Università degli studi di Modena e Reggio Emilia

Dipartimento di Ingegneria Enzo Ferrari

Automotive Connectivity

Anno Accademico 2024/25

Indice

Capitolo 1

Introduction

1.1 Structure and Content

- **Module 1**:
  1. ***intra-vehicles communications***: nodes, sensors, ECU
  2. ***signal busses***: CAN, LIN, FlexRay, MOST, Ethernet [T1/T1S]
  3. ***car domain and OS***
- **Module 2**:
  1. ***inter-vehicles communications***: *V2V* and *V2X* (car is a node)
  2. ***wireless technologies***: Bluetooth, LoRa, C-V2X, IEE 802.11p (bd)
  3. application, messages, broadcast, GPS

Different **domain** or **application** needs different *communications protocols*, is important to understand how each nodes in domain communicate each other (inside the car).

1.2 Intra-Vehicles

From the 80's, where the car's control unit are isolated an there was a dedicated wires connect sensors and actuators with less electronic than now, until the reach the greatest goal of evolution in the automotive sector: autonomous drive. The complexity of the number of connection from each ECU's to the other, also the number of ECU's for each car, is growing. While the number of signal increase in a liner way, the connection between ECU's is growing with a quadratic complexity $O(n^2)$.

If we examine the evolutions of the ECUs number inside an "Audi A6" we can observe that in 1997 it has 5 ECUs and in the 2007 it has 50 ECUs, instead the "Tesla M3" in the 2017 has 70 ECUs. The quadratic increase of ECUs number, however has reach a cap for two main reason: the cost and the space inside the car. Traditionally one ECUs is responsible of one task, but nowadays it could be two type of trends:

1. *distributed of function across ECUs*
2. *integration of multiple function in one ECU*

1.3 Architectures

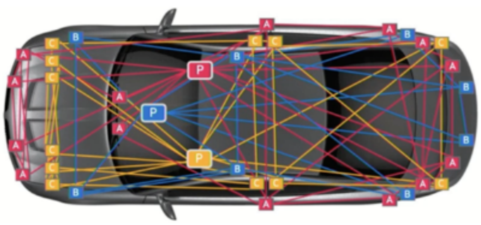


Figura 1.1: *Domain Architecture*

1. central domain controller (**P**) or high performance computer
2. ability to handle more complex functions
3. cost optimization
4. cable harness is rigid and expensive

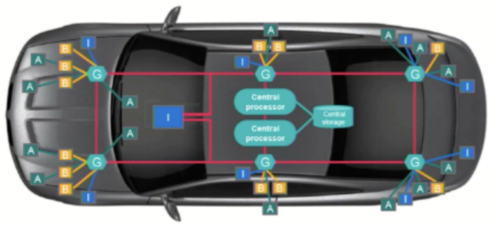


Figura 1.2: *Zonal Architecture*

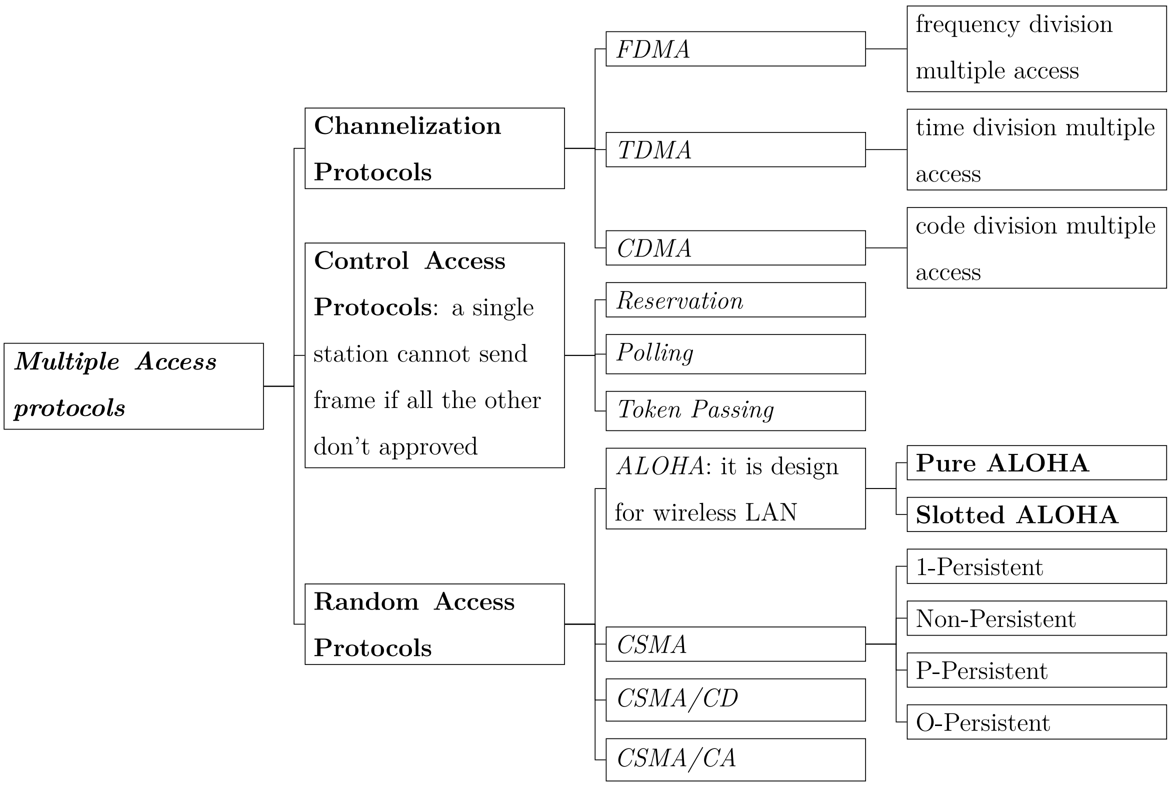
1. local ethernet per zone (**G**)
2. ultra high-speed secured backbone between zone
3. centralized software
4. central computer storage

1.4 Basic Knowledge

1.4.1 Multiple Access Protocols

In the ISO/OSI stack the first layer is the *data link layer* and it is used, in a computer network, to transmit the data between two or more devices or nodes. The data link layer it is normally split in two different sub-layer:

1. **data link control**: is a reliable channel for transmitting data over a dedicated link using various techniques such as framing, error control and flow control of data packets in the computer network.
2. **multiple access protocol**: if the link doesn't connect only two nodes, but multiple nodes can access to the physical link is possible that two or more nodes start to communicate in the same time, and it could be possible to have collision and cross talk between two or more devices. In this case the *multiple access protocol* is required to reduce the collision and avoid cross talk between the channel.



In this course it could be useful to see in dept three type of *Multiple Access Protocols*: the first one is ***Carrier Sense Multiple Access - Collision Detection***, next is the ***Carrier Sense Multiple Access - Collision Avoidance*** and the last one is the ***Time Division Multiple Access***. In the autonomotive domain indeed there is needs to have a bus topology network and it is important to avoid collision.

CSMA/CA - Carrier Sense Multiple Access - Collision Avoidance: the idea is that before transmitting, a node first listens the shared medium to determine if the channel is not used (**idle**), if not it could start to transmit, but the problem start when two nodes begins to write on the nodes together. The **Collision Avoidance** part get in the game when two or more device try to write in the channel simultaneously in this case if another nodes is sense the transmitting node wait for a period of time (usually random) before re-start the writing procedure.

CSMA/CD - Carrier Sense Multiple Access - Collision Detection: is use in early Ethernet technology for LAN. It use carrier-sense to detect if the media is **idle** and it is combined with collision-detection in which a transmission station sense collision by detecting transmissions from other stations while it is transmitting a frame.

1. is the frame ready for the transmission? if not, wait for the frame.
2. is medium idle? if not, wait until it becomes ready.
3. start transmission and monitor for collision during transmission.
4. did a collision occur? if yes, go to collision detecting procedure.
   (a) continue the transmission (with **jam signal**) until minimum packet time is reached to ensure that all receiver detect the collision.
   (b) increment re-transmission counter.
   (c) was the maximum number of transmission (time out) attempts reached? if yes, abort transmission.
   (d) restart from 1.
5. reset the transmission counter and complete frame transmission.

TDMA - Time Division Multiple Access: is a channel access method for share-medium networks. It allow several users to share the same *frequency channel* by dividing the signal into different time slot. The users transmit in rapid succession, one after the other, each using its own time slot. This type of access to the physical medium has higher syncronization overhead tha *CSMA*.

1.4.2 Bit Coding

The first thing is to introduce the ***Electromagnetic Inferference - EMI*** that is a disturbance generate by an external source that affects an electrical circuit by *electromagnetic induction*, *electromagnetic couplig* or from conduction. For reduce EMI there are three possible way: add shield to wires, used twisted pair wiring or use coding with few rising/falling signal edges. At this point we can introduce the two main coding techniques: ***NRZ - Non Return to Zero*** or ***Manchester Coding*** (original variant).

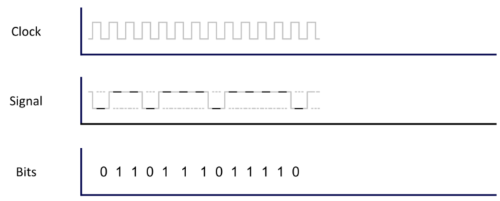


Figura 1.3: *Non Return to Zeros*

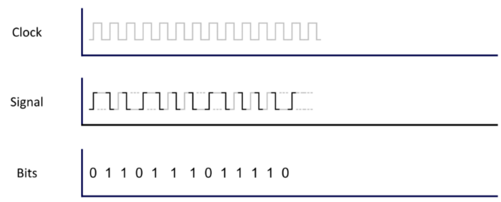


Figura 1.4: *Manchester Coding*

In the ***Non Return to Zero*** the digital ones is, usually, the positive voltage, while digital zeros are represented by other significat condition, like negative voltage.

In the ***Manchester Coding*** (original variant) the digital ones is the rising edge of the signal, instead the digital zeros are represented by the falling edge of the signal.

In both case it must be identify the digital zeros or one on the rising edge of the clock, so the syncronization problem between the clock of the transmitting node and the receiving nodes it is foundamentals.

Capitolo 2

Intra-Vehicles

2.1 ISO/OSI Layers

In telecommunication the idea is to divide each steps into layers starting from the application layer to the fisical ones, every layers have different function and it needs different protocols. Each layer can interact with the one that is above or below it and the communication of two layers follow rigid and specifics rules. Nowadays the standard *de iure* is the **ISO/OSI**, instead the the *de facto* standard is the **TCP/IP** that relax the rigid guidelines. The *ISO/OSI* has seven layers (bottom to top):

1. **physical layer**: specifies the mechanical and electrical properties to transmit bit (in the "real" world) and to control time synchronization.
2. **data link layer**: checked the transmission of the frame, error checking, frame synchronization and flow control.
3. **network layer**: it is used for the transmission of the packets, it is also know as *IP Layer*, in is normally use in ethernet.
4. **transport layer**: reliable end to end transport segment, you can manage how the data have to flow. In 99.99 % of the car domain it doesn't need.
5. **session layer**: establish and tear down sessions.
6. **presentation layer**: define the syntax and the semantics of information.
7. **application layer**: uses data transmitted via physical medium.

In the first module we need only two layers: **physical layer** and **data link layer**. We have to study the behaviour of the communication protocols like CANBus, LIN, FlexRay, MOST and Ethernet in this two layers. Starting from the **transmission medium**, normally the hardware pieces that we use to interact with is:

- **transceiver**: is used to "convert" analog signal to bits (brain less).
- **controller**: control the communication (brain full).

Initially the idea is to focus a little more on **CANBus**, the ***Physical Layer***: is compose by three component: **Physical Signaling - PLS**, **Physical Medium Attachment - PMA** and **Media Dependant Interface - MDI**.

1. **physical signaling**: the main purpose is to understand the bit encoding/decoding (if it is *NRZ* or *Manchester*) and to mantein the synchronization all over the network, every transceiver it must have a the same clock source. The synchronization is the most important things both for the bit encoding/decoding and for don't introduce delay in the communication.
2. **physical medium attachment**: driver/receiver characteristics based on the communication protocol.
3. **media dependant interface**: the connector for access to the physical medium.

Data Link Layer is compose by two component: **Logical Link Control - LLC** and **Medium Access Control - MAC**.

1. **logical link control**: from now on, we start to call *frame* the data that are send/receiver from the physical channel. It is used for *acceptance filtering* that permit to decide if a frame is important for the application above the *controller* and if not discard it. This component include also the *overload notification* and *recovery management* in the case there is an error on the communication they could ask to a re-transmit the data.
2. **medium access control**: is purpose is **error detection** it could check the data encapsulation/decapsulation, frame coding and error detection/signaling/handling.

2.2 Network Topology - The Bus System

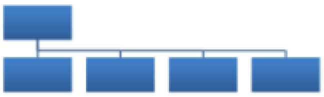


Figura 2.1: *Line Topology*

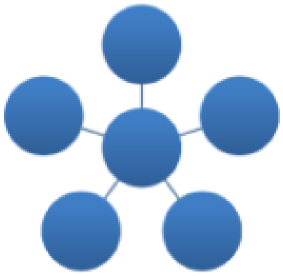


Figura 2.2: *Star Topology*

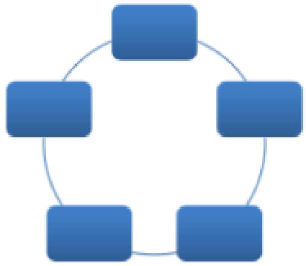


Figura 2.3: *Ring Topology*

In the **Line** topology also know like ***Bus*** topology each node is connected by interface connectors to a single center cable. It is cheaper than the others and it has lower complexity but it is not very robust.

In the **Star** topology every peripheral nodes is connected to a central node called *hub* or *switch*. It has an higher cost and complexity than the *bus* topology, but it is much more robust (if the *hub* goes down it is a *single point of failure*).

The **Ring** topology is a *daisy chain* in a closed loop. When a node sends data to another, the data passes through each intermediate node on the ring until reach its destination (it use only one direction). It is not too munch expensive, but has higher complexity (if you want add a new node it could be troublesome).

In the autonomotive domain it is chosen the ***Bus Topology***, why? The first thing is that in the automotive industry it is mandatory to maintain lower the cost. The *busses* are very cheap for the materials, the weight and the volume. In the *bus* topology it is possible to have higher modularity, you can *plug & play* a node "when you want", in that way it is possible to have fully customizability inside the vehicles. The last things is that there is shorter development cycles. In the autonomotive field there is three main component:

1. ***transceiver***: it is the *physical layer definition* and implement the first layer of the *ISO/OSI* stack.

2. ***communication controller***: it is the communication protocol and implement the first and the second layers of the *ISO/OSI* stack.
3. ***ECU***: also know like **electronic controller unit** and implement the last layer of the *ISO/OSI* stack, the **application** layer.

The idea is to made possible to abstract the application layer in order to, if you want, change the first two layers, for example from CANBus to FlexRay, but nothing change at the application layer.

2.3 Controller Area Network

The **Controller Area Network** also know as **CAN** is a vehicle bus standard to enable efficient communication. It is originally developed to reduce complexity and cost of electrical wiring. ***CANBus*** use an **electrical** medium over wires and a **broadcast** data transmission. CANBus use the ***CSMA/CR*** like *multiple access protocol*, it means *carrier sense multiple access collision resolution* protocol, that permit to CANBus to have ***arbitration*** on the channel access. In this way there is random access to the physical channel, but it is impossible that there is some collision on the communications.

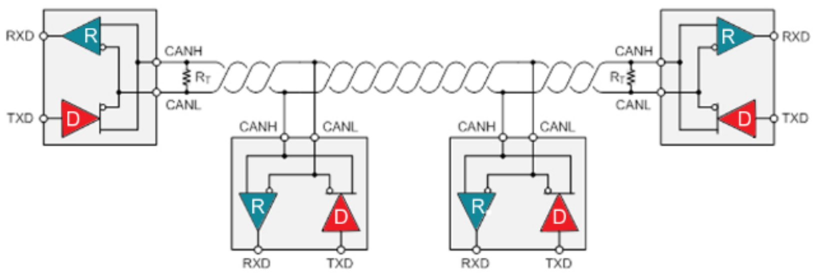


Figura 2.4: CANBus Network Topology

The **CANBus** network is compose by two wires: **CAN High** and **CAN Low**. The data is transmit over the wire using the *potential difference* on each transceiver. Two twisted wires are use because it gives to the protocol **noise resistance** and **increase resiliency**, if one brakes, CAN Low *survives*. At the end of the wire in the bus topology there are place two impedance R_T of 120Ω. Each CANBus node has three element:

- **CAN Transceiver**: is directly connected to the medium access by two pin (one on CANH and the other on CANL). It has the goal to translate the voltage level into bits (during the reception) and send it to the *CAN Controller* and translate bit into voltage level (during the transmission).
- **CAN Controller**: is connect to the *CAN Transceiver* by two pin (CANTX and CANRX) and is scope is to: message completion, control bus access, transmission and reception of the message, bit timing.
- **Microcontroller**: application software communicating with other ECUs via messages over the bus.

CAN Message							
1 bit	29 bit	1 bit	6 bit	0-64 bit	16 bit	2 bit	7 bit
SOF	CAN-ID	RTR	Control	Data	CRC	ACK	EOF

- **SOF**: is the **start of frame** is always set to *dominant **0*** to tell the other ECUs that a message is coming.
- **CAN-ID**: contains the message identifier - lower value have higher priority.
- **RTR**: is the **remote transmission request** allow to ECUs to "request" message from other ECUs.
- **Control**: informs the lenght of the *Data* in bytes (0 to 8 bytes), two bits are *reserved* for future implementation.
- **Data**: contains the actual data values, which need to be "scaled" or converted to be readable an ready for analysis.
- **CRC**: is the **cyclic rendundancy check** is used to ensure data integrity.
- **ACK**: is the **acknoledgement** this slot indicates if the CRC is OK all the bits must be **recessive** (*logical 1*).
- **EOF**: is the **end of frame** marks the end of CAN message all the bits must be **recessive** (*logical 1*).

The CANBus use a *message passing* technologies, it means, when a message is sent through the wire by an ECUs all the CAN Transceiver reciver the message, but if a application layer of one of another ECUs doesn't need that message it could ignore or if it need it, it could accept that message, using the *CAN-ID* as identifier. In other word the CANBus use the **receiver-selective** form of addressing. In the CANBus the bit logic is pretty simple, each ECUs reads the wire (through a buffer) and each ECUs **can** write on the line (through a transistor), in this way the **basic state** is **up** ($+5V$ or logical ones) when one or more ECUs want to set signal low turn on transistor conductive (diode), this connect the bus to signal ground in this case the bus level is **low** ($0V$, or logical zeros) indipendently from other ECUs. The **0** is named **dominant level**. It could be see the CANBus wires as **logical AND** (if an ECUs write zeros the state is *zeros*).

The CANBus is an **event-driven** bus system, it means that there is no need to wait a scheduled time slot for sending data and there is the possibility of collision over the communication channel. If an ECU X registers an event e it is authorized to access the busses immediately and send data, but if another ECU Y is alredy transmitting data, then X waits. We want to calculate how long it takes a message to be sent, the first thing to do is to calculate the maximum bits number that is allow in a CAN Message: 130 bits. The CANBus can have lots of different bus speed $B \in \{5k \cdot \frac{bit}{s}, 125k \cdot \frac{bit}{s}, 250k \cdot \frac{bit}{s}, 500k \cdot \frac{bit}{s}, 800k \cdot \frac{bit}{s}, 1M \cdot \frac{bit}{s}\}$, let's consider the average $B = 500k \cdot \frac{bit}{s}$, the resulting time for sending a message is equal to $T_x(time) = \frac{M}{B} = \frac{130bit}{500k \cdot \frac{bit}{s}} = 0.25ms$, but what is happen if two ECUs start the communication on the same time? Let's consider the case where there are three ECUs X, Y, Z, X and Y are waiting Z because it is using the medium access, but probably they start to transmit in the same time when the busses is free, in this case we have a **collision**, the solution is how CANBus implement the ***CSMA-CR***, **carrier sense multiple access - collision resolution**, the two ingredients are how we can see the CAN busses (like a logical AND) and the **CAN-ID** to the logic prioritizing.

1. ECU X want to send: it must check if the bus is free (*carrier sense* - **CR**).
2. if it is busy the ECU have to wait.
3. when the bus is free, it could happen that one or more ECUs are ready to transmit, and start the communication together (*multiple access* - **MA**).
4. the last incredient is how to avoid the impending damage born from the collsion? (*collision resolution* - **CR**) $\rightarrow$ ***bitwise arbitration***.

All the ***bitwise arbitration*** is base on the first two field of the CANBus Message:

SOF (it is for everyone a **dominant bit**: ***0***) and **CAN-ID** (it could be 11 bits, in the standard CANBus and 29 bits for the extended ones). We know that in CANBus the ones with the lower *ID* has the greatest priority. Another basic know is that the CANBus network work like a *wired-AND* so if a nodes wrote on the bus a ***0*** the entire network has logically low value, also if someone else try to wrote a logically high value.

	ID 10	**ID 9**	**ID 8**	**ID 7**	**ID 6**	**ID 5**	**ID 4**	**ID 3**	**ID 2**	**ID 1**	**ID 0**
A	1	1	0	**0**	1	0	0	1	1	0	0
bus	1	1	0	***0***	1	0	0	1	1	0	0
B	1	1	0	**1**							

node B loses *arbitration*

→ stop sending and re-start sensing

wired-and bus logic		
sender a	**sender** b	**bus level**
1	1	1
1	0	0
0	1	0
0	0	0

We have three knowledge: the default value of the CANBus network is logically high, the bus work as *wired-AND* and the logic ***0*** si the **dominant** value, so if the *sender a* or *sender b* send over the bus the ***0*** value, it win the *arbitration* with the other *sender*.

arbitration logic		
sender	**bus**	**interpretation**
0	0	**next**
0	1	***fault***
1	0	**stop**
1	1	**next**

We alredy know that CANBus is *carrier sense* if the sender sent over the network a logical ***1*** but read logical ***0*** knows that it losts the *arbitration* with another *sender* and have to stops the transmission.

Priorities instead of Collision: the bus logic and arbitration logic not only prevent collision, it ensure a priority-controlled bus access: smaller ECUs ID, higher priority.

CANBus Message Integrity: the idea is to use the Data field to generate a CRC to permit the check on the integrity of the message, but wee need some basic knowledge before start: *polynomial division* and *XOR*.

> ***Polynomial Reminder Theorem***: given two polynomials $M(x)$ (the dividend) and $G(x)$ (the divisor), asserts the existence (and the uniqueness) of a quotient $Q(x)$ and a remainder $R(x)$ such that:
>
> $$M(x) = Q(x) \cdot G(x) + R(x)$$
>
> N.B. the degree of $R(x)$ is stricly lower than the degree of $G(x)$.

In the calculation of *CRC* depends on the arithmetic of modulo 2 polynomial. A modulo 2 polynomial is like:

$$a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + ... + a_2 \cdot x^2 + a_1 \cdot x + a_0$$
$$a = \{0, 1\} \quad \forall a \in \{a_0, a_1, ..., a_n\}$$

An example of the representation of a binary polynomial is like: $x^3 + x + 1 = 1011$. If exist an x with a certain exponent e like: x^e in the binary representation the position e is fill with a 1.

$\oplus$	**0**	**1**
0	0	1
1	1	0

The ***XOR*** is a digital logic gate that gives a true (logical 1) when the input number is odd, otherwise is false (logical 0).

CRC Encoding:

1. we need to transmit a *n bits* **message** $M(x)$: $deg(M(x)) = n - 1$.
2. we have a *m + 1 bits* **generator** $G(x)$: $deg(G(x)) = m$.
    - the **remainder** $R(x)$ of the division $\frac{M(x)}{G(x)}$ will have strictly lower degree respect to $G(x)$ and, in the worst case, the maximum value will be $deg(R(x)) = m - 1$.

- $R(x)$ can always expressed with m bits.

3. add m zeros at the end of $M(x)$: this means to do the following $M(x) \cdot x^m$.
4. divide the **new message** $M(x) \cdot x^m$ with the **generator** $G(x)$ to obtain the ***reminder*** of m bits called ***CRC***.
5. the final message $B(x)$ is equal to $M(x) \cdot x^m + CRC$: this means to add the CRC bits at the end of the message replacing the m zeros padded before.

Example:

$$M(x) = 1101011011 \qquad G(x) = 10011 \quad (m = 4)$$
$$M(x) \cdot x^m = 11010110110000$$

```
             1100001010
      10011)11010110110000
            10011
             10011
             10011
              000010110
                  10011
                   010100
                    10011
                      1110
```

The final message $B(x)$ is equal to: $B(x) = \underbrace{1101011011}_{\text{M(x)}}\underbrace{10011}_{\text{R(x)}}$

CRC Decoding

1. the receiver **acquire** $B(x) = M(x) \cdot x^m + CRC$.
2. the receiver **knows** $G(x)$.
3. the receiver **divides** the whole message by the generator: $B(x) = \frac{M_x \cdot x^m + CRC}{G(x)}$
4. if the receiver obtain **no reminder** the transmission was successfully (no errrors detected).

CRC Error Resistance: consider an error $E(x)$ occurs on the transmission channel and the receiver $B(x) + E(x)$ instead of simply $B(x)$, when the *CRC logic* can failed? The problem occure when $E(x)$ is multiple of $G(x)$ in this way $\frac{B(x)+E(x)}{G(x)}$ gives no reminder, so the receiver mark $B(x) + E(x)$ as a correct message. To avoid this problem we need to choose in appropriate way the generator $G(x)$, this is the reason why the $G(x)$ it is standard in the *CRC Encoding* (by the protocol).

CRC Design Priciples: $G(x)$ is extremely important in a way that $E(x)$ cannot easily be multiple of $G(x)$. For **detecting single bit of error**:

- $E(x) = x^i$ for error in i-th bit.
- if $G(x)$ has more than 1 term it cannot divide x^i.

Mathematical theory help us to desing powerful $G(x)$ with fancy characteristics, in CANBus the generator is: $G(x) = x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$. **sender** and **receiver** must to agree on the **generator**.

CANBus bit coding: we know that there are two main bit coding algorithm: **Non return to Zero** (is less noisy) and **Manchester coding** (carries the clock with him on every single bit). In CANBus is important the clock for the synchronization between nodes, so it could be thinks that *Manchester coding* is the best one to be used. The *Manchester coding* has a big problem: the ***clock drift problem***. The *clock drift problem* is **caused** by natural variations of ***quartz*** (environment), for the correct working of CANBus the receiver must sample signal at the right time instant. *Clock drift* leads to ***de-synchronization*** of the clock that comport a bad interpretation of bit sequence. In order to avoid this type of problem, it is necessary to reduce the rising/falling edge of the signal, so it is advise the usage of **NRZ**.

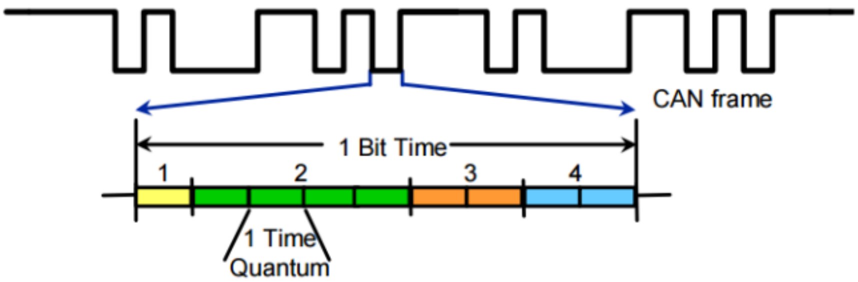
Problem

When using *NRZ* coding, sending many identical bits leaves no signal edges that could be used to compensate for the clock drift.

Solution

Insertion of extra bits after n consecutive identical bits $\rightarrow$ ***Bit Stuffing***. In CANBus $n = 5$.

Time Quanta (TQ): is the smallest time slice it could be count.



It is normally divided into four kind of field: *synchronization segment*, *propagation segment*, *phase buffer segment 1* and *phase buffer segment 2*. A *bit* it is compose from 8 to 25 **time quanta** and it is the smallest discrete timing resolution used by CANBus node. Each **TQ** is generated by programmable divide of the oscillator. Each segment is composed by an integer number of TQs and segments are non-overlapping. The bitrate is selected by programming the width of the TQ and the number of TQ in the various segments.

1. ***synchronization segment***: it is used to synchronization the various node, only the receiver nodes have to adjust their own clock during the receiver of the payload. The lenght of the segment is always **1**.
2. ***propagation segment***: if one node transmits to another faraway ones (geographically speaking) how we can synchronize the first *TQ* of the *synchronization segment*? The **propagation segment** allow the signal propagation across the network and through the nodes. This segment it could be compose from *1 TQ* to *8 TQs* and it is necessary to compensate for signal propagation delays on the bus line and through the electornic interface circuit of the bus nodes.
3. ***buffer segment one & buffer segment two***: this two segment it could have a programmable lenght between *1 TQ* and *8 TQs*. Between this two segment there is the ***sample point***. This point is used from the node to sample the information through the bus channel. This two segment are used to the ***re-synchronization***, in some circumstances we need to compensate the oscillator tolerances within the different CAN nodes.

Jump Width

The *jump width* is the amount of *TQs* that we can add (in the *phase buffer segment one*) or remove (in the *phase buffer segment two*) that permit to adjust the lenght during the *re-synch*.

Nowadays in many CANBus Modules the *propagation time segment* and *phase buffer segment one* are combined in a new segment named ***timing segment 1*** (the *phase buffer segment two* is renamed in ***timing segment 2***).

Dynamic Sample Position: programming the sample point position allow **flexibility**:

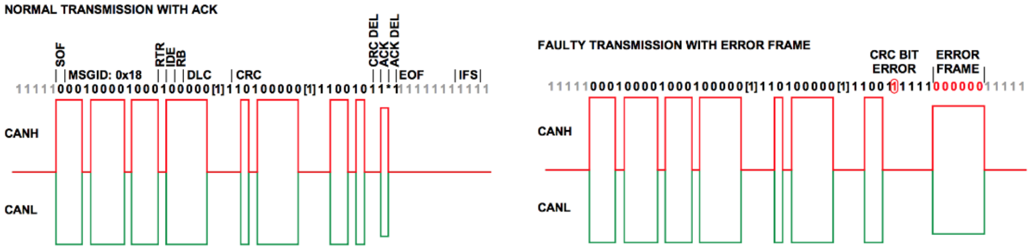
1. ***early sample***: decrease the sensitivity to oscillator tolerances and permit to use lower cost oscillators.
2. ***late sampling***: allow maximum signal propagation time (**reachability**), maximum bus lenght and poor bus topologies can be handled (more *time quanta* in the *propagation segment*).

CANBus Error: there are six possible different error:

1. ***Bit-Error***: write *logical 0* over the bus and sense a *logical 1* (or viceversa). In general if a transmitting ECU detects an **opposite bit** level on the CANBus we have a **bit-error**.
   - ECU writes *logical 0* and reads *logical 1* → very bad error.
   - ECU writes *logical 1* and reads *logical 0* → it is "possible" when there is the **bitwise arbitration** or it is expected that the bus state will change to dominant as other nodes acknoledge the message
2. ***Stuff Error***: reminder on the *bit stuffing*: it needs one opposite bit stuffed each 5 consecutive bits, it is used only from the beginning of the frame to the CRC delimiter. From the ACK field to the end is used the **fixed-form bit fields**. Each node receiving a message that breaks the bit stuffing rules will transmit an **error frame**.
3. ***Format Error***: if one of the *CRC delimiter field*, *ACK field* or *End Of Frame* have an divergent form, the receiving nodes perform a check to ensure these are

correct, if not send a **error frame**.

4. ***CRC Error***: *CRC delimiter field* is the only weapon to ensure the integrity of the message, it depends on the polynomials division, if the *CRC checks* (the reminder of message plus CRC divided by the Generator) is not 0 it generates a **CRC Error**.
5. ***General Error***: the seven **recessive** bits in the *EOF* are used to inform the CANBus nodes about a general error occured during the transmission. If a receiver node found out an error, it writes six consecutive "**zeros**" forcing an error in the current frame that can be captured from everyone.



6. ***ACK Error***: it happen when no one of the receiver nodes write on the buses an **dominant** bit in the *ACK field* of the transmitting frame.

> ***CANBus ACK***
>
> The transmitting nodes, after the DATA and the CRC, write in the bus a *logical 1* (**recessive**) and it hopes, in the mean time, that **at least** one receiver write a *logical 1* (**dominant**) in the ACK bit, if not the transmitting node (reads on the bus *logical 1*) and will resend the message.
> There is two bits for the *ACK field* to absorbe possible delay. We need to allocate space for "not perfect synchronized receiver" to push a **dominant** bit on the bus.

The *ACK* is triggered by another node so the voltage value could be slightly different. These technologies have some implication on the CANBus protocol, like:

- also the recevier node/s can (have to) transmit during specific frame slot (the *ACK field* or *EOF*).

- all the receiver must check the *CRC* very quickly in order to know if the message have pass the integrity checks.
- a CANBus network ***must have at least two nodes to work***, because with only one node no one can acknoledge a message.

For the calculous of the time in the circuit (in the CANBus controller) it is normally used ***time crystal***, the smallest *ICs* possible is the $8MHz$ *time crystal.* If we consider each clock cycle for the smallest unit in CANBus (*time quanta*) for each bit we have at least *8 TQs* (up to *25 TQs*).
If we minimize the size of the of a single bit we have to consider *8 TQs.* $\frac{8MHz}{8TQs} = 1MHz$ we can obtain the maximum bitrate for the CANBus.

CANBus Recap:

1. ***low cost***: the price is **always** a costraint, with it's two wires has a good price-performance tradeoff. This enables the use of CANBus outside the autonomotive domain.
2. ***reliability***: CANBus has sophisticated error detection and handling mechanisms. If failed the integrity checks of the frame it could repeat the sending of the same data and every nodes are informed about the error. CANBus has high immunity to EMI.
3. ***latency***: CANBus means real-time (soft) because there is low latency between transmission and request and actual start of transmission. CANBus has inherent arbitration on message priority due to the bitwise arbitration logic.
4. ***flexibility & speed***: CANBus nodes are "plug & play" and there are not limited number of nodes into a network.
5. ***multi master operation***: (ECU peers) each nodes is able to access to the bus, if there is a fulty nodes the bus communication is not disturbed and they switch-off from the communication.
6. ***broadcast capabilities***: message can be sento to single/multiple nodes and every node simultaneously receive common data.
7. ***Standardize***: *ISO-DIS 11898* (high speed), *ISO-DIS 115192-2* low speed.

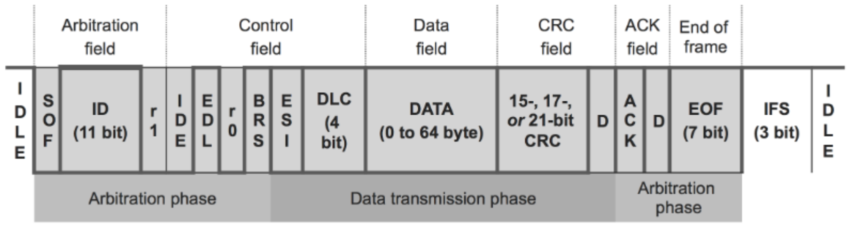
2.4 Controller Area Network Flexible Data-Rate

The ***CAN-FD*** is the evolution of the *CANBus*. The mainly disadvantages of CANBus are: $1MHz$ in some circumstances are not enough and only *8 bytes* of payload are often restrictive. To be compliant to standard CANBus the *arbitration phase* (before the data) and *ACK phase* (after the data) must be mantein to the same frequency.
CAN-FD data frames can be transmitted with two different bit-rates, in the *arbitration phase* and in the *ACK phase* the bitrate depends on the network topology and it is limited to $1MHz$, instead in the *data phase* the bitrate is limited by the *transceiver characteristics*:

- support a bitrate higher than $1MHz$.
- support a payload larger than *8 bytes*.

The increase of the frame speed is possible by shortening the bit time. We define the ***Bit Rate Shift - BRS*** is the bit in the *control field* used to inform **ALL** the nodes that sender will transmit faster in the *data transmission phase* and the ***Extended Data Lenght - EDL***. The implication of this change are:

- **larger payload**: it needs more *CRC bits* to maintain the robustness of CANBus, to have more *CRC bits* it needs a **larger generator**.
- **shorter bit-time**: new bit-time logic in the state machine, a *factor* is introduce between the bit time during arbitration phase and the bit-time during the transmission. The tipical factor is 8 $\rightarrow$ considerering the fastest rate of CANBus *arbitration phase* and the longer header and CRC, the final result is more or less $6MHz$.



To summarize the **CAN-FD** could reach in the *data transmission phase* the speed transmission of $6MHz$ and the possibility of sending a payload large up to **64 bytes**

2.5 Local Interface Network