The open-source Sigrok project is a set of drivers and tools. It provides a
desktop oscilloscope and logic analyzer UI (user interface) that can control
different instruments (from Siglent, Rigol, and others).

The UI runs on Mac OS, Windows and Linux and is
called Pulseview. Integrated is also a command-line
tool for batch decoding, useful in an automated test envi-
ronment. Pulseview has an API (application programming
interface) for protocol decoders. Recently, a decoder for
CAN has been introduced.

Figure 1 is a screenshot of the decoder showing a CAN
frame. Here, the Pulseview interface runs on Ubuntu Linux.
The logic analyzer hardware used here is a 16-channel
Saleae Logic16. But the available USB logic analyzers
that cost less than 10 US $ with eight channels and a
sample rate of up to 20 MHz are also suitable for use
with CAN. A falling-edge trigger condition is typically
used with CAN (this is the sync point for the protocol). A
pre-trigger buffer enables the decoder to see at least ten
recessive bits to know that the next dominant bit is a new
frame.

The decoder shows four lines of details about a CAN
frame:
◆ The raw bitstream (including stuff bits)
◆ The decoded CAN fields
◆ The decoded CAN-ID and payload bytes
◆ An information line showing protocol events and
  warnings

View of details

Pulseview shows as much details as fits into an item for a
given time scale, but a tooltip appears with the full data if
the mouse pointer hovers over an item. For example, the
value of the 4-bit DLC (data length code) field with a tooltip
is shown in Figure 2.

The decoder also checks the frame for valid fields and
marks when an error is detected. For example, it will show
in the warning line when the received CRC (cyclic redun-
dancy check) does not match with the calculated CRC, when
the ACK (acknowledge) field is not 0, when a stuff error has
been detected, and so on. It also shows an active error frame
including the superposition, the error delimiter, and the IFS
(interframe space) field following an error frame.

The warning as shown in Figure 3 is a stuff error – the
result of an error being signaled by another CAN controller.
The decoder shows the error flag (which includes the super-
position of dominant bits from many controllers) and the error
delimiter. The trace also shows the frame being re-transmit-
ted successfully.

The double-receive event is a particularly interesting
property of CAN. Because a frame is received one bit-time
before it is transmitted, it is possible that an error in the last
bit of the EOF (end of frame) causes the transmitter to detect
an error and retransmit the frame, leading to it being received
twice. This is not a bug in the CAN protocol: it is an inevita-
ble consequence of implementing an atomic broadcast pro-
tocol (something that most other communication protocols do
not even attempt to provide, which is one reason why CAN is
such a superbly reliable fieldbus protocol).

As seen in Figure 4, the decoder warns of this specific
event (double receive). This event should happen rarely
(a bit error must occur exactly at the last bit of EOF). But
it can be engineered to occur by an attack on the bus: by
deliberately injecting a dominant bit at the last bit of EOF,
an attacker can force the frame to be retransmitted and
received twice. If the frame being targeted contains an event data, then that event will be acted upon twice by receivers, which could cause all kinds of things to go wrong – the very purpose of a malicious attack.

The decoder also shows when there is an overload frame – something that should never be seen since modern CAN controllers never generate these frames. The screenshot in Figure 5 shows a frame that is sent, but then a sequence of overload frames is injected to hold all the CAN controllers in an overload loop. This is a clear indication of a type of denial-of-service attack on the CAN network. In this case it was carried out by the CANhack toolkit. The CANhack toolkit is an open-source library for demonstrating attacks on the CAN protocol: [github.com/kentindell/canhack](https://github.com/kentindell/canhack).

The timestamps attached to the CAN frames in the packet stream are very accurate. These are useful when hunting for a particular incident in the frame view of the Wireshark (or other tools) to navigate within Pulseview to find details of what was happening on the wire around an incident.

Unveiling hidden problems

The decoder has already helped one developer to solve a problem with their system. Ian Tabor (@mintynet on Twitter) is a car hacker who has developed a low-cost “car in a box” system for people to practice hacking. The hardware includes the ability to send CAN frames from three different buses to a monitoring bus via an MCP2515 CAN controller from Microchip. The driver was running very slowly and unable to sustain throughput that was needed. But Pulseview was able to show where the time was going: the CAN protocol decoder showed the CAN frames and the Pulseview SPI decoder (serial peripheral interface) showed where the SPI transactions were taking place. The screenshot in Figure 8 shows the situation before.

The screenshot in Figure 9 shows the situation after the drivers were optimized, reducing the time between frames from nearly 500 µs to just over 300 µs.
The tool’s availability

The source code to the CAN protocol decoder is available in the CANhack toolkit repository on Github at github.com/kentindell/canhack. The decoder is in the folder src/can2. The best way to install the Sigrok tools (Pulseview and Sigrok-cli) is to download them directly from Sigrok. More details on setting up the decoder can be found here.

CAN board for Raspberry Pi Pico

The CANPico board has been recently released by Canis Automotive Labs. It integrates a Microchip MCP2517/18FD CAN controller with a 2-KiB buffer and the Microchip MCP2562FD CAN transceiver. Jumpers are available for connection of a 120-Ω CAN termination resistor and for disabling of transmit access to the CAN network (listen-only access). There is also a 6-pin header for connection of a logic analyzer (e.g. the CAN2 protocol decoder) or oscilloscope. The included Trig pin can be set in order to trigger the logic analyzer on a specific CAN-ID or a CAN error frame.

Along with the board is a pre-built Micropython SDK (start development kit) firmware, with a CAN API that includes priority-inversion-free drivers, time-stamping (both send and receive), control of CAN-ID filters, and a CAN bit-rate setup. The board is ready for order online from SK Pang. It is shipped with a Raspberry Pi Pico and the pre-installed firmware. More detailed information is available here.

CiA seminars online

<table>
<thead>
<tr>
<th>Date</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN</td>
<td>2021-07-07</td>
</tr>
<tr>
<td>CANopen</td>
<td>2021-07-08</td>
</tr>
<tr>
<td>CAN</td>
<td>2021-09-14</td>
</tr>
<tr>
<td>CANopen</td>
<td>2021-09-15</td>
</tr>
</tbody>
</table>

CiA seminars

Having the Covid-19 pandemic in mind, hopefully we are able to welcome you to our on-site seminars at CiA office.

<table>
<thead>
<tr>
<th>Date</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN for Newcomers</td>
<td>2021-10-05</td>
</tr>
<tr>
<td>CANopen for Newcomers</td>
<td>2021-10-06</td>
</tr>
</tbody>
</table>

CiA in-house seminars online

CiA engineers discuss your urgent CAN-related issues that are currently of high interest with regard to your projects.

For more details please contact CiA office at events@can-cia.org

www.can-cia.org