MANUAL DE ESTUDIO · MECATRÓNICA AUTOMOTRIZ

Buses CAN, LIN, FlexRay, MOST y Ethernet automotriz

Capítulo 04 · En el video: **minuto 44:56** · Duración 15:17

Objetivo. Explicar cómo CAN resuelve colisiones sin perder tiempo y cómo se detectan los errores de transmisión.

Ideas clave

Capa física CAN

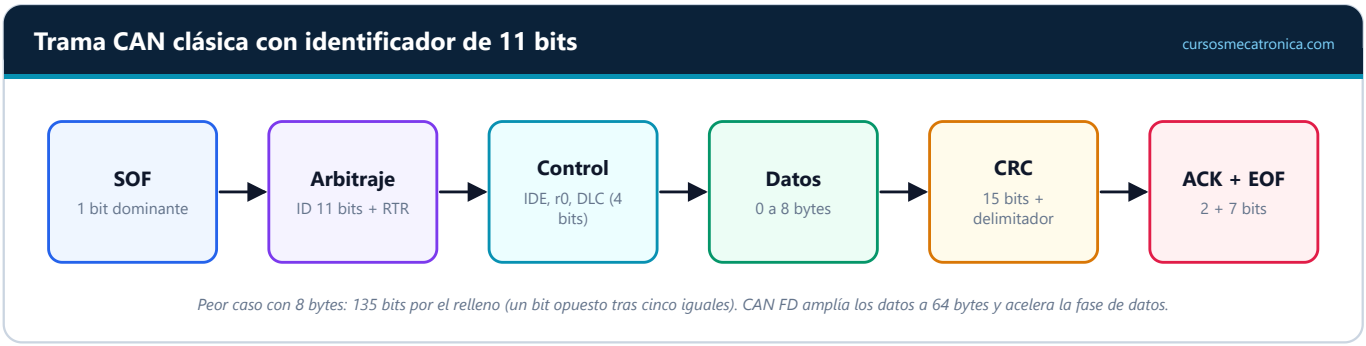
Par trenzado CAN_H/CAN_L con terminaciones de 120 Ω en los extremos. En recesivo ambos quedan en ~2,5 V; en dominante CAN_H sube a ~3,5 V y CAN_L baja a ~1,5 V. La señal diferencial rechaza el ruido común.

Arbitraje bit a bit

Los identificadores no son direcciones sino prioridades del contenido. Todos transmiten a la vez; el bit dominante (0) se impone en el cable. Quien envía un 1 y lee un 0 deja de transmitir: el mensaje de ID menor sigue sin perder ni un bit.

Trama y errores

Tras cinco bits iguales se inserta uno opuesto (relleno) para mantener la sincronización. El CRC-15 detecta errores; el receptor confirma en el bit ACK. Los contadores de errores llevan a un nodo defectuoso a "bus off" para que no bloquee la red.



Campos de la trama del capítulo 04.

CAN FD y CAN XL

CAN FD conserva el arbitraje a velocidad normal y acelera la fase de datos (2 a 8 Mbit/s) con hasta 64 bytes por trama. CAN XL llega a 2048 bytes y unos 20 Mbit/s, como puente hacia Ethernet.

LIN, FlexRay y MOST

LIN es un bus de un cable de bajo costo con un maestro que agenda los mensajes. FlexRay combina un segmento estático (TDMA) y uno dinámico, con relojes sincronizados y guardián de bus. MOST, un anillo óptico de audio, fue reemplazado por Ethernet.

Ethernet automotriz

100BASE-T1 y 1000BASE-T1 usan un solo par trenzado; los conmutadores con TSN garantizan latencia. Con ISO-TP, los mensajes largos de diagnóstico se segmentan en una primera trama y tramas consecutivas.

Fórmulas y relaciones

Relación	Expresión	Cómo leerla
Polinomio CRC-15	$x^{15} + x^{14} + x^{10} + x^{8} + x^{7} + x^{4} + x^{3} + 1$	0x4599.
Tiempo de bit	$t_{bit} = 1/\text{bitrate}$	2 µs a 500 kbit/s.
Largo máximo del bus	$L \approx (t_{bit}/2 - t_{nodos}) \cdot v$	El bit debe recorrer el bus ida y vuelta.
ISO-TP	$FF + \lceil (n-6)/7 \rceil \, CF$	Mensaje de n bytes en tramas de 8.

Ejemplo resuelto: arbitraje entre cuatro nodos

1. Transmiten a la vez 0x123, 0x120, 0x7FF y 0x0F0.
2. Bit 1: 0x7FF envía 1 y lee 0: se retira.
3. Bit 3: 0x123 y 0x120 envían 1 y 0x0F0 envía 0: ambos se retiran.
4. Gana 0x0F0, el identificador de mayor prioridad (menor valor).

Una trama 0x0A0 con 8 bytes lleva 13 bits de relleno y dura 124 bits = 248 µs a 500 kbit/s. 64 bytes por CAN FD (500k/2M) tardan unos 347 µs frente a 2160 µs en ocho tramas clásicas (can_arbitraje_y_crc.py).

Niveles del bus CAN de alta velocidad (ISO 11898-2)

Estado	CAN_H	CAN_L	Diferencial	Bit
Recesivo	≈ 2,5 V	≈ 2,5 V	≈ 0 V	1
Dominante	≈ 3,5 V	≈ 1,5 V	≈ 2 V	0

Código del capítulo

Los archivos completos vienen en el paquete ZIP del capítulo (carpeta `codigo/`).

can_arbitraje_y_crc.py — arbitraje bit a bit, CRC-15, relleno de bits, duración de tramas y segmentación ISO-TP. *(extracto: primeras 60 de 83 líneas)*

```
"""
can_arbitraje_y_crc.py — Capítulo 04: arbitraje bit a bit, CRC-15, relleno de bits,
duración de tramas CAN y CAN FD y segmentación ISO-TP.
"""


def arbitraje(ids):
    """Bus de 'Y cableado': el 0 (dominante) gana. Devuelve el ganador y quién perdió en qué bit."""
    activos, perdidas = list(ids), {}
    for bit in range(10, -1, -1):
        nivel = min((i >> bit) & 1 for i in activos)
        for i in [i for i in activos if (i >> bit) & 1 != nivel]:
            perdidas[i] = 11 - bit          # posición 1 = primer bit transmitido
            activos.remove(i)
    return activos[0], perdidas


def crc15(bits):
    crc = 0
```

```
    for b in bits:
        siguiente = b ^ ((crc >> 14) & 1)
        crc = (crc << 1) & 0x7FFF
        if siguiente:
            crc ^= 0x4599
    return crc


def a_bits(valor, n):
    return [(valor >> k) & 1 for k in range(n - 1, -1, -1)]


def trama(id_std, datos):
    bits = [0] + a_bits(id_std, 11) + [0, 0, 0] + a_bits(len(datos), 4)
    for byte in datos:
        bits += a_bits(byte, 8)
    return bits + a_bits(crc15(bits), 15)


def relleno(bits):
    """Cuenta los bits de relleno: tras 5 iguales se inserta uno opuesto."""
    n, previo, racha = 0, None, 0
    for b in bits:
        if b == previo:
            racha += 1
        else:
            previo, racha = b, 1
        if racha == 5:
            n += 1
            previo, racha = 1 - b, 1
    return n


def isotp(carga):
    """Segmentación ISO 15765-2 en tramas clásicas de 8 bytes."""
    if len(carga) <= 7:
        return [[len(carga)] + carga]
    tramas = [[0x10 | (len(carga) >> 8), len(carga) & 0xFF] + carga[:6]]
    resto, sn = carga[6:], 1
    while resto:
        tramas.append([0x20 | sn] + resto[:7])
```

Errores frecuentes

- Olvidar una de las dos terminaciones de 120 Ω (reflexiones y errores).
- Asignar IDs bajos a mensajes poco importantes y retrasar los críticos.
- Medir CAN_H y CAN_L respecto de masa y no en modo diferencial.

Autoevaluación

1. ¿Qué bit gana en el bus CAN?
2. ¿Cuántos bytes lleva una trama CAN FD?
3. ¿Para qué sirve el relleno de bits?

Respuestas. 1. El dominante (0). **2.** Hasta 64. **3.** Para asegurar flancos que mantengan sincronizados a los nodos.