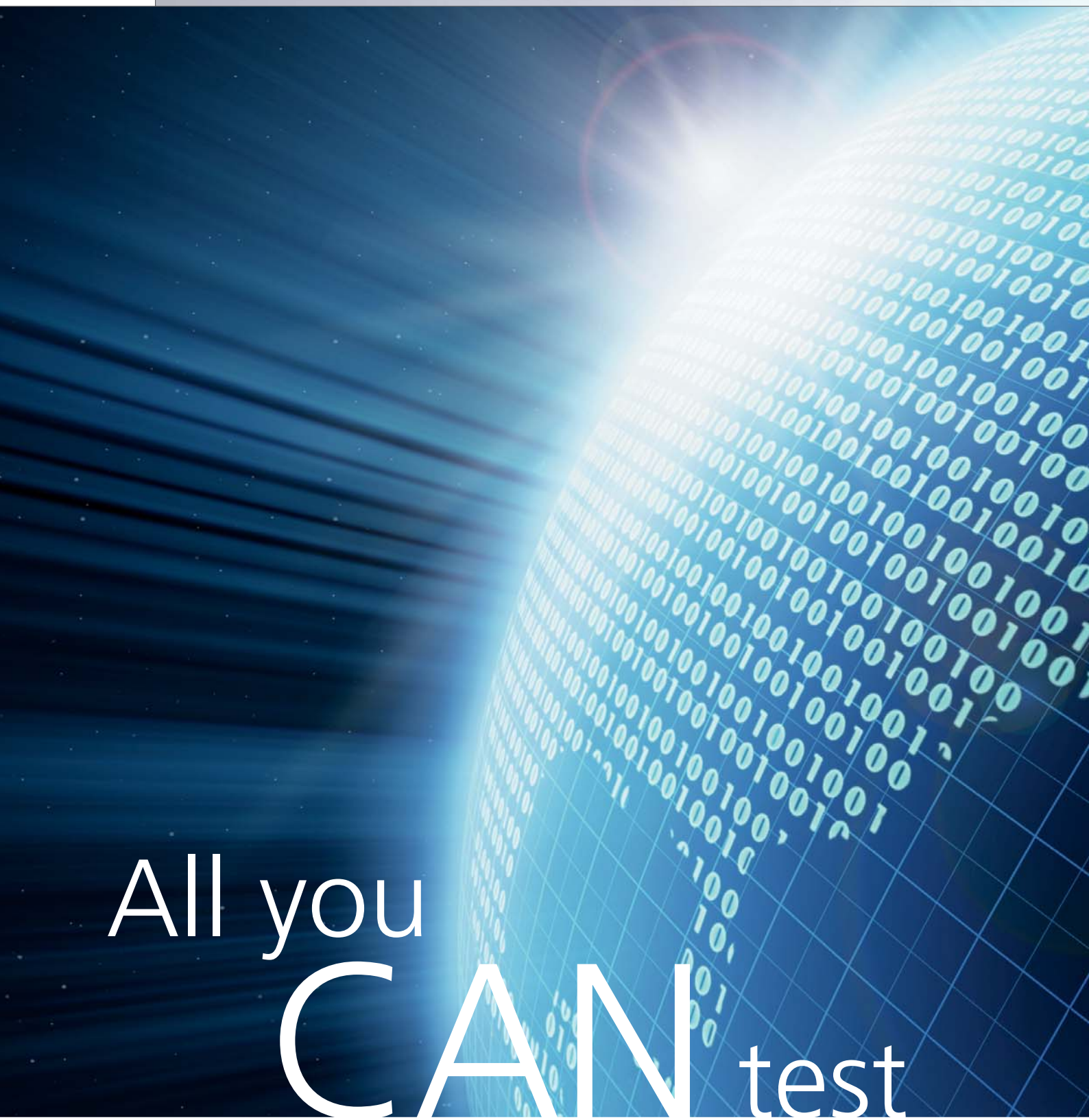

All you CAN test

From CAN Set-up to Real-Time Test

The RTI CAN MultiMessage Blockset from dSPACE is a proven tool for representing large CAN set-ups flexibly and conveniently. It can now be integrated into ControlDesk via the CAN Navigator and also supports real-time testing, allowing users to switch seamlessly between developing and testing. This article describes a typical multi-tool workflow.

Automotive applications in the fields of rapid control prototyping (RCP) and hardware-in-the-loop (HIL) simulation typically use a CAN bus. Depending on the application area, either the controller or the controlled system is represented by a real-time model which also includes the CAN communication. This bus is configured with the RTI CAN MultiMessage Blockset using a data basis (such as a DBC file) that defines CAN bus signals and messages. The blockset provides a graphical user interface for Simulink® (fig. 1), from which users can select the necessary Rx (receive) and Tx (transmit) messages. This procedure provides a basic configuration of the CAN bus modeled in the real-time model. Variant handling, transmission control and various signal manipulations can also be configured. This configuration is particularly relevant to HIL applications, as it generates (within the real-time application) the dynamic intervention points to be used later for

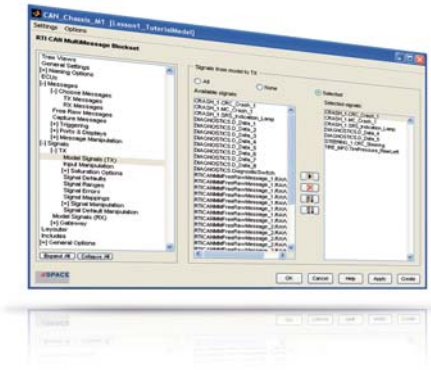

Fig. 1: Selecting model signals for transmission via the RTI CAN MultiMessage Blockset.

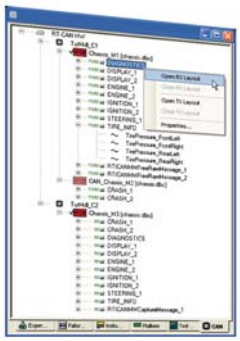

Fig. 2: The CAN Navigator is the central access point to handle CAN in ControlDesk.

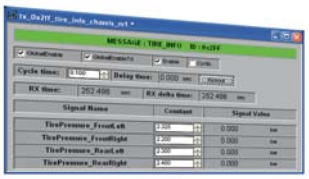

Fig. 3: Tx layouts can be generated directly from the CAN Navigator.

testing and automating the HIL simulator's CAN communication. CAN monitoring in ControlDesk and CAN support for real-time testing are also prepared within the blockset.

CAN Navigator Is the Control Center

In ControlDesk, the CAN Navigator's tree is the central access point for handling CAN. The process of generating code for the real-time model supplies all the data needed for producing the tree: the user just has to insert a reference to the application in ControlDesk and the tree fills automatically (fig. 2). The tree then displays a consistent configuration of the CAN communication as defined in the model and configured with the RTI CAN MultiMessage Blockset.

The tree visualizes all the CAN controllers contained in the model with their assigned DBC files, and the CAN messages with their signals. During run time, it can be used to switch between variants of the CAN controller's DBC configuration. In addition to generating these layouts from the CAN MultiMessage Blockset in advance with Python, as was previously the case, users can now generate the layouts from the tree whenever they are needed – no Simulink installation is necessary (fig. 3). The layouts directly mirror the send and receive configurations of the messages and signals in the real-time model. Typical examples of such generated layout elements are input fields for setting the cycle and time buttons for transmitting messages sporadically.

Analyzing Communication

Global layouts can be generated to handle the transmission control of multiple messages simultaneously. In the CAN Navigator, it is also possible to create layouts for the online configuration of CAN gateways – this is not possible in the blockset. There is a CAN monitoring window with various views (fig. 4, fig. 5) and sort options for comprehensive analysis of communication behavior. Monitoring can either be based on raw data or be performed symbolically with the current DBC reference.

The messages selected for monitoring can be restricted via freely definable and savable filter rules, but it is also possible to visualize the entire CAN traffic. Pass and stop filters can be applied selectively to IDs, ID ranges and ECUs and can also be combined. The messages visualized in the CAN monitoring window can also be saved to a file (*.csv or *.asc). This file can be used as the starting point for precisely-timed CAN replays of the CAN communication. For example, communication data recorded during real test drives can easily be reproduced in a restbus simulation on the HIL simulator as often as required.

Testing under Real-Time Conditions

For more complex test scenarios, there is Real-Time Testing (RTT) for the Python-based development of real-time test scenarios. In this, the tests run time-synchronously to the real-time model, and read and write access can be performed to all model variables in each simulation clock cycle.

If the real-time tests have to access the CAN bus, they are developed via a special library (canmmlib) for reading and writing raw-data-based messages. The necessary messages are added to the model by means of the RTI CAN MultiMessage Blockset and are then available in transparent form for real-time tests or for CAN replays of the CAN Navigator. The advantage is that changes to the CAN database or the model structure do not affect the executability of the real-time tests.

An example test scenario is the monitoring of one of the HIL simulator's analog input signals. When a defined trigger threshold (for example, 14.7 V) is exceeded, a

Fig. 4: Static monitor view with the decoded contents of CAN message TIRE_INFO.

Fig. 5: Continuous monitor view (here sorted according to the time received).

predefined CAN message has to be sent cyclically every 50 ms until the value goes below the threshold again (fig. 6). The associated real-time test implementation is explained in the text inset.

Multi-Tool CAN Handling

The versatile configuration options provided by the RTI CAN MultiMessage Blockset ensure that CAN communication is configured consistently across multiple tools. It covers all the application scenarios that are typical of ECU testing, such as interactive experimentation, CAN monitoring and replay, and automated restbus simulation. Thus, dSPACE provides a complete, flexible, convenient solution for editing CAN communication. Such solutions will be available for other bus systems in the near future. ■

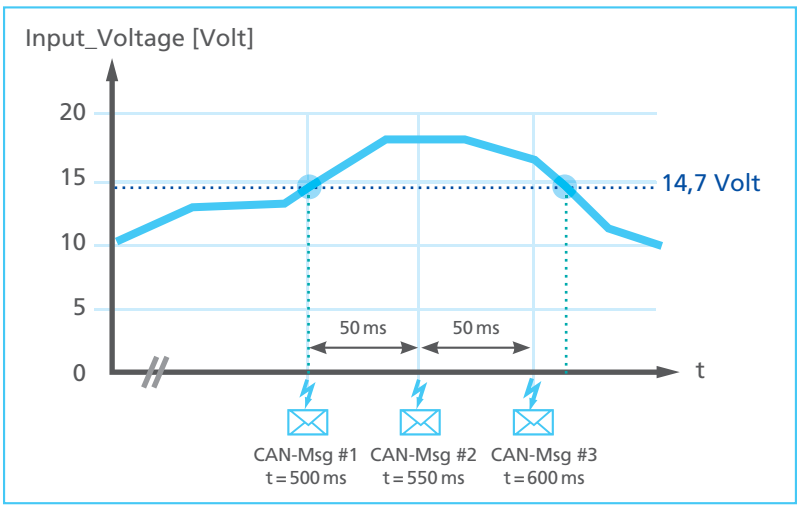

Fig. 6: Example scenario: Monitoring a model signal and its reaction with triggered, cyclic CAN communication.

Real-Time Test Implementation of the Example Test Scenario

In the real-time test, the send data of the CAN message can be defined freely via Python programming. This means that real-time tests can include test-case-specific CAN behavior that is not intended to be a permanent part of the real-time model. Real-time testing is therefore an ideal supplement to the static (model-defined) CAN configurations of the RTI CAN MultiMessage Blockset. The ability to load several real-time tests independently of one another and execute them simultaneously ensures maximum scalability in implementing restbus simulation. The bandwidth reaches from simple, reactive real-time observers (as in the example) to extensive test sequences that are executed completely on the real-time hardware.

```
1.  # -*- coding: ascii -*-
    from rttlib.canlib.controllers import canmmlib
    from rttlib import utilities, variable
    CurrentTime = utilities.currentTime

2.  # Voltage to be observed
    Input_Voltage = variable.Variable(r'Model
    Root/Input_Voltage/Value')

3.  # Initialize CANMM controller
    Controller              = None
    ControllerTRCPathName   = r"BusSystems/CAN/Chassis_M1"
    Controller              = canmmlib.GetController
                              ControllerTRCPathName)
    Channel                 = Controller.GetChannel()

4.  # Initialize the response CAN message
    Message = Channel.GetRawMessage()
    Message.Format  = canmmlib.canmmbaselib.ftSTD
    Message.ID      = 0x123
    Message.DLC     = 8
    Message.TX.Data = 0x10203040506070B0

5.  # Wait helper function
    def WaitGen(Duration):
        EndTime = CurrentTime.Value + Duration
        while(EndTime > CurrentTime.Value):
            yield None

6.  def MainGenerator():
        # Observe Input Voltage continously
        while(1):
            if(Input_Voltage.Value > 14.7):
                Message.Transmit()
                yield WaitGen(0.05)
            yield None
```

1. *Import real-time testing libraries (canmmlib, etc.)*
2. *Generate the variable object of the voltage to be monitored*
3. *Select the CAN send controller*
4. *Define the Send message with the real-time test*
5. *Help function for precise time measurement (for later transmission of CAN messages at exact time intervals)*
6. *Real-time test sequence: Check the voltage once per simulation step and react by sending a cyclic CAN message*