E S P 3 2 CANBUS DEMO V1

Wednesday, April 23, 2025

Ron Kessler

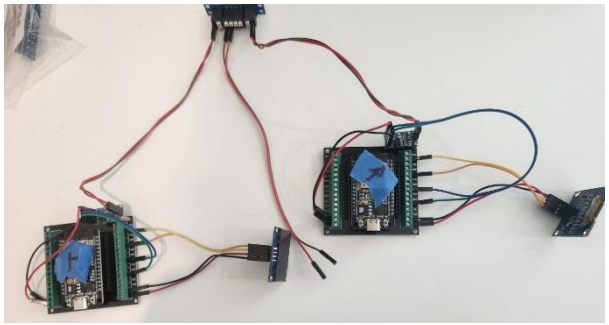

Figure 1: General Layout with two ESP32 Do It Microcontrollers and Start-Link connector for two branch lines.

This paper describes how to connect ESP32 DoIt DevKit breakout boards in a Canbus network and OLEDs for the transmitter and receiver (Figure 1). The transmitter sends a standard packet and an Extended packet every 5 seconds to the receiver. The first message sent is a simple “Hello” string and the extended packet transmits “Old Dog” as its data string. This is the first attempt at using this type of network with ESP32’s and shows the basic operation needed to get a simple network up and running.

While many Arduino CanBus projects use the MCP2515 SPI board to manage the network, this demo uses the small TJA1050 transceiver (Figure 2). In this configuration, the SPI hardware of a MCP2515 is integrated in the ESP32 module. This greatly improves performance and simplifies the design. In Figure 1, the TJA1050 is mounted on the controller’s breakout board using a small standoff.

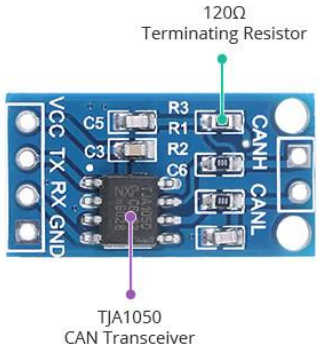

Figure 2: TJA1050 transceiver board manages CANBUS communication via a UTP cable. The MCP2515 is integrated into the ESP32 and is not used here.

This paper assumes the reader has a basic understanding of how Canbus networks are configured and the basic topology of those networks. Consequently, we will focus on understanding the signals that are decoded by the receiver MCU.

Decoding Canbus Packets

Figure 3 shows the CAN signals captured on the oscilloscope. The yellow trace is Can-H and the blue one is Can-L. Each bit is 2µs wide and that equates to a frequency of 500Kb/s (Freq. = 1/.000002). You could examine this bit pattern and decode the message yourself. However, the easiest method of decoding automotive network communications is to use an oscilloscope that can interpret the pulses for us. I used a Pico Automotive 4225 dual-channel USB scope. Refer to Figure 4 as you read this section.

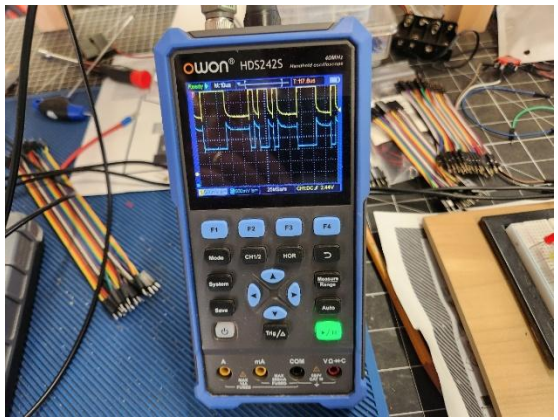

Figure 3: Owon Handheld scope shows the Can-H and Can-L signals.

The different colored sections in the middle of Figure 4 shows some of the different components/fields of a packet:

1. The orange is the packet ID which is #012).
2. Green is for Data Length Code and is a 4-bit number representing the total size of the data payload.
3. The blue section represents the actual data payload.
4. The turquoise is for the cyclical redundancy check which makes sure all the bytes have been received.

Turn your attention to the blue section and notice the first byte is Data-48. The value 48 is a hexadecimal number. In ASCII format, that equates to the letter ‘H’. The next byte is 65 and equates to ‘e’. The remaining fields show 6C, 6C, and 6F. Taken collectively, the hex values: 48, 65,6C,6C, & 6F = ‘Hello’ which is the standard packet received by my ESP32. The last value in the data field = 20 which is simply a blank space. The extended packet (not shown) is sent 5 seconds later and would show up as ‘Old Dog’. The 5 second delay was added to make it easier to read the messages on the OLEDs.

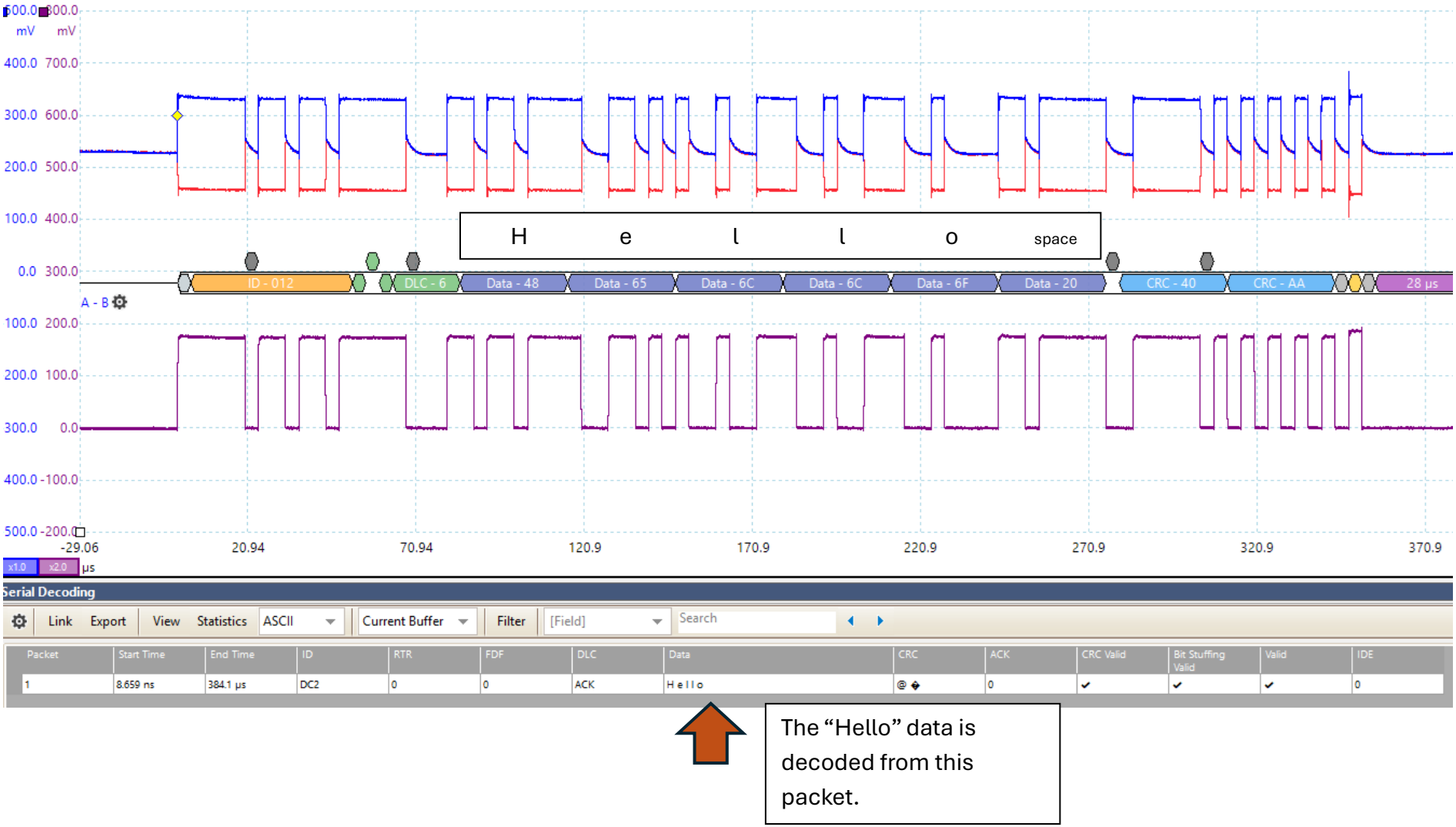

Figure 4: Pico Automotive 6 decoded image of the received packets from this layout. It shows the pulse rate/width and shows the data packet decoded into ASCII. See red arrow that shows "Hello" was received in this packet.

More to come! Stay tuned.

Enter ASCII/Unicode text string and press the *Convert* button:

From

Text

To

Decimal

Open File | Sample | 🔍

Paste text or drop text file

```
Hello
```

Character encoding

ASCII

Output delimiter string (optional)

Space

= Convert | × Reset | ⇅ Swap

```
48 65 6C 6C 6F
```