give me a real example demonstrating all that let's say we have that payload "hello" tell me exactly what happen in transmission

**ChatGPT said:**

Let's go through a real example of how the transmission works when sending a simple payload like the string "hello" using the **LIN protocol**. We will break it down into each step, from the creation of the frame to the checksum calculation, to show you how the data is transmitted and validated.

**Step 1: Frame Structure**

We’ll assume this transmission is happening from a **Master ECU1** (sending the message) to a **Slave ECU2** (receiving it).

**The frame structure will look like this:**

1. **Header (Master to Slave)**:
   * **BREAK**: 14 bits
   * **SYNC**: 8 bits
   * **ID (6 bits) + Parity (2 bits)**: 8 bits
2. **Response (Slave to Master)**:
   * **DATA**: 16-64 bits (the actual payload, which is "hello")
   * **Checksum**: 8 bits (to ensure data integrity)

**Step 2: Frame Construction**

Let’s assume:

* The **ID** used by the Master is 0x01 (in 6 bits: 000001).
* The **parity** is calculated based on the **ID**: For ID = 0x01, the parity would be 00 (even parity).
* The **DATA** sent is the ASCII encoding of the word "hello", so we would use the bytes:
  + h → 0x68
  + e → 0x65
  + l → 0x6C
  + l → 0x6C
  + o → 0x6F

Thus, the payload "hello" is: 0x68 0x65 0x6C 0x6C 0x6F

**Step 3: Transmission Breakdown**

**1. Master Sends Header**

* **BREAK (14 bits)**:
  + The **BREAK** field is a long dominant signal (low voltage level) to force synchronization.
  + In the actual transmission, this would appear as a dominant (0) state for 14 bits. For now, we assume the slave is reset and prepared to receive the message.
* **SYNC (8 bits)**:
  + The **SYNC** byte is always 0x55 (binary: 01010101), and this is used by the slave to synchronize the baud rate.
  + In transmission: 0x55
* **ID and Parity (8 bits)**:
  + The **ID** is 0x01, which in 6 bits is 000001.
  + The **parity** is 00 (even parity) because the number of 1s in ID is even.
  + So, the **ID + Parity** becomes: 00000100 (8 bits).

Thus, the **Header** transmitted looks like this:

* **BREAK**: 14 bits of dominant (0)
* **SYNC**: 0x55 (binary 01010101)
* **ID + Parity**: 0x04 (binary 00000100)

**2. Slave Sends Response**

* The **DATA** part contains the payload "hello", which has been encoded in ASCII as:
  + 0x68 (for h)
  + 0x65 (for e)
  + 0x6C (for l)
  + 0x6C (for l)
  + 0x6F (for o)

So, the **DATA** part is: 0x68 0x65 0x6C 0x6C 0x6F.

* Now, we calculate the **Checksum**.

**Step 4: Checksum Calculation**

**Checksum Calculation (Classic or Enhanced)**

Since we’re using **Classic Checksum** (ID 0x01, which is between 0x00 and 0x3B), we **exclude the ID** and only calculate the checksum over the **DATA** bytes:

**DATA bytes**: 0x68, 0x65, 0x6C, 0x6C, 0x6F

**Step-by-Step Calculation:**

1. Add the data bytes together:

0x68+0x65+0x6C+0x6C+0x6F=0x1B50x68 + 0x65 + 0x6C + 0x6C + 0x6F = 0x1B50x68+0x65+0x6C+0x6C+0x6F=0x1B5

1. Take the lower 8 bits and add the carry:

(0x1B5&0xFF)+(0x1B5>>8)=0xB5+0x01=0xB6(0x1B5 \& 0xFF) + (0x1B5 >> 8) = 0xB5 + 0x01 = 0xB6(0x1B5&0xFF)+(0x1B5>>8)=0xB5+0x01=0xB6

1. Perform the bitwise **NOT** (invert all bits):

∼0xB6=0x49\sim 0xB6 = 0x49∼0xB6=0x49

So, the **Checksum** is 0x49.

**Final Response Frame from Slave:**

The **Response** from the Slave will be:

* **DATA**: 0x68 0x65 0x6C 0x6C 0x6F
* **Checksum**: 0x49

**Step 5: Transmission Sequence**

Now, let’s go over the actual **transmission sequence**:

1. **Master to Slave**:
   * The **Master** sends the **Header**:
     + BREAK (14 bits)
     + SYNC (0x55)
     + ID + Parity (0x04)
2. **Slave to Master**:
   * The **Slave** sends the **Response**:
     + DATA (0x68 0x65 0x6C 0x6C 0x6F)
     + Checksum (0x49)

**Step 6: Slave Verification**

Upon receiving the **DATA + Checksum** from the slave, the **Master** performs the checksum validation:

1. It takes the **DATA** bytes: 0x68 0x65 0x6C 0x6C 0x6F.
2. It calculates the checksum over those bytes using the same process outlined earlier.
3. It compares the computed checksum with the one received (0x49).
   * If the checksums match, the **Master** knows that the data was transmitted correctly.
   * If they don’t match, the Master can decide to request a retransmission or take appropriate action.

**Summary of Transmission:**

* **Master sends**: BREAK, SYNC, ID + Parity.
* **Slave sends**: DATA (the actual message "hello") and CHECKSUM.
* **Master verifies** the CHECKSUM to ensure the message was received correctly.

In this example, "hello" was transmitted with a checksum of 0x49 for error detection. If the checksum is valid, the transmission is considered successful.

If you'd like, I can also help you implement the checksum calculation or further details on the LIN protocol!

4o mini