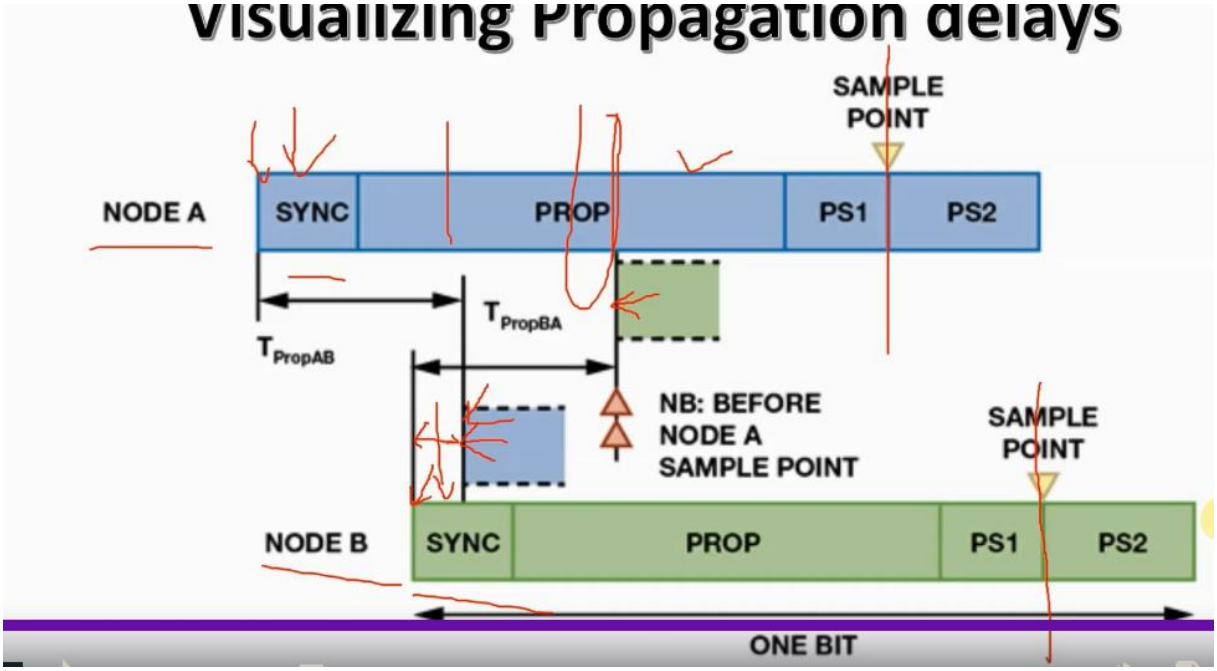


Point of the bit in the bus



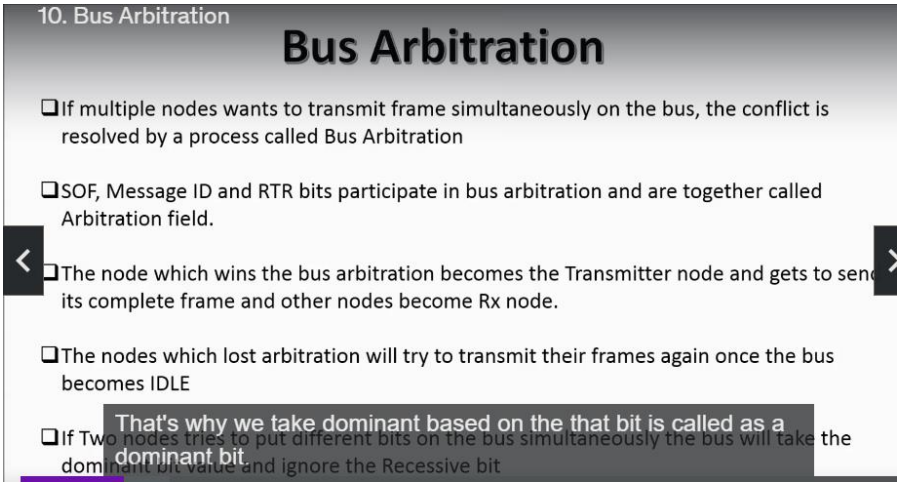


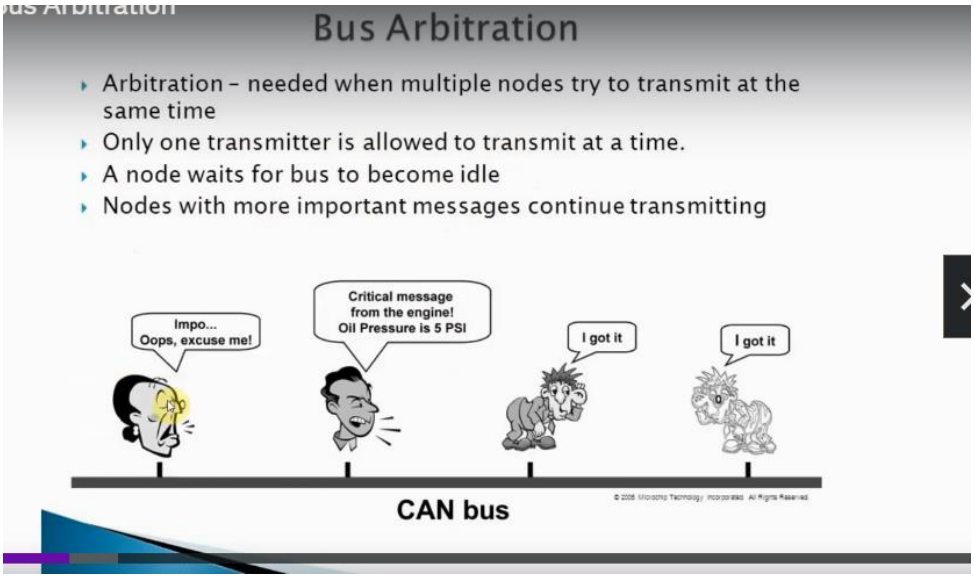
That s whay we have Prop-_Seg

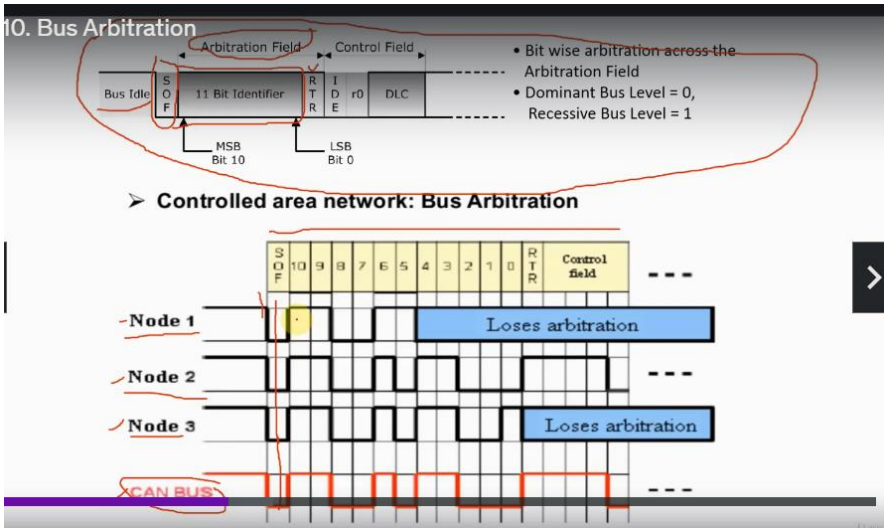


14. BIT Arbitration

Look at resource



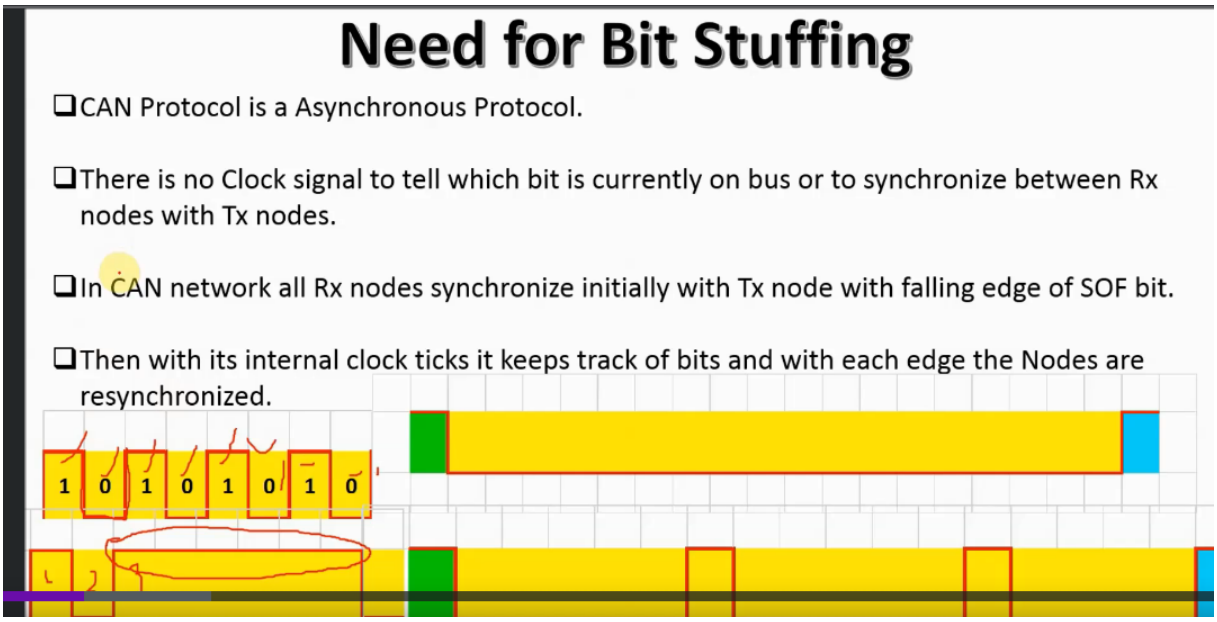




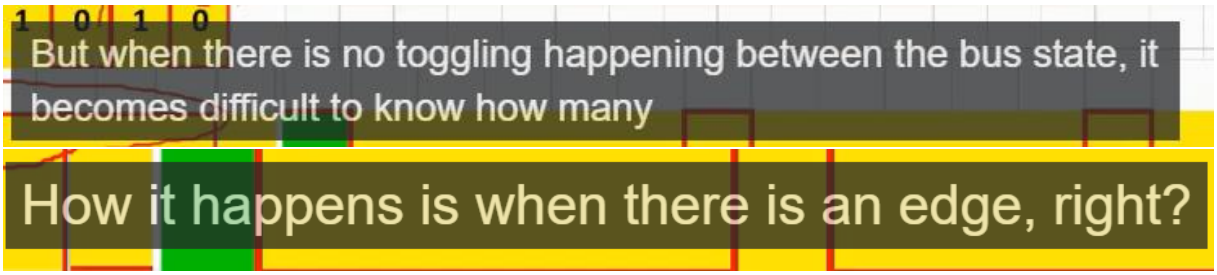
15. BIT stuffling

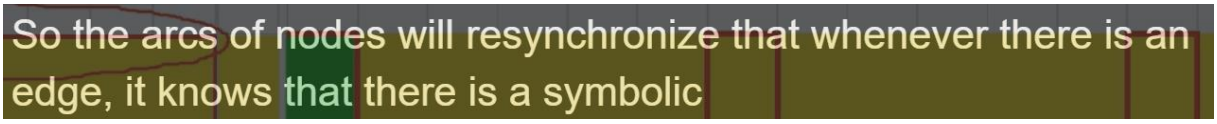
Need for Bit Stuffing

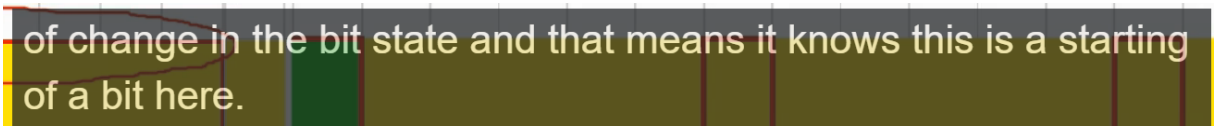
- All Rx nodes have internal clock ticks depending upon which it knows when one bit ends and when next bit starts on the bus. But with each bit the tolerance or delta error keeps on building.
- With every edge (or bus state change) the Rx node knows the next bit has come and re adjusts the internal clock ticks and this is called as Resynchronization of Rx nodes with Tx node.
- So having bus state constant or unchanged for long time is risky in terms of Rx nodes being in synch with Tx node.
- SO CAN protocol designers after research came up with the strategy of bit stuffing.
- If CAN bus has constant value for 5 bit lengths atleast then a complimentary bit is inserted next to it. This is called as bit stuffing

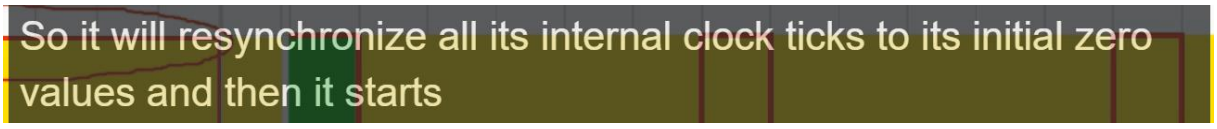


When we have a succesice bits it hard to detect each bit









Like tick, tick, tick, tick, tick, tick.

Some 100 ticks then it will depending on that, it will know some one bit is ended.

But what happens in can is because the nodes are whenever there is no change in the bus state because

it is counting the number of bits, it marks the end of one bit and starting of next bit depending on

its internal clock ticks.

If there is some error for each tick, let us say uh, instead of 100 ticks which it counts because

of some minute crystal problems or some clock tick problems, physical problems, etcetera, because

So if we have an accumulation of Delta error, this could be transorm the data

So but if there is a edge right then this error will be reset to zero because the internal clock ticks

will reset itself to zero again.

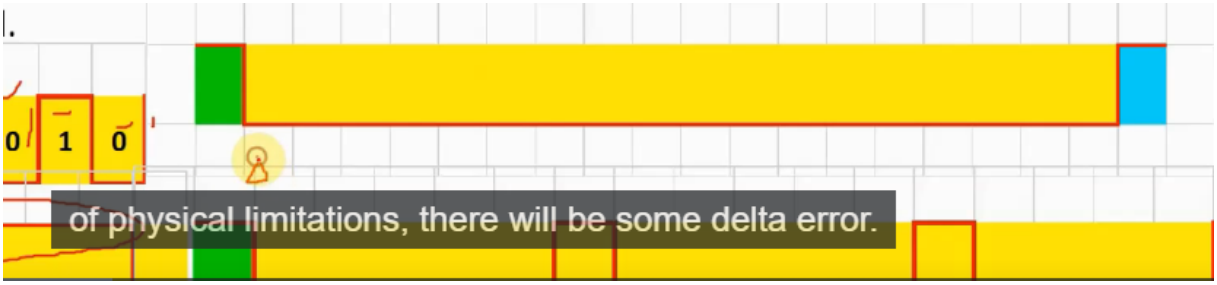
So, so because this is a asynchronous protocol and the frame itself is used for synchronization of

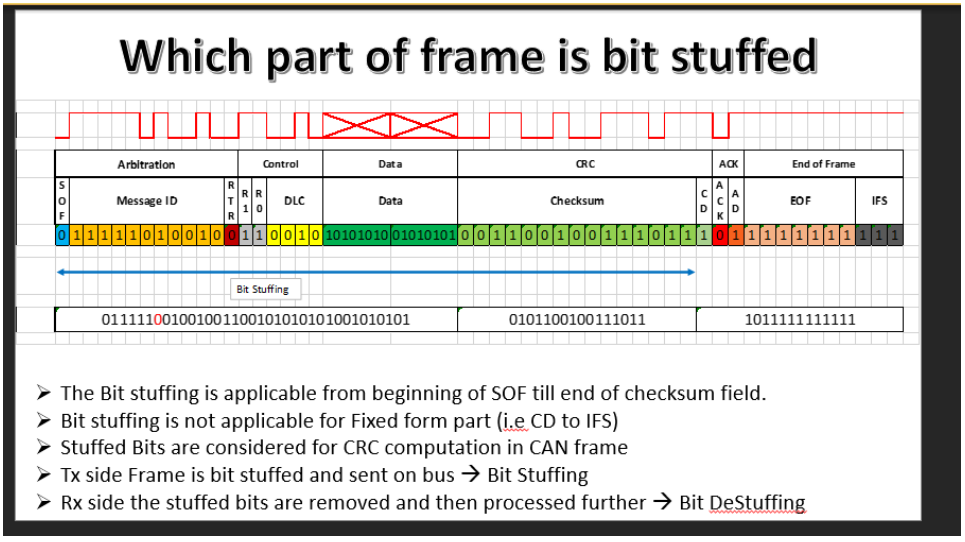
nodes with nodes, how does the node synchronize itself with the nodes?

It's basically because of edge, right?

All nodes have their internal clock ticks depending upon which it knows when one bit ends and when next

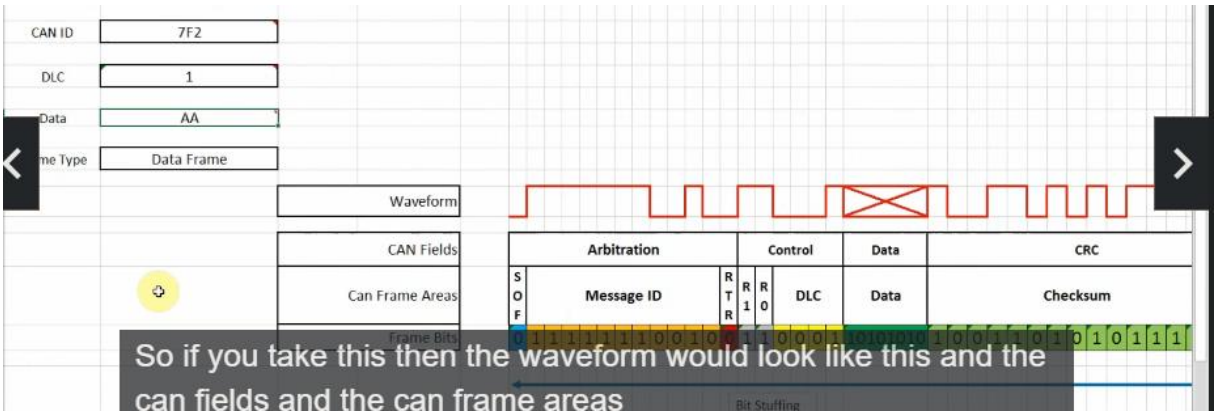
ALL RX nodes *

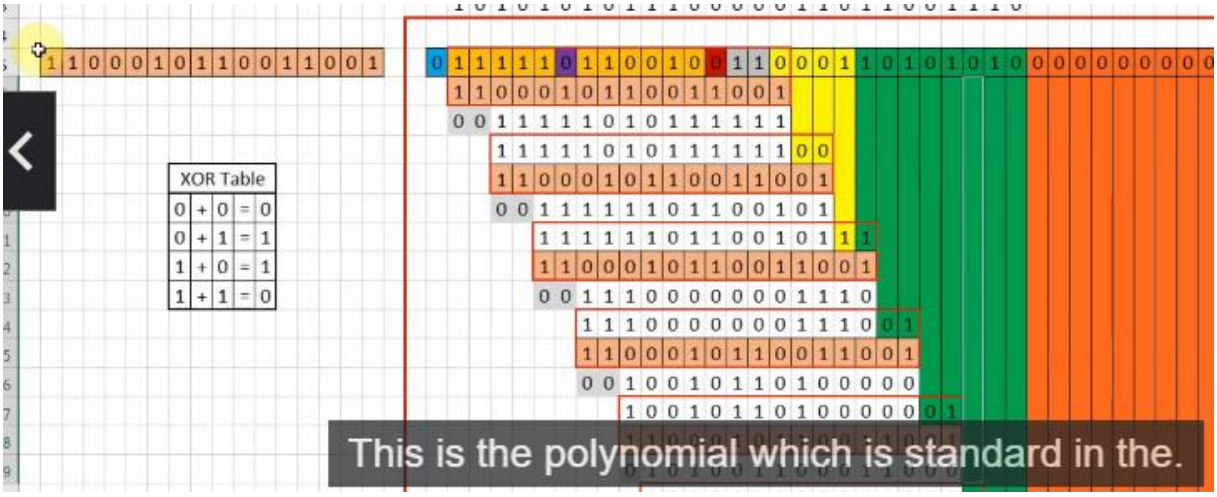




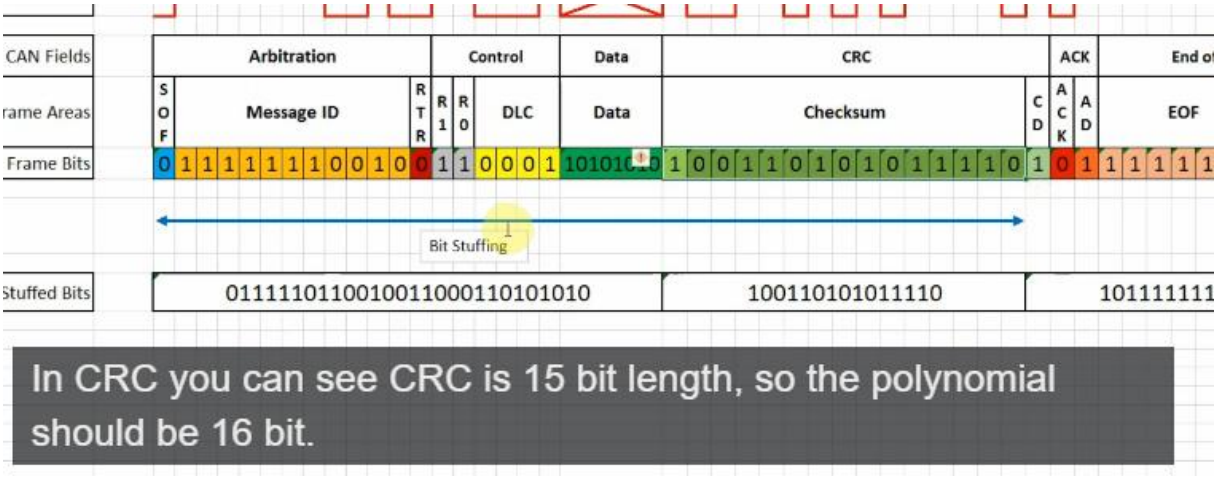
16. CRC in CAN

In can the crc I scomputed by the Can controller itself, by using a method BCH

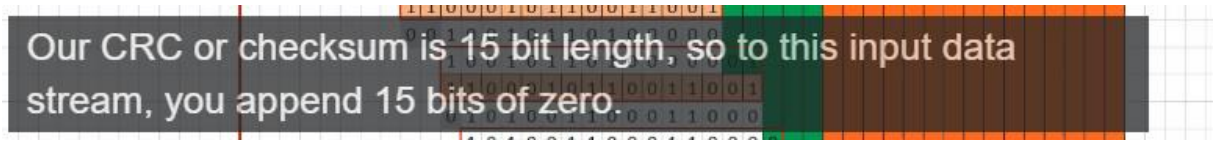

CAN ID
7F2
DLC
1
Data
AA
Data Frame
Waveform
CAN Fields
Arbitration
Control
Data
CRC
Can Frame Areas
SOF
Message ID
RTR
R1
R0
DLC
Data
Checksum
So if you take this then the waveform would look like this and the can fields and the can frame areas

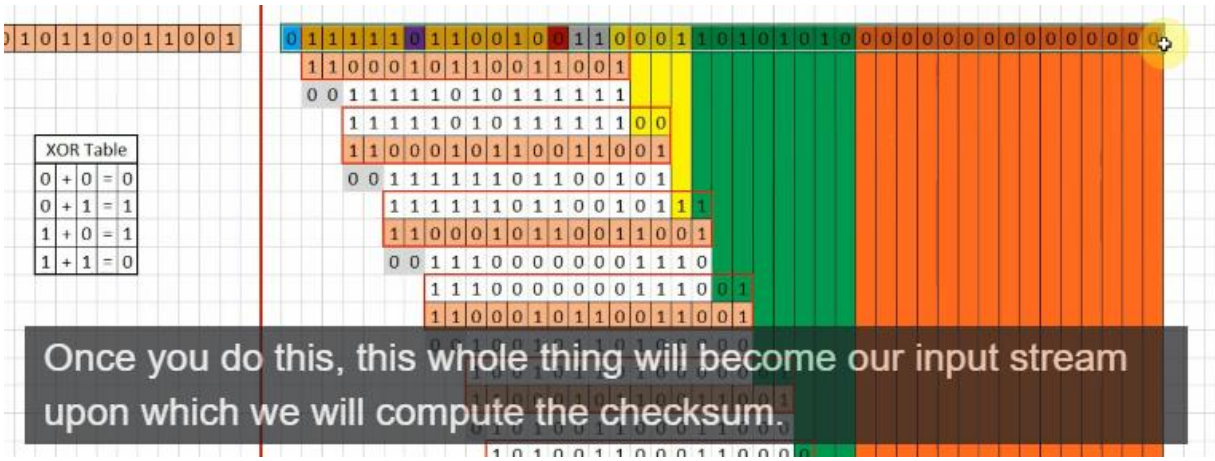

XOR Table
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0
This is the polynomial which is standard in the.


Research method and for can protocol.

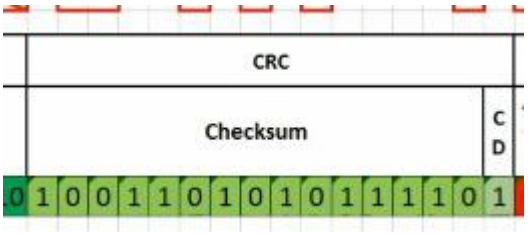

CAN Fields
Arbitration
Control
Data
CRC
ACK
Message ID
DLC
Checksum
EOF
Frame Bits
Bit Stuffing
Stuffed Bits
01111101100100110001101010 10
100110101011110
In CRC you can see CRC is 15 bit length, so the polynomial should be 16 bit.

t


Our CRC or checksum is 15 bit length, so to this input data stream, you append 15 bits of zero.



Sfter the calculation the pulynom of checksum should be = to the result



17. CAN errors

What is CAN Error

- Purpose of CAN: Communicate data from one node to other nodes in a network.
- The data to be communicated is clubbed with frame related fields and communicated serially bit by bit
- All the Rx nodes must receive the same information which the Tx application is intended. If some problem happens in this then we call as CAN error.
- The problem may happen in Tx node, or CAN bus or Rx node. CAN protocol has capability to detect any such problem and raise an error flag.
- So Depending on which node detects an error we have Tx errors and Rx errors

18. Reaction to CAN errors

Reaction to CAN Error

- When a node (Tx or Rx) detects a CAN error it starts sending the error flag on the bus immediately.
- This Error flag is designed in a way that it destroys the data frame currently present on the bus.
- Depending on this error flag other nodes also may put its own error flags on CAN bus.
- All Error flags and delimiters together forms the error frame on the bus.
- Depending on the node which raised the error flag, will increment its error counter value to keep track and manage CAN error states.

---

19. Can errors

CAN Errors

There are 5 types of CAN errors

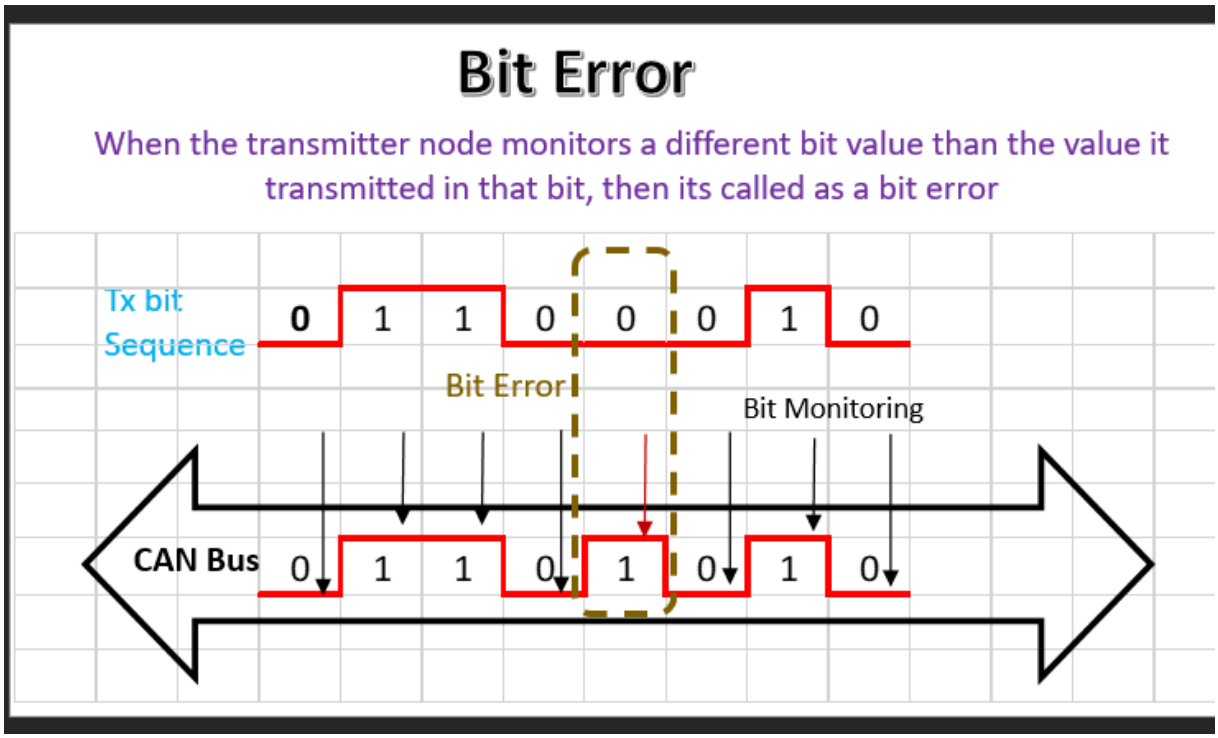
- Bit Error
- Acknowledgement Error
- CRC Error
- Stuff Error
- Form Error

Bit Error and Ack Error are called Transmitter errors
These decide the TEC counter increments

CRC, Stuff and form errors are called Receiver errors
These decide the REC counter increments

Transmitter errors means that only the Tx can detect those errors. Same fot Rx erreors

20. Bit errors



If the transmitter put a value in the bus the bus should take the same value, but for some reason shit happens as u see in the pic below, and we will have a bit errors. So the TX after detection of the b.it errors will send an error flag, just after this detection

6.2 Bit errors exceptions

Bit Error Exceptions

Use cases:

1. Tx puts Dominant (0) bit but finds a Recessive(1) bit on the bus
2. Tx puts Recessive (1) bit but finds a Dominant (0) bit on the bus

Exceptions:

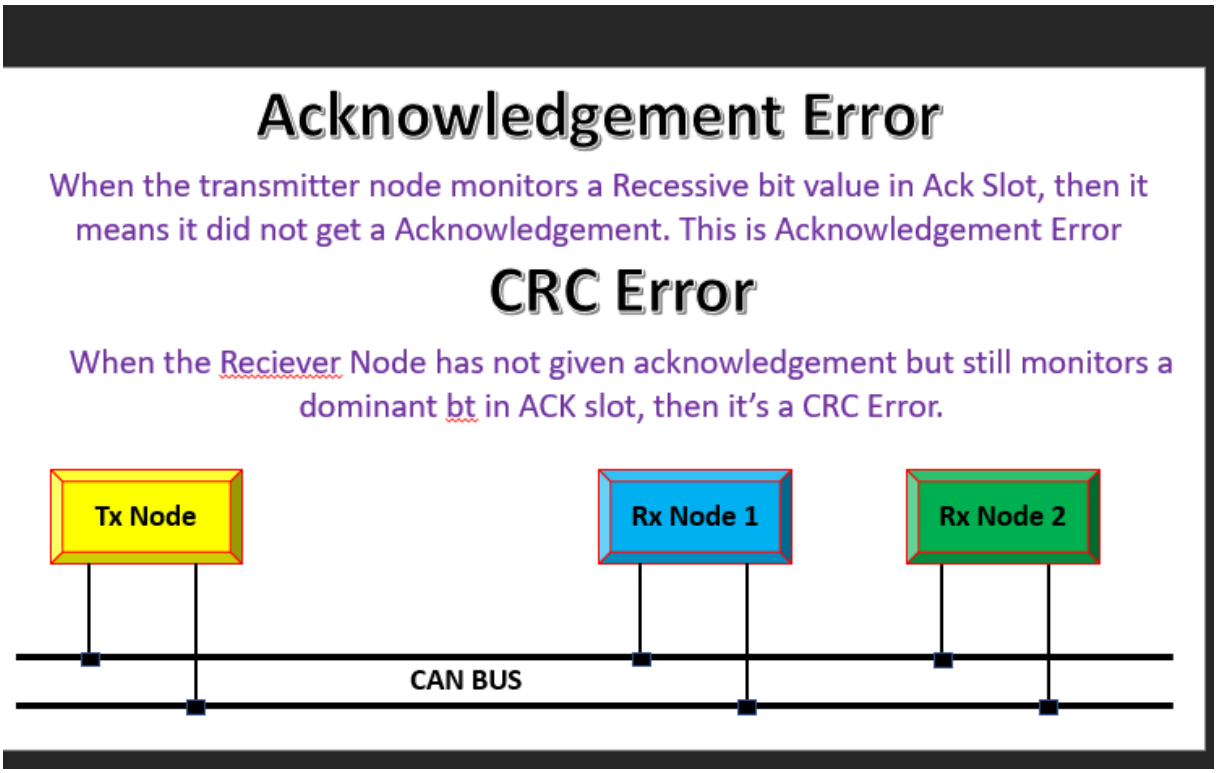
1) During Arbitration, Tx puts a Recessive bit but finds a Dominant Bit then this means ***Lost in Arbitration*** and not ***Bit Error***.

Note: During Arbitration if Tx node puts a Dominant bit but finds a Recessive bit on the bus, then this is a Bit Error only.

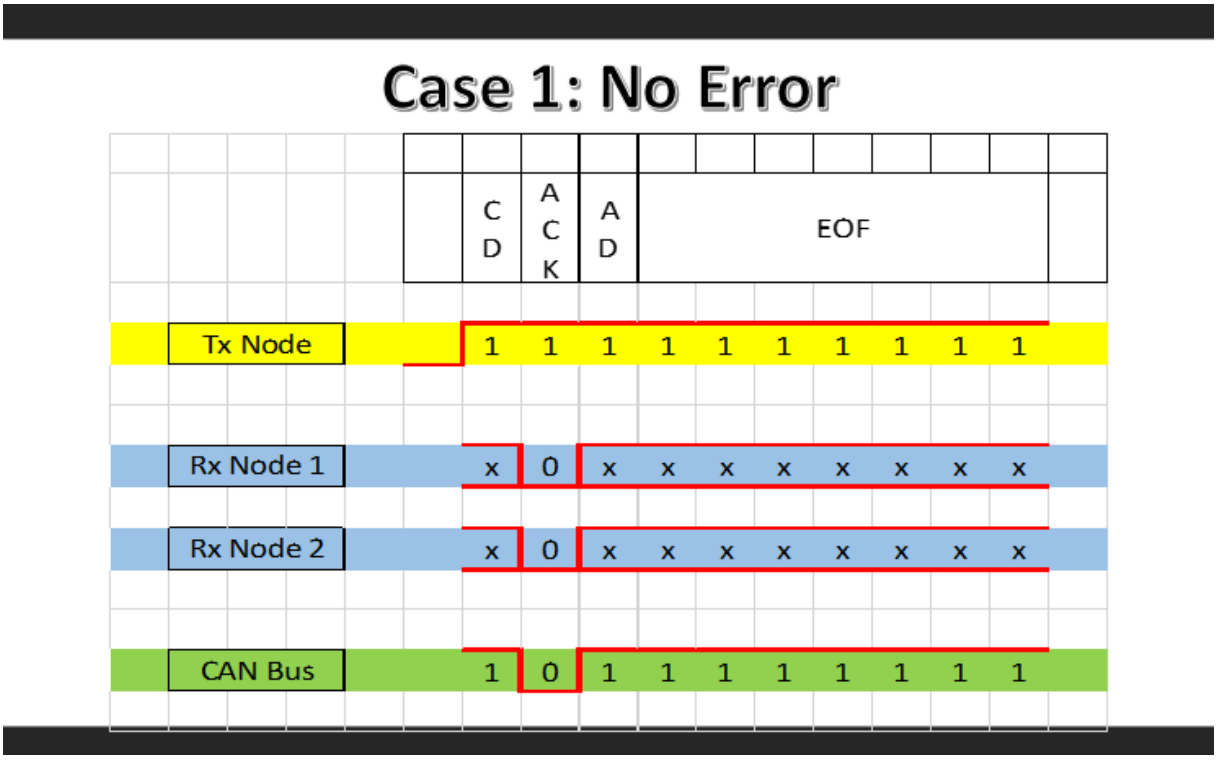
2) During Acknowledgement Slot, Tx node puts a Recessive bit but finds a Dominant Bit then this means ***Acknowledgement Received*** and not ***Bit Error***.

3) After Ack slot, 11 consecutive Recessive bit is transmitted. In that fixed frame if a dominant bit is monitored then it will be a form error and ot Bit error.

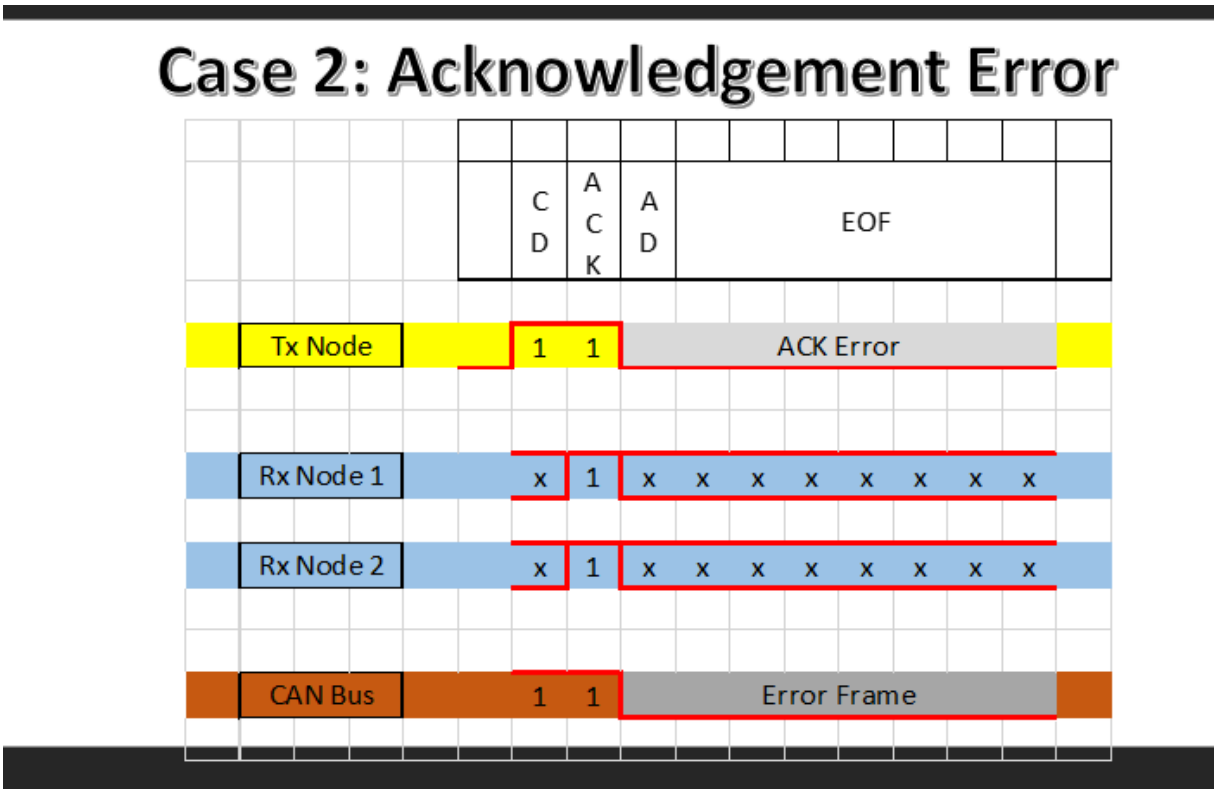
21. Ack error



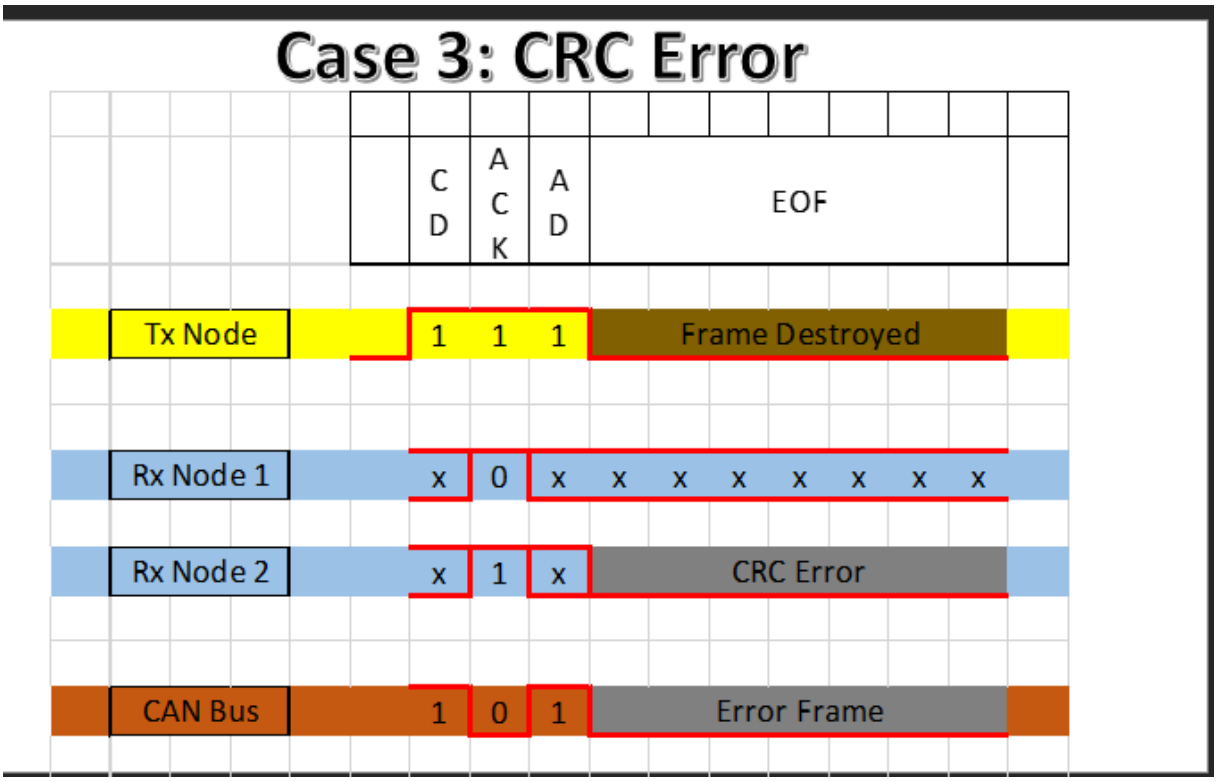
To understand this we will have some cases below:



In this case there is no error



So here the Tx understand that there an error in the frame, because he not receiving any ACK, stull have a rececive bit. That s what we call a ACK error,



So to make a difference between ACK And CRC errors, THE first one start just after the ACK slot and the second one after the DLC slot. And that s why ze have a delimiter BIT DLC

7.2 Stuff errors

Stuff Error

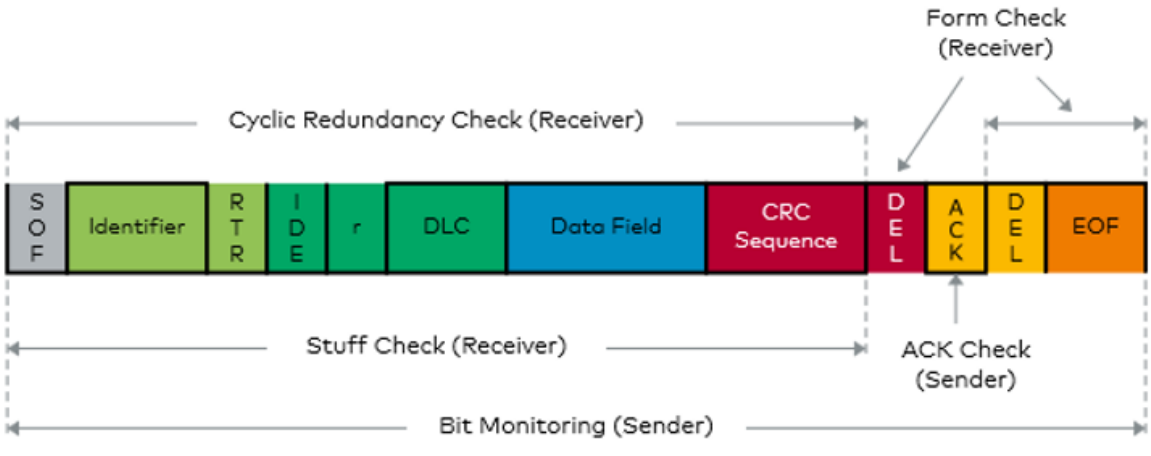
When a receiver receives at-least 6 consecutive bits of same polarity, then it means the bit stuffing rule is violated. This is Stuff Error.

- In CAN protocol, if frame has consecutive 5 bits of same polarity then a complimentary bit is inserted after that called Stuff bit. This helps in better synchronization between Rx and Tx nodes.
- Bit Stuffing is applied from SOF to Checksum field. So the scope of stuff error also is the same.
- The moment the Stuff error is detected, the Rx node raises an error flag the very next bit, hence destroying the current data frame on the CAN Bus.

22. Form errors

Form Error

When the Fixed form of the CAN data frame (CD + AD + EOF) is altered then Form error occurs.



23. Conclusions'

Points to Remember

- When a node Detects a error, it raises the error flag on the bus and destroys that data frame which is erroneous.
- If the error is detected by a Tx node then they are called Tx Errors. Bit and Ack errors
- If the error is detected by a Rx node they are Rx Errors. CRC, Form and Stuff errors
- For Ack, Bit, Form and Stuff error, The error flag is raised as on the very next bit when the error was detected, For CRC error, error flag is raised after 1 bit gap(AD) when error is detected.
- Tx error results in increments of TEC and Rx errors result in REC within a node.
- Error frame format for all CAN error are same. Depending on at which part of data frame the error is raised, we can know which error is it.

24. ERRORS Frame format

Error Frame

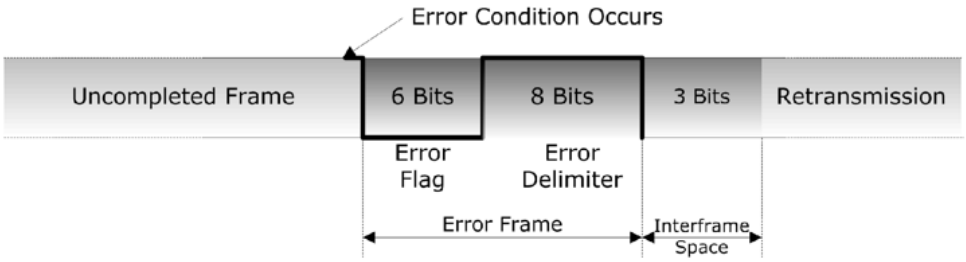
- Error Frame is one of the 4 frame types given in the CAN protocol
- Error Frame Signals to other nodes on the network that current Data frame is erroneous.
- Any of the 5 CAN error upon detection by a node will result in destroying current data frame by overlap of Error frame
- To understand Error frame format we have to understand below three aspects:
  1. Error Flag (Active and Passive)
  2. Basic Error Frame
  3. Complete Error Frame

Error Flag

- When a node detects any CAN error, it knows that the current Data frame is not valid. So the node has to signal other nodes on the network.
- The Node signals the error to remaining nodes by a flag. This is called Error flag.
- Error Flag has to be noticed by all other Nodes and not be mistaken to be normal frame bits.
- So the format of a Error flag is chose in a way that such format cannot exist anywhere in a normal data frame. → Violation of Bit stuffing rule.
- So Error Flag is 6 consecutive bits of same polarity. Depending on Polarity of bits, there are two kinds of Error Flag
  1) *Active Error Flag : 6 consecutive dominant bits (000000)*
  2) *Passive Error Flag : 6 consecutive recessive bits (111111)*

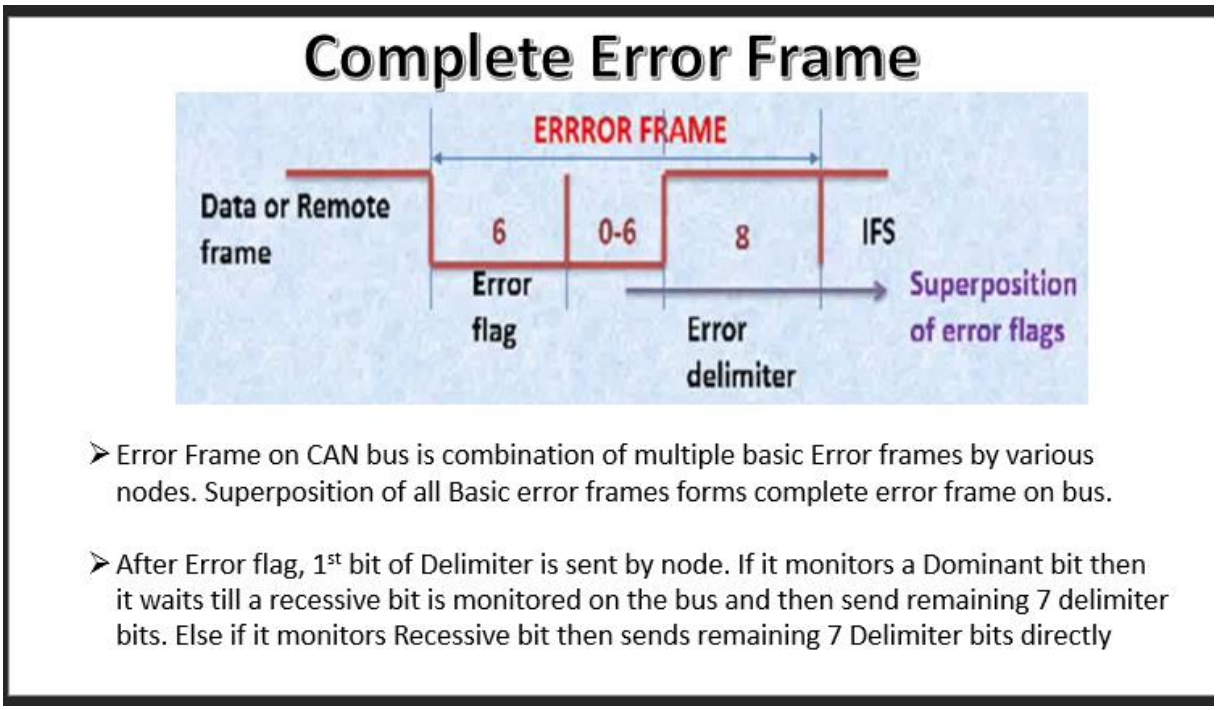
25. Basic error frame

Basic Error Frame



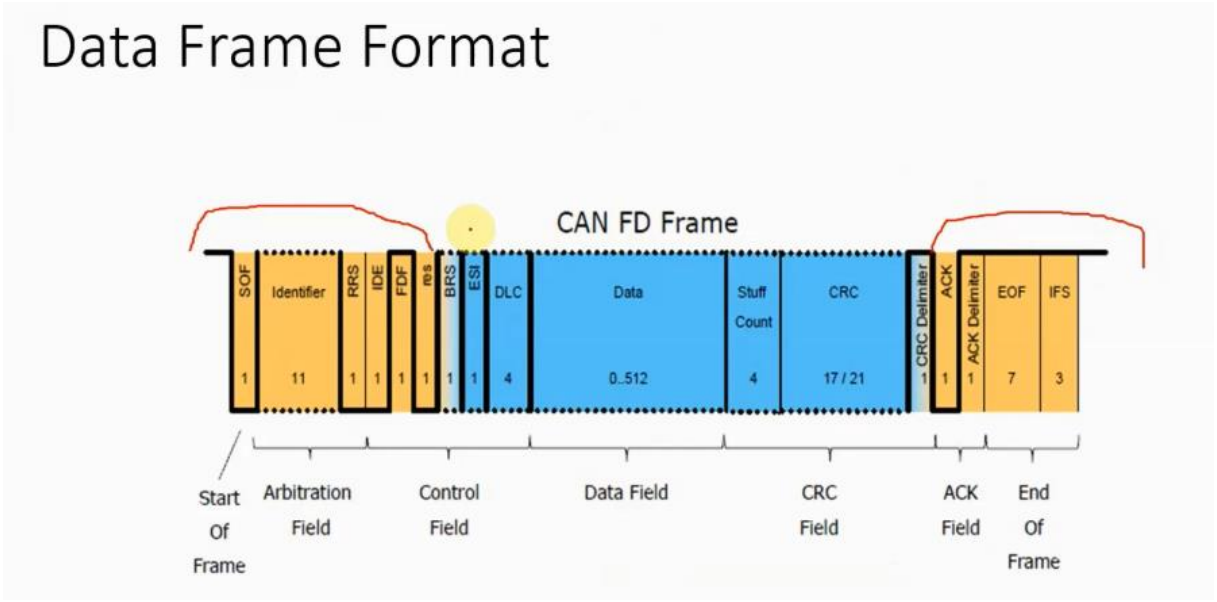
A Basic Error frame consists of 3 parts:

1) 6 bits of Error flag field
   Active error flag : all dominant bits
   Passive error flag : all recessive bits
2) 8 recessive bits of Error delimiter field. (1 + 7)
3) 3 recessive bits of Inter-frame space field.



26. CAN FD

Look the ressource



The orange part will transferred at regular speed, the blue high speed

BRS: ables us to choos if the frame will be in higher frame or not. Cuz the option of high speed is optional. If it 1 then it is a dual frame