All you CAN test

Vom CAN-Setup zum Echtzeittest

Das RTI CAN MultiMessage Blockset von dSPACE ist ein bewährtes Werkzeug, um große CAN-Setups flexibel und komfortabel abzubilden. Durch die seit Kurzem verfügbare Integration in ControlDesk über den CAN Navigator sowie das Echtzeittesten hat der Anwender nun die Möglichkeit, durchgängig zu entwickeln und zu testen. Dieser Artikel beschreibt einen typischen toolübergreifenden Arbeitsablauf.

Automotive Anwendungen aus den Bereichen Rapid Control Prototyping (RCP) oder Hardware-in-the-Loop (HIL)-Simulation sind typischerweise mit einem CAN-Bus ausgestattet. Je nach Anwendungsbereich liegt entweder der Regler oder die Regelstrecke als Echtzeitmodell vor, in dem die CAN-Kommunikation abgebildet ist. Die Konfiguration dieses Busses erfolgt mit dem RTI CAN MultiMessage Blockset unter Verwendung einer Datenbasis (zum Beispiel einer DBC-Datei), in der die Signale und Botschaften eines CAN-Busses definiert sind. Das Blockset bietet eine grafische Oberfläche für Simulink® (Bild 1), um die benötigten Rx (Receive)- und Tx (Transmit)-Botschaften auszuwählen. Damit steht eine Basiskonfiguration des modellierten CAN-Busses im Echtzeitmodell zur Verfügung. Auch das Variantenhandling, die Versendesteuerung und diverse Signalmanipulationen sind konfigurierbar. Diese Konfiguration ist insbesondere für den HIL-Einsatz relevant und erzeugt in der

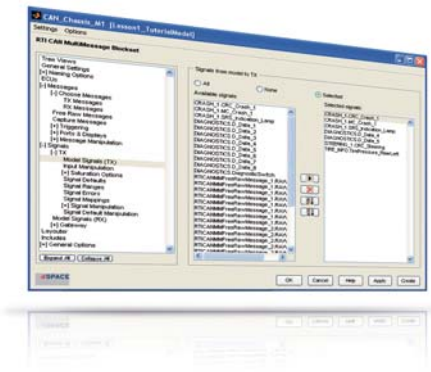

Bild 1: Auswahl von Modellsignalen für den Versand über das RTI CAN MultiMessage Blockset.

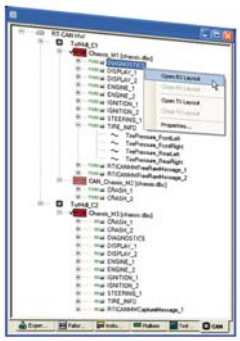

Bild 2: CAN Navigator als zentraler Zugang für das CAN-Handling in ControlDesk.

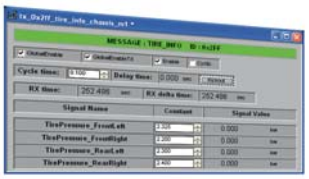

Bild 3: Tx-Layouts können direkt aus dem CAN Navigator erzeugt werden.

Echtzeitapplikation die dynamischen Eingriffspunkte für das spätere Testen und Automatisieren der CAN-Kommunikation des HIL-Simulators. Des Weiteren werden in dem Blockset das CAN-Monitoring in ControlDesk sowie die CAN-Unterstützung für das Echtzeittesten vorbereitet.

CAN Navigator als Steuerzentrale

Unter ControlDesk ist der CAN Navigator mit seiner Baumdarstellung der zentrale Zugang für das CAN-Handling. Die Code-Generierung des Echtzeitmodells liefert hierbei alle notwendigen Daten für die Erzeugung des Baumes: Ein Verweis des Anwenders auf die entsprechende Applikation in ControlDesk genügt und der Baum füllt sich automatisch (Bild 2). Der Baum zeigt anschließend eine konsistente Konfiguration der im Modell vorhandenen und mit dem RTI CAN MultiMessage Blockset konfigurierten CAN-Kommunikation.

Durch die Baumstruktur sind alle im Modell enthaltenen CAN-Controller mit den zugeordneten DBC-Dateien sowie den CAN-Botschaften mit ihren Signalen ersichtlich. Hier lässt sich während der Laufzeit zwischen Varianten der DBC-Konfiguration des CAN-Controllers direkt umschalten. Zusätzlich zu der bisherigen Python-Generierung auf Vorhalt aus dem CAN MultiMessage Blockset können diese Layouts nun auch erst bei Bedarf aus der Baumstruktur generiert werden – eine Simulink-Installation ist hierfür nicht erforderlich (Bild 3). In den Layouts spiegelt sich die Sende- und Empfangskonfiguration der Botschaften und Signale des Echtzeitmodells direkt wider. Typische Beispiele für solche generierten Layoutelemente sind Eingabefelder zur Einstellung der Zykluszeit oder auch Schaltflächen zum sporadischen Versenden von Botschaften.

Analyse der Kommunikation

Um die Versendesteuerung auch für mehrere Botschaften gleichzeitig zu handhaben, lassen sich globale Layouts generieren. Außerdem können im CAN Navigator Layouts zur Online-Konfiguration von CAN-Gateways erstellt werden – dies ist im Blockset nicht möglich. Für eine umfassende Analyse des Kommunikationsverhaltens steht ein CAN-Monitoring-Fenster mit verschiedenen Ansichten (Bild 4, Bild 5) und Sortiermöglichkeiten zur Verfügung. Das Monitoring kann hierbei entweder rohdatenbasiert oder symbolisch mit dem aktuellen DBC-Bezug erfolgen.

Die für das Monitoring ausgewählten Botschaften lassen sich über frei definier- und speicherbare Filterregeln eingrenzen, genauso gut kann aber auch der gesamte CAN-Verkehr visualisiert werden. Durchlass- oder Sperrfilter sind selektiv auf IDs, ID-Bereiche und Steuergeräte (auch kombiniert) anwendbar. Zudem ist es möglich, die im CAN-Monitoring dargestellten Botschaften in einer Datei abzuspeichern. Diese Datei (*.csv oder *.asc) kann als Ausgangsbasis für ein CAN-Replay dienen, das die CAN-Kommunikation zeitgenau wiedergibt. Damit lassen sich zum Beispiel sehr einfach die in realen Fahrversuchen aufgezeichneten Kommunikationsdaten in einer Restbussimulation am HIL-Simulator beliebig oft reproduzieren.

Testen unter Echtzeitbedingungen

Für komplexere Testszenarien steht Real-Time Testing (RTT) zur Python-basierten Entwicklung von Echtzeittestszenarien zur Verfügung. Hierbei verlaufen die Tests zeitsynchron mit dem Echtzeitmodell, wodurch Schreib- und Lesezugriffe in jedem Simulationstakt auf allen Modellvariablen möglich sind.

Sollen die Echtzeittests auf den CAN-Bus zugreifen, erfolgt ihre Entwicklung über eine spezielle Bibliothek (canmmlib), mit der rohdatenbasiert Botschaften gelesen und geschrieben werden können. Mit Hilfe des RTI CAN MultiMessage Blocksets können hierfür im Modell entsprechende Botschaften vorgesehen werden, die dann transparent für Echtzeittests bzw. für das

Bild 4: Statische Monitoransicht mit decodiertem Inhalt der CAN-Nachricht TIRE_INFO.

Bild 5: Fortlaufende Monitoransicht (hier nach Empfangszeitpunkt sortiert).

CAN-Replay des CAN Navigators zur Verfügung stehen. Vorteilhaft ist hierbei, dass Änderungen an der CAN-Datenbasis bzw. der Modellstruktur keinen Einfluss auf die Ausführbarkeit der Echtzeittests haben.

Ein beispielhaftes Testszenario ist die Überwachung eines analogen Eingangssignals des HIL-Simulators. Hierbei soll bei Überschreitung einer definierten Triggerschwelle (zum Beispiel 14,7 Volt) eine vordefinierte CAN-Botschaft zyklisch alle 50 ms so lange gesendet werden, bis die Schwelle wieder unterschritten wird (Bild 6). Die zugehörige Echtzeittestimplementierung wird im Textkasten erklärt.

Toolübergreifendes CAN-Handling

Mit seinen vielfältigen Konfigurationsmöglichkeiten erlaubt das RTI CAN MultiMessage Blockset eine konsistente und toolübergreifende Bearbeitung der CAN-Kommunikation. Es deckt alle typischen Anwendungsszenarien für den Steuergerätetest ab, zum Beispiel interaktives Experimentieren, CAN-Monitoring und -Replay sowie automatisierte Restbussimulation. dSPACE bietet damit eine vollständige, flexible und komfortable Lösung zur Bearbeitung der CAN-Kommunikation, die in diesem Umfang zukünftig auch für weitere Bussysteme angeboten werden soll. ■

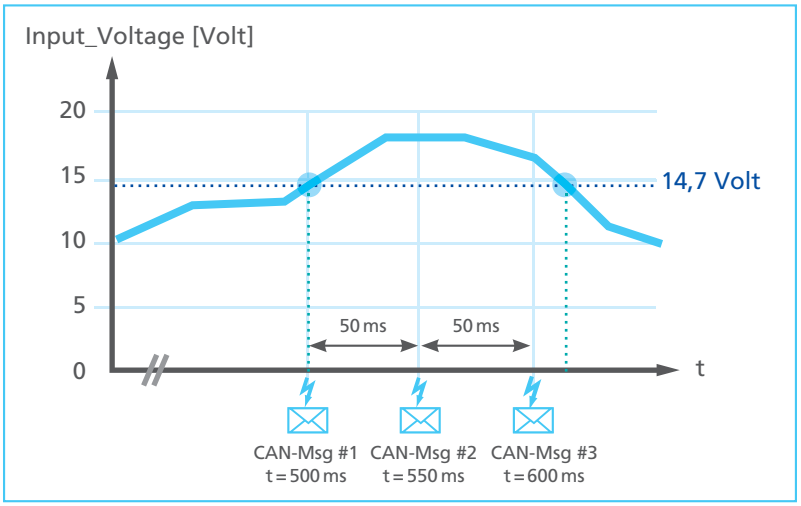

Bild 6: Beispielszenario – Überwachung eines Modellsignals und Reaktion mit getriggerter zyklischer CAN-Kommunikation.

Echtzeittestimplementierung des Beispieltestszenarios

Die Versendedaten der CAN-Botschaft können im Echtzeittest über freie Python-Programmierung festgelegt werden. Durch Echtzeittesten lässt sich dadurch vorzugsweise das testfallspezifische CAN-Verhalten abbilden, das nicht permanenter Bestandteil des Echtzeitmodells sein soll. Das Echtzeittesten ist somit eine optimale Ergänzung zu den statischen, d. h. im Modell festgelegten CAN-Konfigurationen des RTI CAN MultiMessage Blocksets. Durch die Möglichkeit, mehrere Echtzeittests unabhängig voneinander zu laden und gleichzeitig abzuarbeiten, ergibt sich eine maximale Skalierbarkeit bei der Implementierung der Restbussimulation. Die Bandbreite reicht von einfachen, reaktiven Echtzeitbeobachtern (wie im gezeigten Beispiel) bis hin zu umfangreichen, vollständig auf der Echtzeit-Hardware ausgeführten Testsequenzen.

```
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 = 0x10203040506070 80

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 der Echtzeittestbibliotheken (u. a. canmmlib)*
2. *Erzeugung des Variablenobjekts der zu überwachenden Spannung*
3. *Auswahl der Versendebotschaft durch den Echtzeittest*
4. *Definition der Versendebotschaft durch den Echtzeittest*
5. *Hilfsfunktion für exakte Zeitvermessung (für das spätere Versenden der CAN-Botschaften mit exakten Zeitabständen)*
6. *Echtzeittestablauf: Überprüfung der Spannung einmal pro Simulationsabschnitt und Reaktion mit zyklischem CAN-Botschaftenversand*