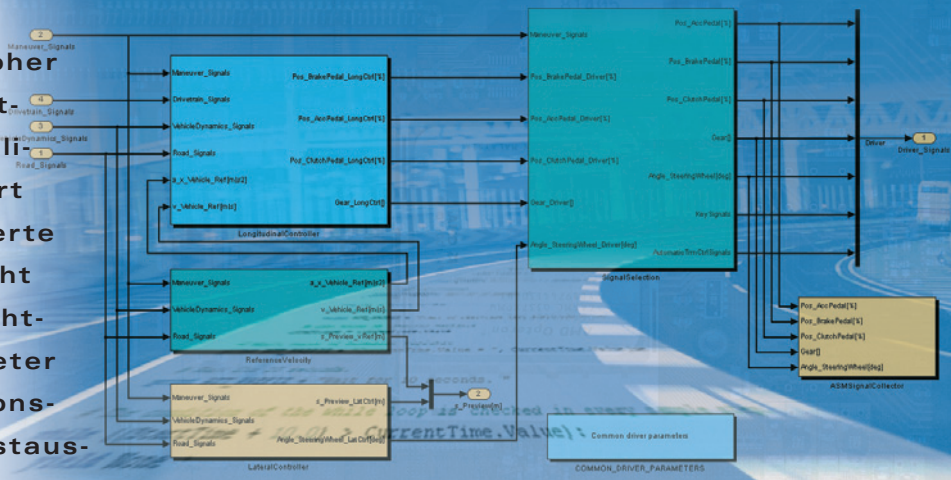

Sollen am HIL-Simulator Steuergerätetests mit hoher Reaktivität und einer zeitlichen Genauigkeit im Millisekundenbereich realisiert werden, ist eine PC-basierte Testausführung allein nicht mehr ausreichend. Ein echtzeitfähiger Skriptinterpreter erlaubt die zum Simulationsmodell taktsynchrone Testausführung auf der HIL-Prozessorkarte. Damit sind zeitgenaue und reproduzierbare Echtzeittests möglich, die in der objektorientierten Programmiersprache Python komfortabel entwickelt werden können.

TAKTSYNCHRONE AUSFÜHRUNG VON HIL-SIMULATIONSMODELLEN UND TESTSKRIPTEN

Automatisierter Echtzeittest von Steuergeräten

Im Automobilbereich hat sich bei nahezu allen OEMs und Zulieferern die Hardware-in-the-Loop-Simulation (kurz: HIL-Simulation) als zwingender Absicherungsschritt in der Elektronikentwicklung durchgesetzt. Hierbei werden die zu testenden Steuergeräte direkt an einen leistungsfähigen HIL-Simulator angeschlossen. Dieser kann über Echtzeitrechnung von Simulationsmodellen z. B. das Motor-, Getriebe- und Fahrdynamikverhalten des realen Fahrzeuges detailgetreu nachbilden. Der Simulator bietet hierbei über seine Hardwareschnittstellen (wie z. B. I/O-Kanäle, Fehlersimulationskarten, Businterfaces für CAN, LIN und FlexRay) umfangreiche Möglichkeiten zur gezielten Beeinflussung und Beobachtung einzelner Steuergeräte bzw. des zu testenden Steuergeräteverbundes.

Für eine effektive Ausnutzung des HIL-Simulators sind leistungsfähige Werkzeuge zur Testautomatisierung verfügbar (wie z. B. AutomationDesk von dSpace). Die Tests stimulieren hierbei das Steuergerät, beobachten dessen Reaktion und führen eine Ergebnisauswertung bzgl. des erwarteten Soll-Verhaltens durch. Auf den Simulatoren wird in der Regel rund um die Uhr und auch an Wochenenden getestet, so dass diese Tests vollautomatisiert und ohne Interaktion lauffähig sein müssen. Die Testschritte werden meistens auf einem Standard-PC abgearbeitet, der über ein Kommunikationsinterface auf den HIL-Simulator zugreift.

Synchrone Simulation und Testausführung auf dem Prozessorboard

Die Anforderungen an Steuergerätetests haben sich hinsichtlich Funktion, Reaktivität, zeitlicher Auflösung und

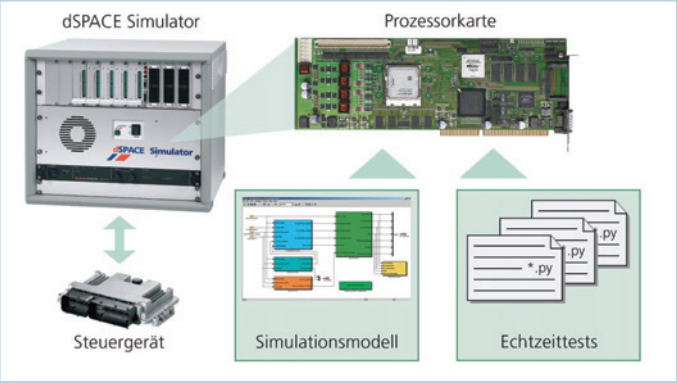

Bild 1: Mehrere Echtzeittests können taktsynchron mit dem Simulationsmodell auf der Prozessorkarte des Simulators ausgeführt werden.

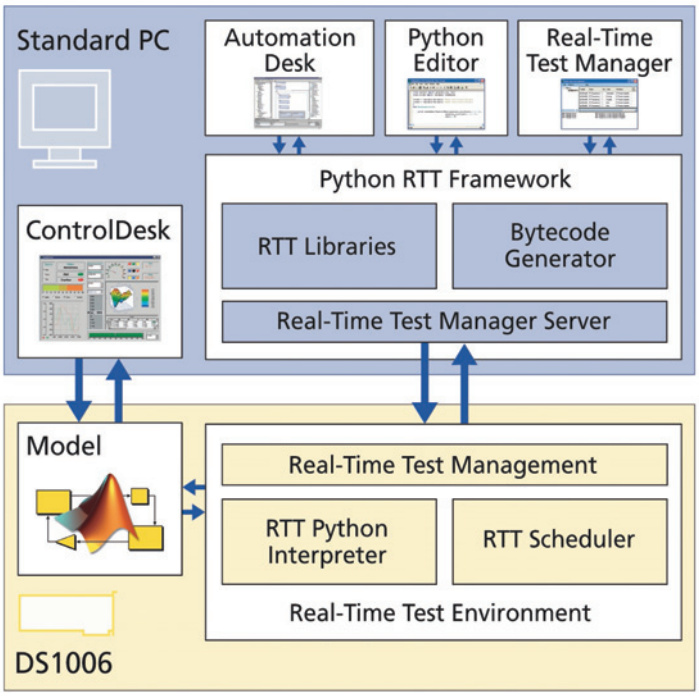

Bild 2: Komponenten des Echtzeittestens.

Reproduzierbarkeit zusammen mit der Komplexität der zu testenden Systeme deutlich erhöht. Beispiele hierfür sind unter anderem

- geforderte Testreaktivität im Millisekundenbereich
- zeitgenaue Stimulation von mehreren Modellsignalen mit komplexen Signalmustern
- präzise Zeitvermessung von Modelländerungen
- zuverlässige Ermittlung von Minimal- und Maximalwerten von Modellgrößen (gerade auch über längere Messzeiträume)
- Messdatenreduktion durch selektive, bedingungsabhängige Aufzeichnung
- dynamisches Aufsetzen einer auf das Steuergerät angepassten Restbussimulation
- parallele Ausführung mehrerer, unabhängiger Steuergerätetests

Eine rein PC-basierte Ausführung ist hierfür nicht immer ausreichend. AutomationDesk 1.4 bietet als neue Möglichkeit an, mehrere Steuergerätetests taktsynchron zum Simulationsmodell auszuführen. Die dadurch realisierbaren Echtzeittests können dynamisch während der laufenden HIL-Simulation auf die Prozessorkarte heruntergeladen und parallel hierzu ausgeführt werden (**Bild 1**). Der Datenaustausch zwischen Test und Simulation ist über den gemeinsamen Speicher des Prozessorboards realisiert, welches kurze Latenzzeiten garantiert. Die verschiedenen Tests können untereinander Daten austauschen, welches eine hohe Flexibilität bei der Testerstellung ermöglicht.

Echtzeittests in Python programmiert

Ein wichtiger Punkt für die Anwenderakzeptanz einer Testlösung ist die Art des angebotenen Nutzerzuganges. Bei der Entwicklung der Echtzeittestlösung stand eine einfach zu erlernende und zu nutzende, aber trotzdem leistungsfähige Testprogrammierung im Vordergrund. Aus diesem Grund werden Echtzeittests mithilfe der objektorientierten Skriptsprache Python (www.python.org) beschrieben. Die Sprache Python an sich bringt schon hochwertige Beschreibungsmittel mit (wie z. B. Importmechanismen, Klassendefinitionen und komplexe Datenstrukturen). Diese können bei der Programmierung von Echtzeittests ohne Einschränkungen verwendet werden.

Die Echtzeittests können in jedem einzelnen Simulationsschritt alle Größen des Simulationsmodells beobachten und beeinflussen. Für den Modellzugriff stehen spezielle, auf das Echtzeittesten zugeschnittene Real-Time Test Bibliotheken (kurz: RTT Libraries) zur Verfügung. Damit können Lese- und Schreibzugriffe auf Simulationsvariablen symbolisch über Angabe ihres Modellpfades durchgeführt werden. In den RTT Libraries stehen unter anderem auch Funktionen zur Zeitsteuerung und zur Abbildung von Parallelität in Echtzeittests zur Verfügung. Zusätzlich kann der Nutzer andere Pythonmodule (selbstgeschrieben oder in Form von Standardmodulen) in seine Skripte einbinden, um Testfunktionen wiederzuverwenden und seine Tests zu modularisieren. Ein Sinusgenerator kann beispielsweise durch den Rückgriff auf die Python Standardbibliothek *math* programmiert werden.

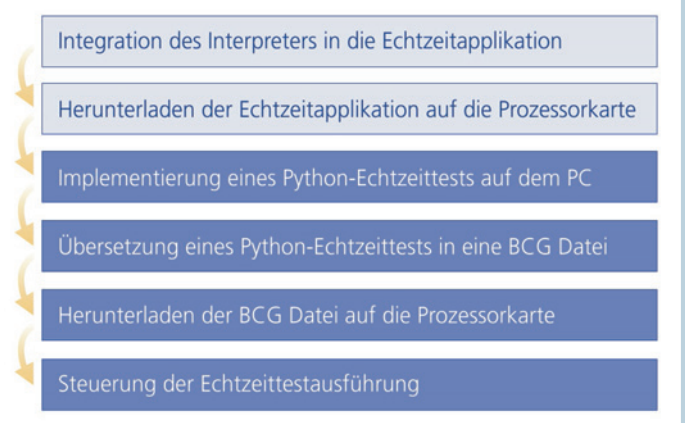

Bild 3. Workflow für die Ausführung von Echtzeittests.

Ausführungsumgebung

Um Python auf der Prozessorkarte abarbeiten zu können, wurde dort eine echtzeitfähige Python-Ausführungsumgebung implementiert (**Bild 2**, Real-Time Test Environment). Diese umfasst einerseits den echtzeitfähigen Python-Interpreter (RTT Python Interpreter), welcher Python-Tests synchron zum Modell ablaufen lassen kann. Dieser kann mehrere Echtzeittests parallel ausführen. Zudem kann auch Parallelität innerhalb eines einzelnen Python-Echtzeittests realisiert werden. Der Real-Time Test Scheduler sorgt für die korrekte Zeitsteuerung aller parallel laufenden Testaktivitäten.

Eine Service-Schicht (Real-Time Test-Management) auf der Prozessorkarte ist dafür zuständig, dass Echtzeittests auf diese heruntergeladen und in ihrem Ausführungszustand verändert werden können (z. B. Starten, Pausieren und Stoppen von Tests).

Workflow zur Ausführung von Echtzeittests

Für die Verwendung von Echtzeittests ist ein spezieller Workflow vorgesehen (**Bild 3**). Der erste Schritt besteht in der Integration des Python-Interpreters in die Echtzeitapplikation, welche auf dem HIL-Simulator ausgeführt werden soll. Die Applikation mit integriertem Python-Interpreter wird dann im zweiten Schritt auf die Prozessorkarte des

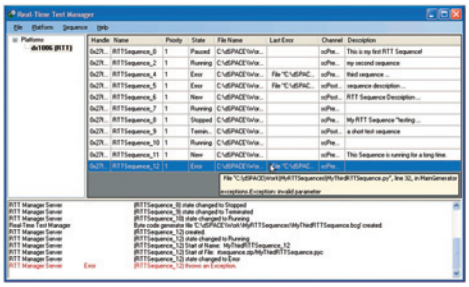

Bild 4: Real-Time Test Manager als GUI für die Verwaltung von Echtzeittests.

Simulators heruntergeladen. Nach dem Ladevorgang können mit diesem Modell Echtzeittests ausgeführt werden. Hierfür programmiert der Anwender seinen Echtzeittest in Python. Der Python-Code wird auf dem PC durch den Bytecode Generator in eine sogenannte BCG-Datei übersetzt. Diese Datei enthält neben dem eigentlichen Echtzeittest auch die durch den Test importierten Python-Module, welche für die spätere Testausführung benötigt werden.

Die BCG-Datei wird im nächsten Schritt auf die Prozessorkarte heruntergeladen. Das Herunterladen kann auch bei

schon laufenden Echtzeittests durchgeführt werden, diese werden nicht in ihrer taktsynchronen Ausführung unterbrochen. Die auf der Prozessorkarte vorhandenen Tests können dann vom Host aus gestartet und in der Testausführung gesteuert werden. Vom PC aus kann die Testbeobachtung durchgeführt werden (z. B. durch die Test- und Experimentiersoftware ControlDesk).

Testbeispiel

Folgendes Beispiel zeigt einen Echtzeittest in Python, welcher eine Fahrzeuggeschwindigkeit überwachen soll. Wird vom Test eine höhere Geschwindigkeit als 60 km/h detektiert, so soll das Bremspedal sofort, d. h. im gleichen Simulationsschritt, zu 100 Prozent betätigt werden.

```
from rttlib import variable
Velocity  = variable.Variable(r'Model Root/Velocity/Out1')
BrakePedal = variable.Variable(r'Model Root/Brake/Value')

def MainGenerator():

  while(Velocity.Value <= 60.0):
    yield None

  BrakePedal.Value = 100.0
```

Im Skript wird hierfür zuerst eine Variablenklasse variable aus dem rttlib Modul importiert. Mithilfe dieser Klasse werden zwei Variablenobjekte durch Referenzierung ihres Modellpfades erzeugt. Hierbei enthält die Variable *Velocity* die aktuelle Fahrzeuggeschwindigkeit, über den Modellparameter BrakePedal kann die Stellgröße des Bremspedals vorgegeben werden.
Die Funktion *MainGenerator* bildet für Echtzeittests den allgemeinen Einstiegspunkt, dort startet immer die Testausführung. In einer while-Schleife wird nach Testbeginn fortwährend der Momentanwert der Geschwindigkeit abgefragt (über Velocity.Value).
Mit dem Befehl *yield None* im Schleifenkörper signalisiert das Testskript, dass es erst wieder im nächsten Simulationsschritt aufgerufen werden möchte. Dadurch wird die Ausführungskontrolle nach jedem Vergleich wieder an den RTT Scheduler abgegeben, welcher das Testskript erst im nächsten Simulationsschritt des Modells reaktiviert. Dadurch wird genau ein Vergleich pro Simulationsschritt durchgeführt, so dass das Skript jede Geschwindigkeitsänderung überwachen kann.
Die while-Schleife wird erst dann verlassen, wenn Velocity den Wert 60 km/h überschreitet. Wenn dieses passiert, wird im gleichen Simulationsschritt die Vollbremsung ausgelöst (über das Schreiben von BrakePedal.Value mit dem Wert 100 Prozent) und das Testskript terminiert.

Verwaltung der Echtzeittests

Mehrere Echtzeittests können gleichzeitig auf der Prozessorkarte geladen sein und bei Bedarf gestartet werden. Um die Übersichtlichkeit zu bewahren, wird der Anwender durch eine grafische Benutzeroberfläche bei der Verwaltung der Echtzeittests unterstützt. Dieser Real-Time Test Manager (**Bild 4**) gibt einen Überblick über die auf der Prozessorkarte vorhandenen Tests (mit Testnamen, zugehörigem Dateipfad, Ausführungsstatus, Fehlerrückgaben usw.) und erlaubt das Herunterladen und die Ausführungssteuerung von Echtzeittests. Die hier angebotene Funktionalität kann durch Pythonprogrammierung auf dem PC und aus Testsequenzen von AutomationDesk heraus genutzt werden. Der Real-Time Test-Manager zeigt hierbei immer den aktuellen Stand aller auf der Prozessorkarte vorhandenen Echtzeittests an.

Benchmarkergebnisse

Mit AutomationDesk 1.4 ist der Python-Interpreter für die dSPACE Prozessorkarte DS1006 mit AMD Athlon Prozessor verfügbar. Die Ausführung der Echtzeittests benötigt (zusätzlich zur eigentlichen Modellsimulation) eine gewisse Rechenzeit. Durchgeführte Benchmarks bei einer Modellschrittweite von 1ms auf einem DS1006 (2,6 GHz Taktfrequenz) haben ergeben, dass das Einbetten des Python-Interpreters ca. 0,25 µs pro Simulationsschritt benötigt. Die fortwährende Überprüfung einer Modellgröße (wie im vorgestellten Skriptbeispiel) schlägt mit ca. 0,45 µs zu Buche. Die Ausführung einer Sinusgenerierung auf einer Modellvariable (implementiert über das Python math-Modul) kostet pro Simulationsschritt typischerweise ca. 1,3 µs. Reserviert man als Anwender 10% der Modellschrittweite für Echtzeittests (in diesem Beispiel 100 µs), so sind parallel zur Modellrechnung umfangreiche Testszenarien durch das Echtzeittesten abdeckbar.

Zusammenfassung und Ausblick

Mit dem Testautomatisierungstool AutomationDesk 1.4 von dSpace ist ein echtzeitfähiger Python-Interpreter für die DS1006 HIL-Prozessorkarte verfügbar. Dieser ermöglicht die zum Simulationsmodell taktsynchrone Ausführung mehrerer unabhängiger Echtzeittests. Hiermit können reproduzierbare, zeitkritische Steuergerätetests realisiert werden, wobei das Simulationsmodell unverändert bleiben kann. Durch die Verwendung von Python erhält der Anwender maximale Flexibilität bei der Testprogrammierung.
Im Frühjahr 2007 wird als zusätzliche Plattform eine Unterstützung für das DS1005 mit IBM PowerPC angeboten. In Zukunft wird das Echtzeittesten weiter durch dSpace ausgebaut werden (z. B. durch neue Bibliotheken für HIL-Hardwarezugriffe und komfortablen Datenaustausch zwischen Simulationsrechner und PC). Insgesamt ermöglicht die vorgestellte Echtzeittestlösung eine neue Qualität beim automatisierten Steuergerätetest am HIL-Simulator. (oe)

Dipl.-Ing. Holger Krisp ist als Produktmanager für Test- und Experimentiersoftware bei der dSpace GmbH in Paderborn tätig.

@ dSpace GmbH
www.dspace.com