**Módulo 1 – Introdução a CAN e CANoe**

1. **Introdução**

Este tutorial, tem como objetivo principal a criação de um ambiente de simulado na ferramenta CANoe. Para isso, abordaremos inicialmente a eletrônica embarcada presente nos veículos atuais, e a arquitetura de comunicação utilizada por eles.

1. **O que é uma ECU?**

Atualmente um veículo comum possui mais de 70 ECU’s presentes nele, mas o que seria uma ECU? Uma Electronic Control Unit (ECU), é simplesmente um módulo eletrônico responsável por executar uma função específica no veículo.

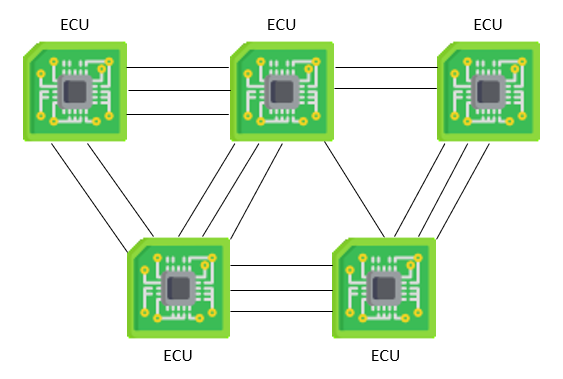
Através de um sistema microcontrolado, uma ECU utiliza de entradas e saídas, para fazer a leitura de sensores e ativação de atuadores. Entre as mais conhecidas temos o Módulo ABS e o Módulo de Injeção Eletrônica.

1. **Barramento CAN**

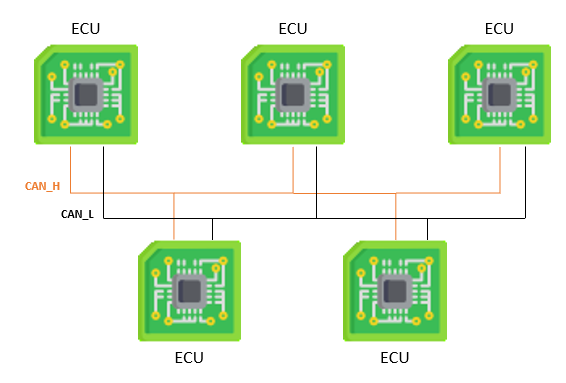
O protocolo Controller Area Network (CAN) foi desenvolvido pela empresa Bosch, com o objetivo de ser um meio de comunicação robusto, e capaz diminuir a quantidade de cabos presentes no chicote do veículo.

Antes desse protocolo ser desenvolvido, a comunicação do veículo era centralizada. Ou seja, o módulo recebia o sinal de todos os sensores que utilizava diretamente em seus pinos de entrada.

Por esse motivo, caso a informação de um sensor fosse necessária em 4 módulos diferentes, cada uma dessas ECU’s teria um fio ligado a esse mesmo sensor ou a uma central que recebesse o sinal desse sensor, aumentando assim a quantidade de fios presentes no chicote do veículo e a quantidade de pinos de entrada em cada módulo.



Com a criação do barramento CAN, houve uma grande mudança no modo que é feito a comunicação. A comunicação entre centrais no barramento CAN passou a ser realizada através de apenas 2 fios (CAN\_H e CAN\_L).



Como exemplo de funcionamento do barramento CAN, temos um sensor de velocidade do veículo que é interligado ao Módulo ABS através de um fio, e o módulo ABS, através do barramento CAN, envia uma mensagem com a informação lida desse sensor para todas as outras centrais presentes no veículo, não requerendo que o sensor esteja ligado através de fio a nenhuma dessas centrais. Com isso, é possível reduzir drasticamente a quantidade de fios do chicote, sobrando mais espaço para inserção de novas centrais.

1. **Características CAN**

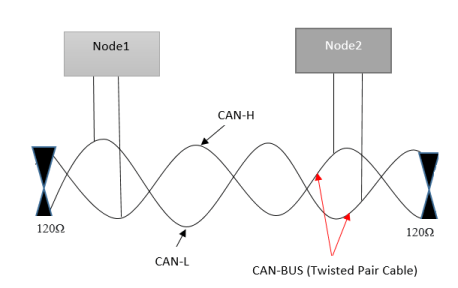
Quando estamos trabalhando com o setor automotivo, estamos lidando com um ambiente suscetível a muito ruído, por esse motivo é necessário um protocolo de comunicação muito robusto, e capaz de detectar falhas.

Por esse motivo, o CAN possui algumas características muito importantes para atender esses requisitos.

Abaixo abordaremos algumas características importantes para o CAN:

**Par trançado diferencial**

Para evitar interferências eletromagnéticas, os dois fios da CAN são utilizados em par trançado, e a sua transmissão é feita utilizando sinalização diferencial, sendo os dois sinais complementares (CAN\_L e CAN\_H), um sinal sempre é o inverso do outro.



(Embedclogic)

A faixa de valores de tensão em que CAN\_H e CAN\_L trabalham também são diferentes sendo próximo aos valores abaixo:

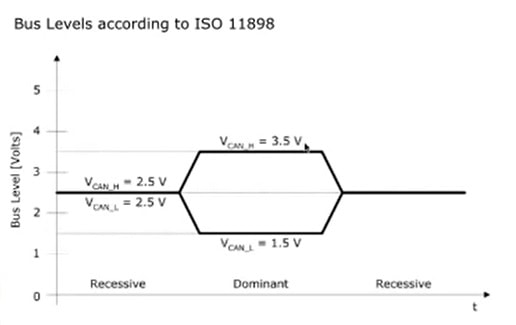
|  |  |  |
| --- | --- | --- |
|  | CAN\_L | CAN\_H |
| 0 | 1,5 V | 2,5 V |
| 1 | 2,5 V | 3,5 V |

**Bits dominante e recessivo**

Por se tratar de dois sinais complementares, o receptor desse sinal extrai as informações através da diferença de tensão entre esses dois sinais. Quando se trata de CAN, temos dois estados no barramento o Dominante e o Recessivo.

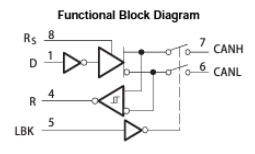
**- Dominante:** Quando enviamos nível lógico **‘0’** através da CAN, temos o estado dominante, que é quando temos uma diferença de potencial de aproximadamente 2 Volts entre CAN\_H e CAN\_L.

**- Recessivo:** Quando enviamos nível lógico **‘1’** através da CAN, temos o estado recessivo, que é quando temos uma diferença de potencial próxima de 0 entre CAN\_H e CAN\_L.



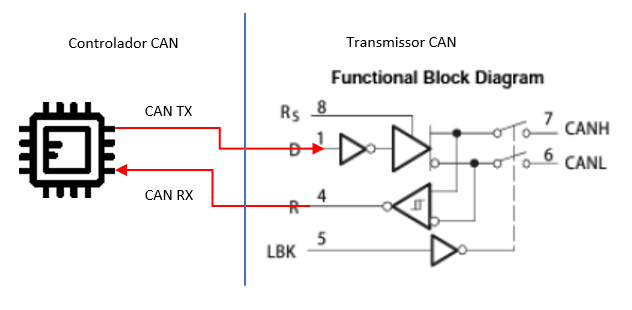
(ATICLE WORLD,2019)

Para entendermos melhor o conceito de Dominante e Recessivo, e também de prioridade de mensagens que será abordado posteriormente, temos a imagem a seguir com o diagrama de blocos funcional de um transmissor CAN.



(DATASHEET XXXX)

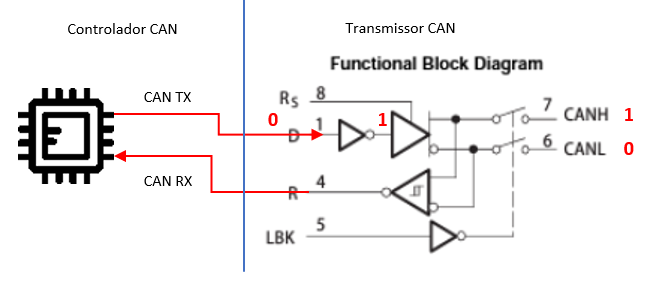
O transmissor CAN, tem a função de receber os sinais ‘0’ e ‘1’ disponibilizados pelo controlador CAN através do pino TX e traduzi-los para os estados Dominante e Recessivo, e também de enviar os dados que estão trafegando na CAN para o pino RX do controlador.



(Baseada em FASTBit)

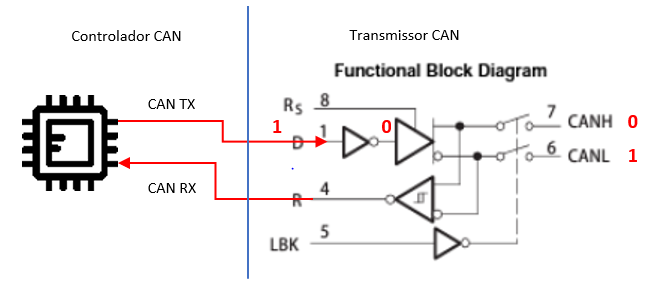
Seguindo a imagem acima, podemos ver que sinal enviado pelo Controlador CAN, entra no pino 1 desse transmissor e logo após passa por um inversor, e antes de sair para o sinal CAN\_L passa por outro inversor, tornando o CAN\_L o oposto de CAN\_H, como dito anteriormente.

**Enviando um bit dominante:**



Ao enviar um nível lógico ‘0’ ao transmissor, em sua saída teremos um bit dominante.

**Enviado um bit recessivo:**



Ao enviar um nível lógico ‘1’ ao transmissor, teremos um bit recessivo.

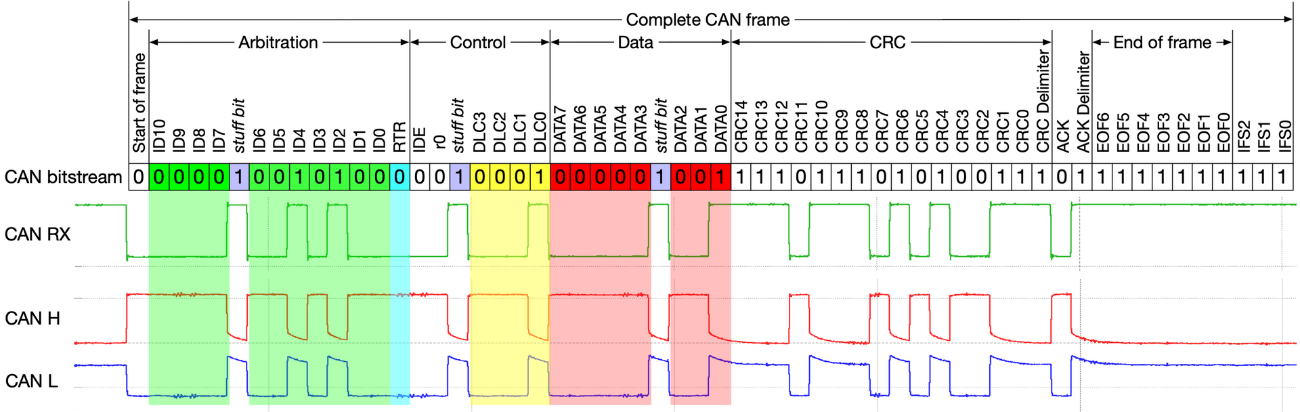
1. **Mensagens CAN**

As mensagens CAN’s são divididas em dois diferentes formatos: o padrão (**Standard)** e o estendido (**Extended)**.

A diferença entre os dois formatos é a quantidade de bits presentes no Identificador (**ID**) da mensagem. O formato Standard possui 11 bits identificadores, já o formato Extended possui 29 bits.

O formato estendido foi desenvolvido com o objetivo de aumentar a quantidade de diferentes mensagens possíveis em um barramento. É mais comum em veículos pesados, que necessitam de uma maior quantidade de mensagens.

Os dois tipos de mensagem podem ser enviados no mesmo barramento. Através do bit **IDE** o controlador CAN consegue identificar se um frame é estendido através de um bit recessivo.



A estrutura básica de todo frame é composta pelos seguintes bits:

**Start-of-frame:** bit dominante (0), que indica o início de uma mensagem.

**Identifier:** identificador da mensagem, também decide a prioridade da mesma, é composto por apenas uma parte de 11 bits no formato padrão. Caso o frame seja no formato estendido, haverá uma segunda parte com mais 18 bits.

**RTR (Remote transmission request):** durante a comunicação no barramento, é possível que as centrais requisitem informações de outras centrais, o bit RTR faz a identificação se a mensagem presente no barramento é uma mensagem de dados (RTR = 0) ou se é uma mensagem de requisição (RTR = 1).

**SSR (Substitute remote request):** bit transmitido apenas nas mensagens estendidas, ocupa o mesmo lugar do RTR nas mensagens padrão. Para que uma mensagem de 29 bits não tenha prioridade sobre uma mensagem de 11 bits, o valor desse bit sempre será recessivo (1).

**IDE:** bit que determina se uma mensagem é padrão ou estendida. No caso de um bit dominante (IDE = 0) temos uma mensagem no formato Standard e no caso de um bit recessivo (IDE = 1) temos uma mensagem no formato Extended.

**Reserved bits (R1,R0):**

**DLC (Data Lenght Code):** bits que determinam a quantidade de bytes de dados que a mensagem terá, pode variar de 0 a 8 bytes.

**Data:** bits que carregam as informações da mensagem, tem seu tamanho baseado na quantidade de bits estipulados pelo DLC.

**CRC:**

**CRC delimiter:**

**ACK:**

**ACK delimiter:**

**End-of-frame:**

1. **Arbitrariedade**

Como a CAN se trata de uma rede em que todos os nós (ECU’s) podem enviar mensagens, é necessário que isso ocorra de forma que uma nenhuma mensagem sobrescreva a outra.

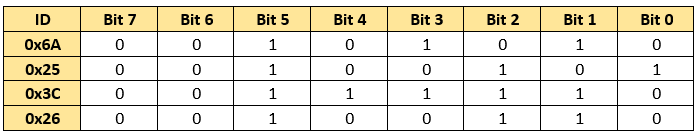
Para isso existe um sistema de arbitrariedade, aonde a mensagem com a maior prioridade sempre terá preferência em relação as de menor prioridade. Ou seja, se 10 mensagens precisam ser enviadas, a de maior prioridade será a primeira a ser enviada, e a de menor prioridade será a décima.

**Exemplo:**

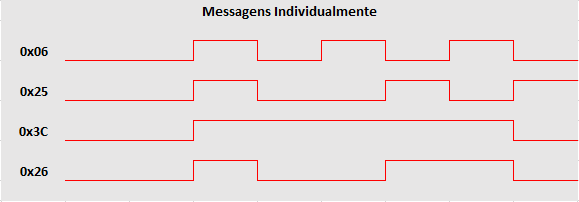
Num cenário onde 4 ECU’s diferentes necessitam enviar 4 mensagens com os respectivos ID’s: 0x6A, 0x25, 0x3C e 0x26. Qual mensagem teria prioridade e como funciona a arbitrariedade?

Como dito anteriormente, um bit Dominante tem prioridade sobre um bit Recessivo, e isso se aplica também quando falamos de arbitrariedade. O nível de prioridade de uma mensagem é definida pelo seu ID. Quanto menor for o ID, maior a prioridade da mensagem.

Transformando os ID’s acima em binários, temos os seguintes valores:



Considerando um cenário onde as mensagens sejam enviadas separadamente, cada uma delas seria enviada da seguinte maneira (CAN\_H):



Num cenário que todas as mensagens precisam ser enviadas ao mesmo tempo, é necessário que a mensagem de maior prioridade seja enviada, sem nenhuma intervenção das outras. As ECU’s fazem o tratamento dessa arbitrariedade através da própia leitura da CAN, utilizando dos bits dominantes e recessivos.

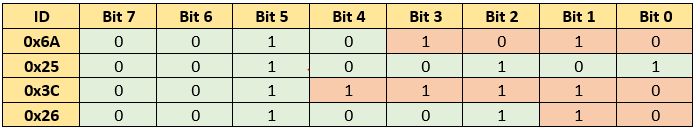
Primeiramente, para entendermos melhor a questão de arbrietariedade, é necessário ter noções de como se inicia o envio de uma mensagem CAN na rede. Todas as mensagens a serem enviadas, se iniciam juntas, sincronizadas bit a bit. O evento de ínicio acontece após as ECU’s perceberam um **“BUS IDLE”** na rede CAN, esse **“BUS IDLE”** seria o envio de XX bits Recessivos seguidos.

Ao perceberem que XX bits Recessivos foram enviados em sequência, as ECU’s podem iniciar a transmissão da mensagem através do envio de um bit Dominante, denominado **“Start of Frame”**, após esse bit, é enviado o ID da mensagem que decide a prioridade da mesma.

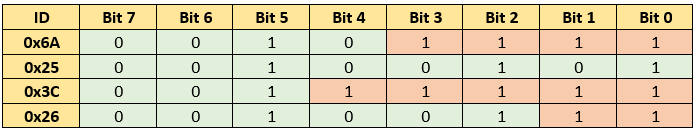
Para que não haja interferência na mensagem de maior prioridade, após o envio de cada bit, a ECU checa o valor presente na CAN, se uma ECU acabou de enviar através do seu pino **TX** um bit Recessivo, porém recebeu no seu pino **RX** um bit Dominante, ela reconhece que ela é uma mensagem de menor prioridade, e passa a enviar apenas bits Recessivos para não atrapalhar as mensagens de maior prioridade, e se mantém assim até o ínicio do próximo ciclo de mensagem, onde ela tentará novamente.

Abaixo serão apresentadas tabelas para melhor entendimento do conceito apresentado acima.

Na primeira tabela, os bits estão coloridos de vermelho apartir do momento que uma ECU envia um bit Recessivo mas recebe um Dominante.



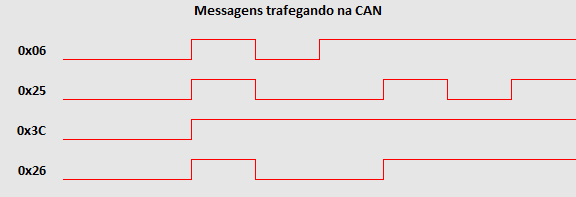
Na segunda tabela, temos o comportamento real de uma mensagem no barramento CAN, apartir do momento que uma ECU envia um bit Recessivo mas recebe um Dominante, todos os próximos bits enviados pela ECU são bits recessivos.



Podemos ver que a mensagem **0x25** que possui o menor ID, manteve sua prioridade e não teve interferência no envio de dados de nenhuma outra mensagem.

Como todas as outras ECU’s se manterão enviando bits Recessivos até o próximo **“BUS IDLE”**, a mensagem de ID 0x25 não terá problemas com interferência ou dados sobrescritos.

Abaixo temos uma imagem de como os sinais se comportarão na CAN:



É possível notar o momento que uma ECU entende que sua mensagem não é prioritária e começa a enviar apenas bits Recessivos.

1. **Ferramenta CANoe**

O CANoe é um software desenvolvido pela empresa **Vector Informatik GmbH** e é muito utilizado no ramo automotivo para a criação de simulações de ambientes e outras coisas. Com ele é possível fazer testes em bancada de ECU’s individualmente, simulando as mensagens e sinais das outras ECU’s do sistema.

É uma ferramenta muito completa, onde também é possível fazer análises, gravações de Log, executar testes automáticos e etc. Justamente por ser uma ferramenta com muitas funcionalidades, nesse tutorial não iremos abordá-la por completo.

Como dito anteriormente, nosso foco será a criação de um ambiente simulado. Para isso posteriormente abordaremos conceitos como a criação e utilização de DBC’s, variáveis de sistema, programação CAPL, painéis de controle, etc.

1. **Referências**

**https://embedclogic.com/can-protocol-protocol-to-broadcast-message-on-a-network/standard-can-vs-extended-can/**