

Università degli studi di Modena e Reggio Emilia
Dipartimento di Ingegneria Enzo Ferrari

Automotive Connectivity

Anno Accademico 2024/25

Indice

| | | |
|----------|---|----------|
| 1 | Introduction | 1 |
| 1.1 | Structure and Content | 1 |
| 1.2 | Intra-Vehicles | 2 |
| 1.3 | Architectures | 2 |
| 1.4 | Basic Knowledge | 3 |
| 1.4.1 | Multiple Access Protocols | 3 |
| 1.4.2 | Bit Coding | 5 |
| 2 | Intra-Vehicles | 6 |
| 2.1 | ISO/OSI Layers | 6 |
| 2.2 | Network Topology - The Bus System | 8 |
| 2.3 | Controller Area Network | 9 |

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 and there was a dedicated wires connect sensors and actuators with less electronic than now, until they reach the greatest goal of evolution in the automotive sector: autonomous drive. The complexity of the number of connections from each ECU's to the other, also the number of ECU's for each car, is growing. While the number of signals increase in a linear 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 reached a cap for two main reasons: the cost and the space inside the car. Traditionally one ECU is responsible for one task, but nowadays it could be two types of trends:

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

1.3 Architectures

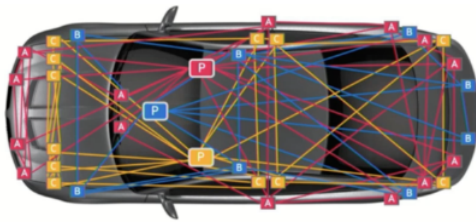


Figura 1.1: *Domain Architecture*

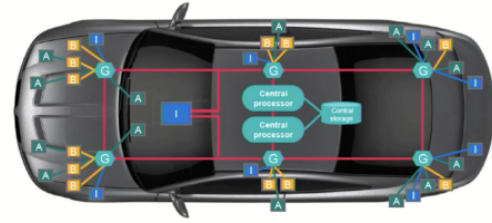


Figura 1.2: *Zonal Architecture*

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

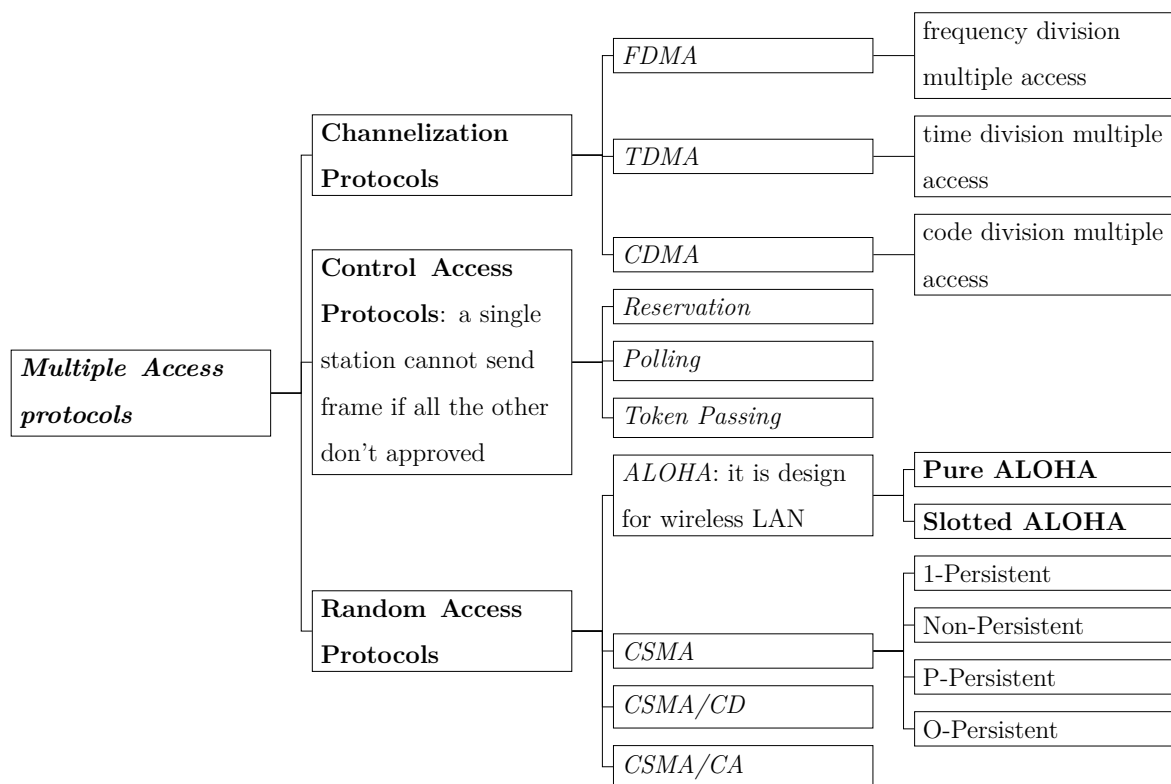
1. local ethernet per zone (**G**)
2. ultra high-speed secured backbone between zone
3. centralized software
4. central computer storage

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 automotive 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 synchronization overhead tha *CSMA*.

1.4.2 Bit Coding

The first thing is to introduce the *Electromagnetic Interference - 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).

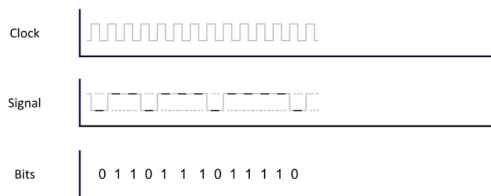


Figura 1.3: *Non Return to Zeros*

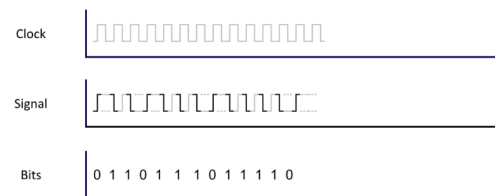


Figura 1.4: *Manchester Coding*

In the *Non Return to Zero* the digital ones is, usually, the positive voltage, while digital zeros are represented by other significant 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 sincronization problem between the clock of the transmitting node and the receiving nodes it is fundamentals.

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 composed by three components: **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 maintain 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 composed by two components: **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 sent/received 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 includes also the *overload notification* and *recovery management* in the case there is an error on the communication they could ask to re-transmit the data.
2. **medium access control**: its 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 known 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 automotive 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 automotive 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.

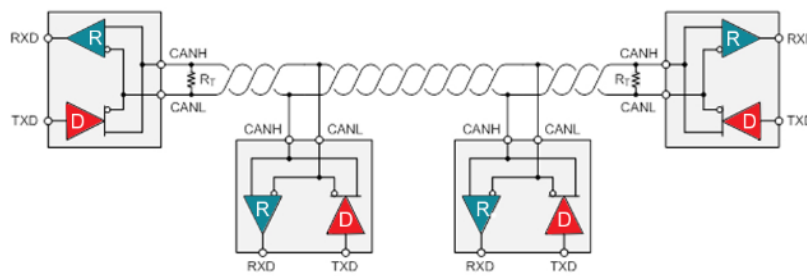


Figura 2.4: CANBus Network Topology

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 redundancy check** is used to ensure data integrity.
- **ACK:** is the **acknowledgement** this slot indicates if the CRC is OK.
- **EOF:** is the **end of frame** marks the end of CAN message.

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) independently 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 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 already 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 ingredient is how to avoid the impending damage born from the collision? (*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 <i>arbitration</i> → stop sending and re-start sensing | | | | | | |

| wired-and bus logic | | | arbitration logic | | |
|---------------------|-----------------|-----------|-------------------|-----|----------------|
| sender <i>a</i> | sender <i>b</i> | bus level | sender | bus | interpretation |
| 1 | 1 | 1 | 0 | 0 | next |
| 1 | 0 | 0 | 0 | 1 | <i>fault</i> |
| 0 | 1 | 0 | 1 | 0 | stop |
| 0 | 0 | 0 | 1 | 1 | next |

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*.

We alrely 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.