Software-in-the-Loop Using Virtual CAN Buses: Current Solutions and Challenges

Thomas Liebezeit, Andreas Junghanns, Mirco Bonin, Roland Serway

1. **Motivation**

Increasing pressure to save time and cut costs in the development and testing of more and more complex automotive systems requires improved methods for development and testing. Software-in-the-Loop is one such method Tm p-191(sea)-3(n)-3(d-3(

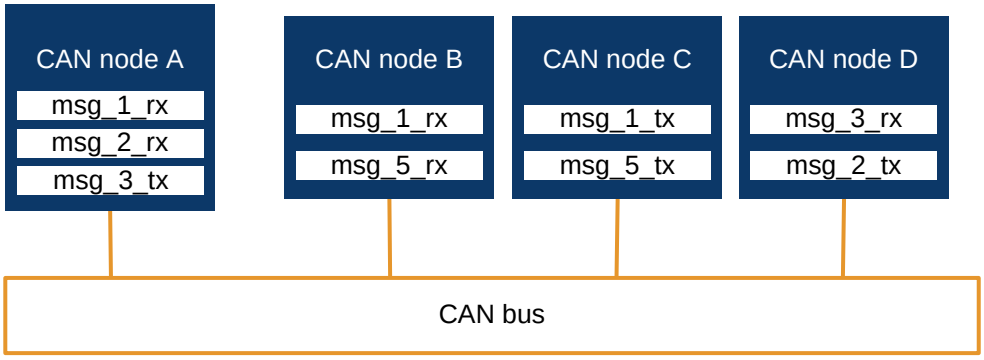

Figure 1: CAN bus

CAN communication is affected by slight variations in transmission delay or even lost messages. Higher-level protocol layers may implement measures to deal with such effects in the transport layer. Messages may contain CRCs and message counter signals in order to ensure message and protocol integrity.

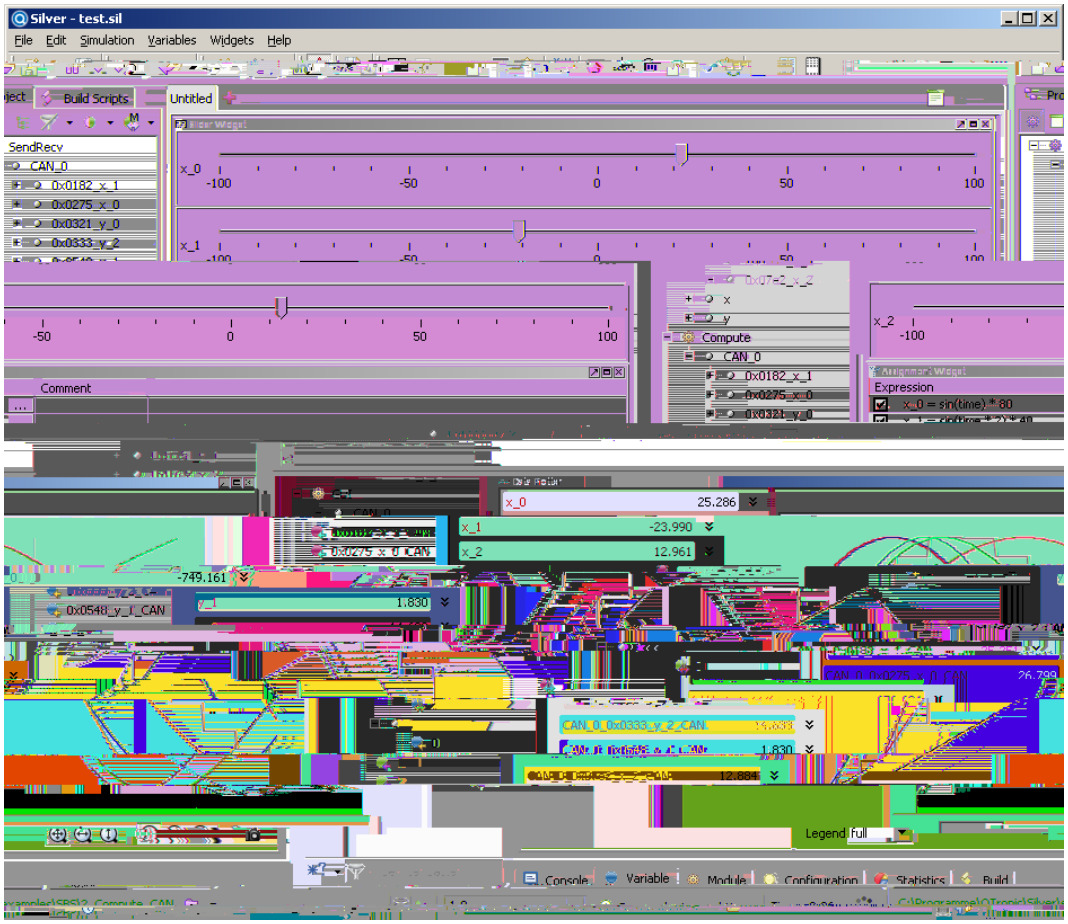

Figure 2: Silver GUI with example CAN

To improve or even remove problem 1) and 2), it is possible to implement modules that delay and lose CAN messages when protocol robustness analyses a

- [illegible] Receive a physical signal value from the corresponding message as received previously
- [illegible] Send a physical signal value with the next corresponding message
- [illegible] Equivalent to `SBS_CAN_GetSigValue`, just with a raw signal value
- [illegible] Equivalent to [illegible] `lue`, just with a raw signal value

For handling signals and messages, raw and physical values, sending cyclic messages and messages on demand, SBS needs a CAN specification in form of a DBC file:

- [illegible]: Configures the CAN communication with
  - the CAN bus ID for distinguishing multiple CAN buses in Silver,
  - the DBC file defining the current network,
  - the node name that this Silver module enacts (possibly a number of node names if this Silver module simulates multiple CAN nodes, such as a rest bus simulation),
  - a number of flags to influence the behavior of the CAN functions, and
  - optionally the name of a file specifying a list of messages (not) to be sent and received (by message ID or message name)
- `SBS_CONF_AddCANMsg`: In case a DBC file is not available, cannot be published or tool support for DBCs is missing, SBS offers a function to add (and configure) one message at a time. Signal extraction is not supported in this case, only 8-byte messages can be received and sent.
- `SBS_CONF_SetCallbackDLL`: When changing the content of a message by inserting signal values, the project-specific CRC for this message will be invalid. Moreover, some messages require message counters to be incremented in a project-

3.3 The Simulink Silver CAN block set

3.4 DBC files to specify the behavior of a CAN node

Using DBC files to specify the behavior of a node in a (virtual) CAN network is convenient, reduces the chances of errors and permits fast introduction of updates, as DBC files are usually well maintained and tested by many engineers. The use of DBC files this early in the development process also helps to test and debug these DBC files.
Silver permits building modules with SBS where the DBC files can be specified at runtime, allowing the end user to incorporate newer versions of the DBC if necessary without rebuilding the module itself.
Changing the DBC file after coding may cause critical problems at runtime, because of the signal names used in the C code. If, for instance, a signal was renamed or removed from a message in the DBC file, the C code will still query its value and create a runtime error. A safe change is a change in how a signal is coded inside a message, as those details are hidden from the C code.

3.5 Accessing physical CAN networks from Silver

Silver offers a convenient way of accessing physical CAN networks (Figure 4). A specialized Silver CAN module lets Silver access a CAN network device and communicate through it with all the other devices on that physical CAN bus.
At the moment, this solution supports accessing Vector CAN cards and CAN USB. A hardware abstraction layer allows easy addition of other CAN hardware in the future.
The Silver CAN module is configured using a DBC file and the network node name Silver is enacted. The CAN module automatically generates its own input and output

4. Example

IAV uses the newly developed CAN support to set up an improved SiL simulation of a transmission control software in mass production.

4.1. General setup

The setup consists in principle of a virtual ECU with control software and an environment model. The following diagram sketches this setup. For a detailed description see [3].

Figure 5: SiL setup for transmission control software

Data exchange between those main simulation components uses exactly the same interface definitions as in the car (CAN with more than 30 relevant messages, less than 20 input ports and less than 20 output ports). Thus the original car CAN definition (a DBC file) is used and the environment model serves as rest bus

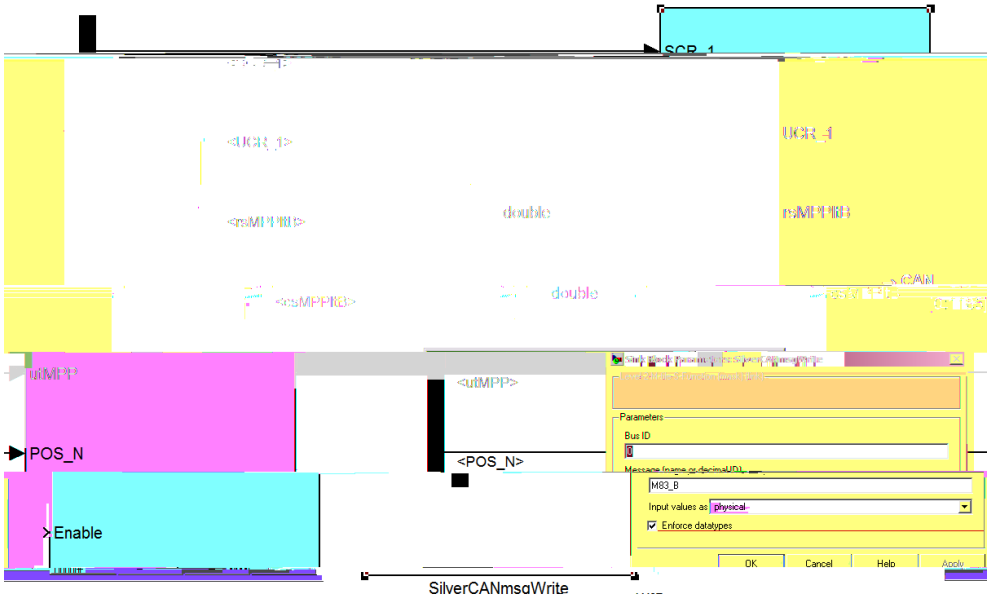

Figure 6: Silver CAN block in Simulink

The DBC file defines a large number of messages being sent on the CAN network, many of which are not part of the required CAN rest bus (Figure 7). A white list of CAN messages (by ID) is used to reduce this list of the DBC-defined messages to the CAN messages actually used in the SiL. A black-list approach is also possible but is less convenient here because of the number of signals to be excluded.
The SiL setup uses a Silver CAN module for mirroring the SiL-internal CAN bus to a Vector virtual CAN bus. This enables reading of all CAN messages by CANape in addioa7-130(m)-3(g) A-133(rexa)-4(l)-40(m)-3(g) A-U7 .1s644(l)(g) AaoC s5.7 7d<</MCID 5/Lar

a

4.3. Example summary

The described solution drastically simplifies the setup of the SiL system compared to previous, non-SBS-based setups. It reduces the amount of work for SiL signal mapping by using existing standard CAN definitions.
It makes the simulation even more realistic and lets users compare signal sequences directly with in-