Technical Article

Introduction to CAN (Controller Area Network)

February 19, 2019 by Stephen St. Michael

This article introduces the Controller Area Network (CAN) serial communication bus, detailing message frames, bus arbitration, and signaling.

This article introduces the Controller Area Network (CAN) serial communication bus, detailing message frames, bus arbitration, and signaling.

Supporting Information

- Serial Communication in Harsh Environments: A New CAN Transceiver from Maxim
- The Why and How of Differential Signaling

What Is CAN?

The Controller Area Network (CAN) is a serial communication bus designed for robust and flexible performance in harsh environments, and particularly for industrial and automotive applications.

Originally invented by Bosch and later codified into the ISO11898-1 standard, CAN defines the data link and physical layer of the Open Systems Interconnection (OSI) model, providing a low-level networking solution for high-speed in-vehicle communications. In particular, CAN was developed to reduce cable wiring, so the separate electronic control units (ECUs) inside a vehicle could communicate with only a single pair of wires. Figure 1 shows the ECUs in a car connected to a CAN bus.

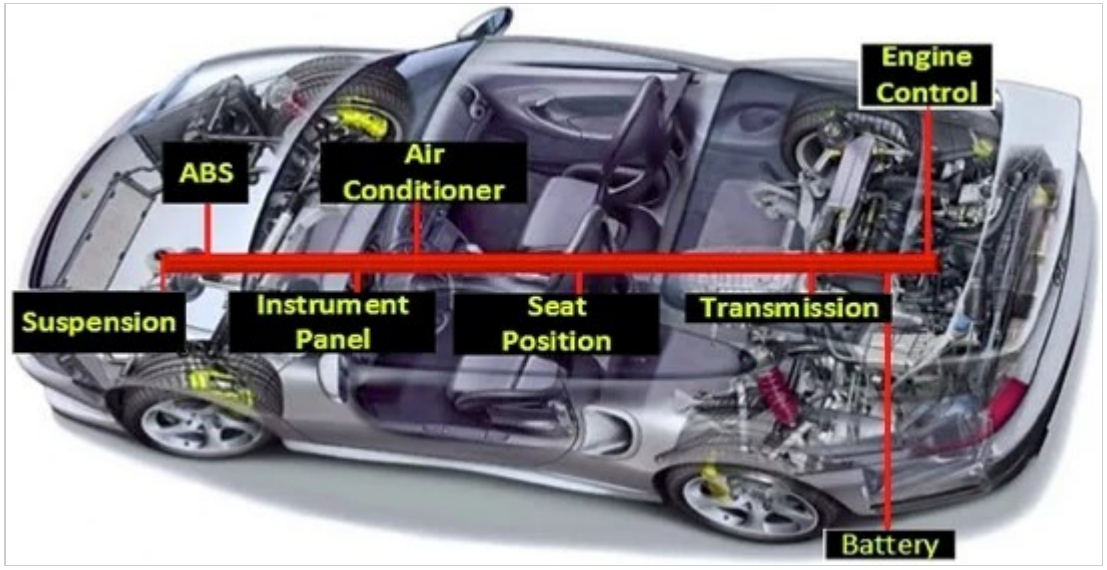

Figure 1. CAN is used for ECU communication in cars. Image used courtesy of the Infosec Institute.

Onboard diagnostics (OBD) is your vehicle's diagnostic and reporting system that allows you or a technician to troubleshoot problems via diagnostic trouble codes (DTCs). When the “check engine” light comes on, a technician will often use a handheld device to read the engine codes off of the vehicle. At the lowest level, this data is transmitted via a signaling protocol, which in most cases is CAN.

DeviceNet is a high-level networking protocol used in industrial applications. It greatly reduces the wiring needed between a control system and I/O devices. Rather than connecting each device to a separate input/output on a PLC's I/O modules, devices can be linked together via a four-wire connector and connected to a network scanner on the PLC. At the lowest level, we find CAN working its magic within the DeviceNet protocol. Figure 2 shows a PLC scanning a network of industrial devices communicating over DeviceNet.

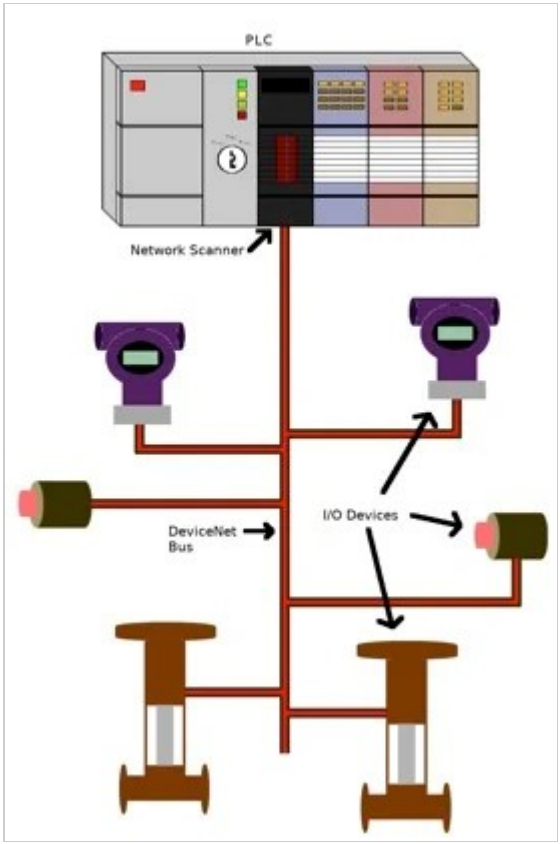

Figure 2. PLC connected to a DeviceNet network that uses CAN for the data link and physical layer.

CAN Message Frames

So what does a CAN message actually look like? The original ISO standard laid out what is called Standard CAN. Standard CAN uses an 11-bit identifier for different messages, which comes to a total of 2^{11}, i.e. 2048, different message IDs. CAN was later modified; the identifier was expanded to 29 bits, giving 2^{29} identifiers. This is called Extended CAN. CAN uses a multi-master bus, where all messages are broadcast on the entire network. The identifiers provide a message priority for arbitration.

CAN uses a differential signal with two logic states, called recessive and dominant. Recessive indicates that the differential voltage is less than a minimum threshold voltage. Dominant indicates that the differential voltage is greater than this minimum threshold. Interestingly, the dominant state is achieved by driving a logic '0' onto the bus, while the recessive state is achieved by a logic '1'. This is inverted from the traditional high and low used in most systems. These two states will be detailed later on in the article. The important thing is that a dominant state overrides a recessive during arbitration.

Standard CAN

The standard CAN message frame consists of a number of bit fields. These are shown in Figure 3.

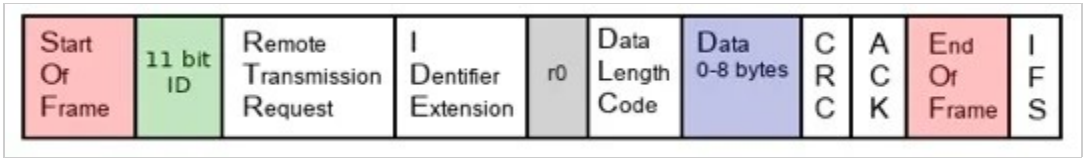

Figure 3. Standard CAN message frame

The first bit is the start of frame (SOF). This dominant bit represents the start of a CAN message. Next is the 11-bit identifier, which establishes the priority of the CAN message. The smaller the identifier, the higher the priority of the message.

The remote transmission request (RTR) bit is normally dominant, but it goes recessive when one node is requesting data from another. The identifier extension (IDE) bit is dominant when a standard CAN frame is being sent and not an extended one. The r0 bit is reserved and not currently used. The data length code (DLC) nibble signifies how many bytes of data are in this message.

Next is the data itself, being as many bytes as represented in the DLC bits. The cyclic redundancy check (CRC) is a 16-bit checksum for detecting errors in the transmitted data. If the message is properly received, the receiving node overwrites the recessive acknowledge bit (ACK) with a dominant bit. The ACK also contains a delimiter bit to keep things synchronized. The end of frame (EOF) signifies the end of the CAN message and is 7 bits wide, for detecting bit-stuffing errors. The last part of a CAN message is the interframe space (IFS), used as a time delay. This time delay is precisely the amount of time needed for a CAN controller to move the received message into a buffer for further processing.

Extended CAN

Extended CAN uses a 29-bit identifier along with a few additional bits. An extended message has a substitute remote request (SRR) bit after the 11-bit identifier, which acts as a placeholder to keep the same structure as standard CAN. This time the identifier extension (IDE) should be recessive, indicating that the extended identifier follows it. The RTR bit is after the 18-bit ID and is followed by a second reserve bit, r1. The rest of the message remains the same.

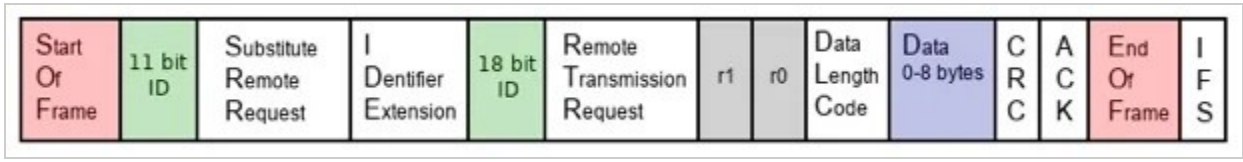

Figure 4. Extended CAN message frame

CAN Message Types

Now that you know what a CAN message looks like, you might be wondering what kinds of messages are passed along the bus. CAN allows for four different message types. They are the data frame, remote frame, overload frame, and error frame.

A standard CAN data frame makes use of the identifier, the data, and data length code, the cyclic redundancy check, and the acknowledgment bits. Both the RTR and IDE bits are dominant in data frames. If the recessive acknowledge bit at the receiving end is overwritten by a dominant bit, both the transmitter and receiver recognize this as a successful transmission.

A CAN remote frame looks similar to a data frame except for the fact that it does not contain any data. It is sent with the RTR bit in a recessive state; this indicates that it is a remote frame. Remote frames are used to request data from a node.

When a node detects an error in a message on the CAN bus, it transmits an error frame. This results in all other nodes sending an error frame. Following this, the node where the error occurred retransmits the message. The overload frame works similarly but is used when a node is receiving frames faster than it can process them. This frame provides a time buffer so the node can catch up.

Bus Arbitration & Signaling

CAN is a CSMA/CD protocol, meaning each node on the bus can detect collisions and back off for a certain amount of time before trying to retransmit. This collision detection is achieved through a priority arbitration based on the message identifiers. Before we discuss arbitration, let's take a closer look at the dominant and recessive bits used on the CAN bus.

Inverted Logic

An interesting aspect of the CAN bus is that it uses an inverted form of logic with two states, dominant and recessive. Figure 5, below, shows a simplified version of a CAN transceiver's output and input. The '101' bitstream is coming from/going to a CAN controller and/or microcontroller. Notice that when the controller sends a stream of bits, these are complemented and placed on the CANH line. The CANL line is always the complement of CANH. For arbitration to work, a CAN device must monitor both what it is sending and what is currently on the bus, i.e., what it's receiving.

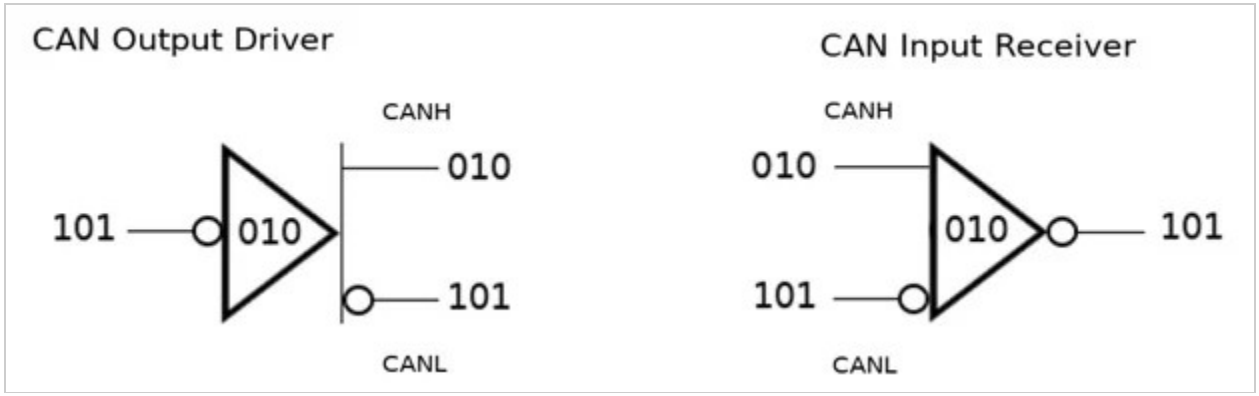

Figure 5. CAN output/input

Figure 6 shows both the CANH and CANL signals simultaneously so that you can see the CAN bus in action. Plotted below the bus signals is the differential voltage that corresponds to the dominant and recessive states of the CAN signals. The first three segments in time, t1–t3, are drawn to match up with the three bits shown above in Figure 5. We will look at this from the perspective of the output driver. The driver's input initially sees a '1' and complements this to a zero, which is placed on CANH. CANL sees the complement of CANH and goes high. This is shown as t1 in Figure 6. Notice that the CANH and CANL voltages are offset from one another. During time t1, CANH – CANL is very close to zero, since CANH and CANL are almost the same voltage. This period, where the driver is sending a logic '1' resulting in CANH and CANL being close to the same voltage, is what we call the CAN recessive state.

The next bit sent is a '0'. CANH gets its complement and CANL again gets the complement of CANH. Notice this time that the CANH and CANL voltages are not close to one another. Therefore, the differential voltage (VDIFF) is larger. This is the CAN dominant state. We say that the logic is inverted because a '1' takes the bus low and a '0' brings it high. The input receiver works in a similar fashion.

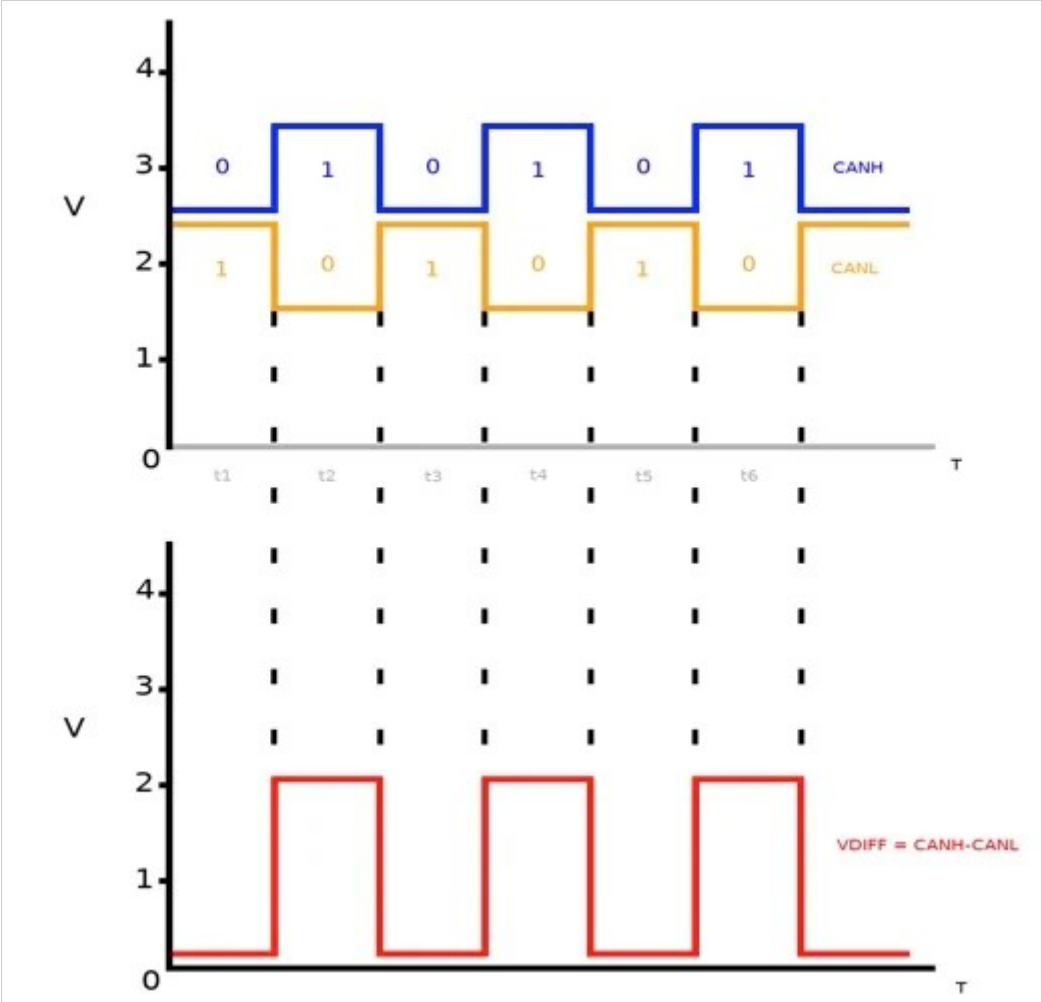

Figure 6. CAN recessive and dominant states with differential voltage shown

Priority Arbitration

As mentioned earlier, the smaller the 11-bit identifier is the greater the message's priority. Every bit that a node transmits, it monitors. This is how a node detects that a higher-priority message is being placed on the bus. The moment a node sends a recessive bit but detects a dominant bit on the bus, it backs off. This is called non-destructive arbitration because the winning message continues to be transmitted without any problem. Notice that a recessive, logic '1', loses out to a dominant, logic '0'. This makes sense since a lower identifier value represents a higher priority. To get a better idea of what this means, take a look at Figure 7, which shows three nodes on a CAN bus attempting to take control. It's important to remember that each time a recessive bit is shown, the controller is sending a '1', while dominant bits correspond to the sending of '0'.

Nodes 1–3 are all sending a stream of bits. This stream of bits represents the message identifiers and their priority. For starters, all three nodes send a '1', which is represented on the CAN bus as a recessive bit. Next, each node sends a '0', or dominant bit. The third bit placed on the bus is another '1', or recessive bit. At this point, none of the nodes have detected any conflict with another node on the bus, so they keep transmitting.

For the fourth bit, Node 1 sends a '0' or dominant bit. Node 2 transmits a recessive bit, yet detects a dominant bit on the bus. It immediately backs off, knowing that a higher-priority message is currently being sent. Node 3 continues to transmit since it read back the same dominant bit that it transmitted. When the fifth bit is placed on the bus, Node 3 then recognizes that it is at a lower priority and stops transmitting. Both Node 2 and Node 3 wait a certain amount of time before attempting again. This is shown on the right half of Figure 7, with Node 3 winning arbitration. As you can see, a logic '0' dominant bit corresponding to a lower message identifier allows the arbitration to take place.

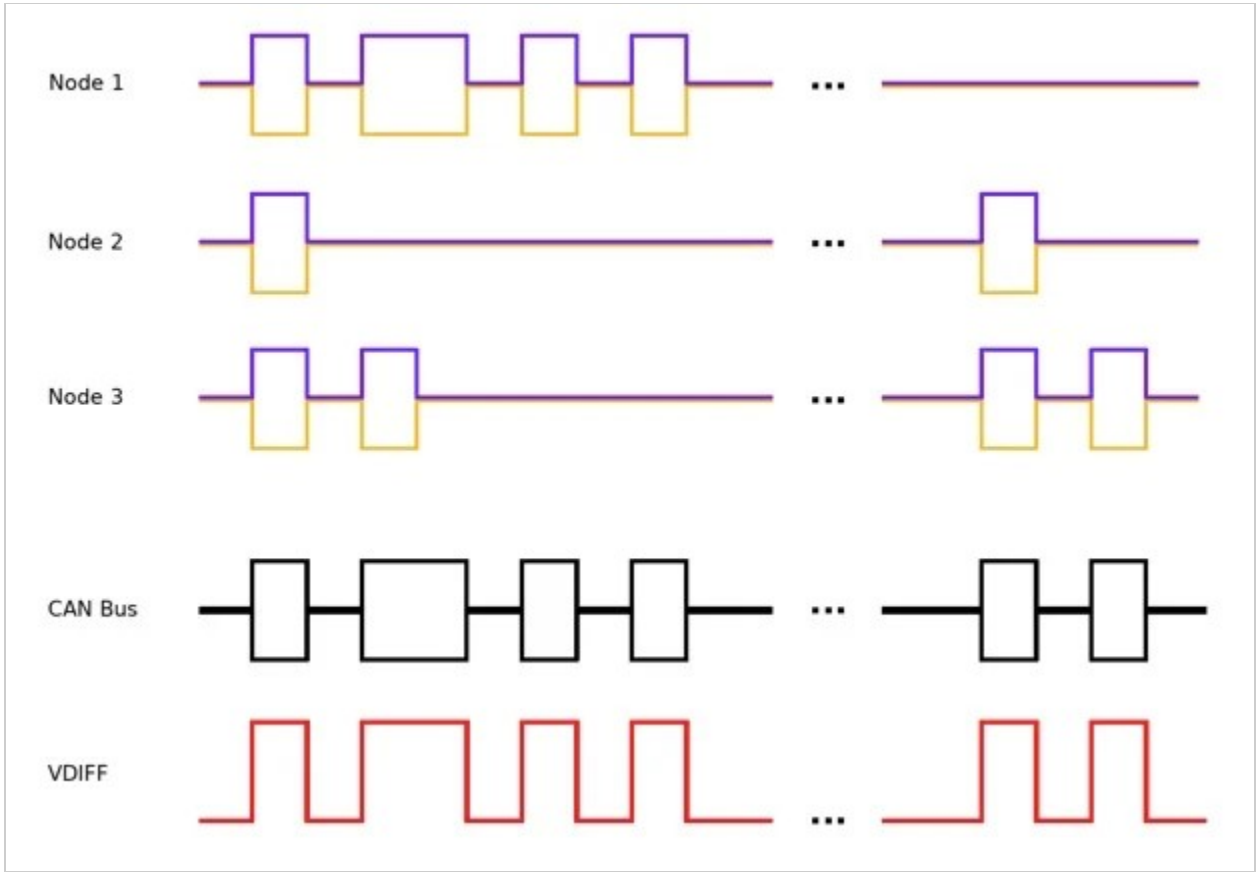

Figure 7. CAN bus arbitration with differential voltage

Conclusion

This article introduced the Controller Area Network or CAN. CAN is a robust serial communication bus found mostly in automotive and industrial environments. CAN uses a differential signal, which makes it more resistant to noise, along with a priority arbitration scheme for non-destructive message transmission. CAN is great for embedded applications that end up in hazardous environments or areas with a lot of electromagnetic interference. Whether you're building a remote-controlled submarine, setting up a microbrewery with pumps and sensors, or just hacking your car's computer, CAN is a great way to further your embedded knowledge while beefing up your next design project.

Load more comments