Von Simulink nach OSEK: Automatische Codegenerierung für Echtzeitbetriebssysteme mit TargetLink

Lutz Köster, Thomas Thomsen, Ralf Stracke, dSPACE GmbH

Der Beitrag beschreibt die Koppelung von OSEK-Betriebssystemen mit Simulink und Stateflow, mit der ein weiterer Meilenstein in der Integration des Steuergeräteentwicklungsprozesses auf Modellebene erreicht wird. Nachdem die automatische Codegenerierung aus Modellen in Seriencodequalität bereits seit einiger Zeit mit Hilfe von TargetLink der Firma dSPACE möglich ist, beschreibt der nächste Schritt eine vollständige Integration eines Echtzeitbetriebssystems in den Codegenerierungsprozess. Hierzu bietet sich der OSEK-Standard an, welcher zu Beginn des Beitrags kurz vorgestellt wird. Im Folgenden werden Konzepte und Techniken beschrieben, die in TargetLink zum Erreichen einer OSEK-kompatiblen Codegenerierung angewandt werden. Der Beitrag zeigt, wie Betriebssystemfunktionen und -eigenschaften im Blockdiagramm spezifiziert werden, wie Code verschiedener Abtastraten vom Codegenerator in Tasks gebündelt wird, wie die Intertaskkommunikation realisiert wird und wie alles zusammen auf einem Evaluierungsboard simuliert werden kann. Der Artikel zeigt zudem in eindrucksvoller Weise, wie neue Standards die Weiterentwicklung von Entwicklungswerkzeugen fördern können und die Zusammenführung von bisher völlig getrennten Tools ermöglichen.

Einleitung

Die Softwareentwicklung für Steuergeräte wird im steigenden Umfang mit der Hilfe von modernen Simulationswerkzeugen durchgeführt, die mittels ausführbarer Blockdiagramme programmiert werden. Eine der populärsten Toolumgebungen ist MATLAB, Simulink und Stateflow (im Folgenden wird abkürzend nur noch von Simulink gesprochen) von The MathWorks, welche zum Quasi-Industriestandard bei der Steuergeräteentwicklung geworden ist. 1999 hat dSPACE den Codegenerator TargetLink auf den Markt gebracht, welcher die Simulink-Umgebung um Codegenerierung in Serienqualität ergänzt. TargetLink ist in der Lage, sowohl portablen ANSI-C-Code als auch prozessor- und compilerspezifischen C-Code zu generieren. Der Code ist gut lesbar und zudem klein genug, um auf ressourcenbegrenzten Echtzeitsteuergeräten lauffähig zu sein. Dieses wurde bereits in einer Reihe von Serien- und seriennahen Projekten nachgewiesen. dSPACE erweitert nun den Anwendungsbereich der Codegenerierung in der nächsten Version von TargetLink durch die Unterstützung von OSEK-kompatiblen Betriebssystemen.

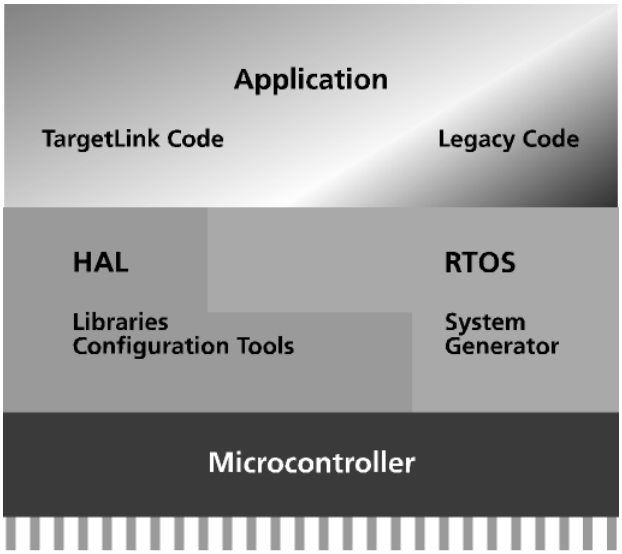

Bild 1: Softwareschichten in einem Steuergerät

Moderne Steuergeräte, wie sie heute in Kraftfahrzeugen oder Flugzeugen eingesetzt werden, bestehen aus drei typischen Softwareschichten (siehe Bild 1). Die oberste Schicht enthält den eigentlichen Anwendungscode, welcher üblicherweise aus Signalverarbeitungs-, Regelungs-, Failsafe- und Diagnosefunktionen besteht. Simulink ist in idealer Weise für die Spezifikation und Simulation dieser Funktionen geeignet. TargetLink kann anschließend zur automatischen Generierung

von Seriencode benutzt werden.

Die unterste Schicht bildet der Hardware Abstraction Layer (HAL), welcher einfache I/O Routinen, Interrupt Service Routinen, Treiber und Kommunikationshandler beinhaltet. Blockdiagramm-Spezifikationen bieten keine nennenswerten Vorteile für diese Schicht, hier wird weiterhin in C oder Assembler programmiert. Viele dieser Routinen sind bereits durch Zulieferer verfügbar oder werden über prozessorspezifische Konfigurierungstools erzeugt.

Zwischen den beiden Schichten liegt das Betriebssystem, welches verantwortlich für das Task-Scheduling, die Intertask-Kommunikation und das Ressourcen-Management ist. Kommerzielle Echtzeitbetriebssysteme für Steuergeräte sind bereits vielfach verfügbar. Es ist üblich, dass sie mit einem Konfigurator und einem Builder ausgeliefert werden. Letzterer generiert das Betriebssystem nach den Vorgaben des Anwenders. In den letzten Jahren hat sich der OSEK-Standard durchgesetzt, welcher das Application Programming Interface (API) und die Funktionalität des Betriebssystems standardisiert. Diese Spezifikation macht es nun möglich, Codegeneratoren mit Betriebssystemgeneratoren zu verbinden und den generierten Code untereinander kompatibel zu halten.

OSEK: Konzepte und Ziele

Ziel der OSEK-Spezifikation ist es, die Wiederverwendbarkeit und Portierbarkeit von Anwendungs-Software zu verbessern. Um dieses Ziel zu erreichen, wurden von den OSEK-Arbeitsgruppen Schnittstellen in den Bereichen Echtzeitbetriebssystem, Kommunikation und Netzwerk-Management spezifiziert. OSEK selbst ist also kein Betriebssystem, sondern ein Standard, der die Schnittstellen und das Verhalten eines OSEK-kompatiblen Betriebssystems definiert. Die Spezifikation unterstützt Skalierbarkeit und Konfigurierbarkeit, um sowohl die Anforderungen von kleineren Anwendungen auf Prozessoren mit beschränkten Ressourcen, als auch die Anforderungen von größeren Anwendungen auf leistungsfähigen Prozessoren zu berücksichtigen. Der Standard läßt dem Hersteller eines OSEK-kompatiblen Betriebssystems die nötige Flexibilität, die es erlaubt, eine Anpassung an bestimmte Microcontroller zu optimieren. Es sind bereits einige kommerzielle Echtzeitbetriebssysteme verfügbar, die diesem OSEK-Standard entsprechen. Sie sind im Folgenden als OSEK-Implementierungen bezeichnet.

Die großen Vorteile des OSEK-Standards sind:

- Verkürzung der Entwicklungszeit
- Erhöhte Software-Qualität durch Wiederverwendung bereits getesteter SW-Module
- Optimale Speicherausnutzung durch Skalierbarkeit
- Wiederverwendung von Software in anderen Umgebungen

Die OSEK-Spezifikation beschreibt ein statisches Echtzeitbetriebssystem, dessen Objekte (wie z.B. Tasks, Events, Messages und Ressourcen) alle zur Compile-Zeit angelegt werden. Diese Betriebssystem-Objekte werden in einer speziellen Beschreibungssprache (OIL = OSEK Implementation Language) definiert. Zu einem Objekt gehören bestimmte OIL-Attribute wie z.B. die Priorität einer Task. Zusätzlich zu den Standard-Attributen können OSEK-Hersteller weitere Attribute definieren, die z.B. ihre OSEK-Implementierung auf bestimmte Microcontroller anpassen.

Die Anwendungssoftware wird entsprechend ihren Echtzeit-Anforderungen in einzelne Tasks aufgeteilt. OSEK unterscheidet zwischen "Basic Tasks" und "Extended Tasks". Extended Tasks können während ihrer Abarbeitung auf bestimmte Events warten. Während dieser Zeit ist der Prozessor einer anderen Task zugeordnet. Basic Tasks dagegen verzichten auf diesen Mechanismus und können nur durch Tasks höherer Priorität unterbrochen werden. Sie benötigen weniger Speicher zur eigenen Verwaltung.

Die Skalierbarkeit einer OSEK-Implementierung wird durch die Einteilung des Schedulers in verschiedene Conformance-Klassen erreicht. In den beiden Basic Conformance-Klassen sind nur Basic Tasks erlaubt. In den beiden Extended Conformance-Klassen dürfen sowohl Basic Tasks, als auch Extended Tasks verwendet werden.

Die OSEK-Spezifikation definiert Ressourcen um den gemeinsamen Zugriff von verschiedenen Tasks oder Interrupt Service Routinen

(ISR) auf kritische Bereiche zu regeln.

Counter und Alarme werden in OSEK definiert, um wiederkehrende Ereignisse zu verarbeiten. Ein Alarm wird genau einem Counter zugeordnet. Der Counter zählt Zeiteinheiten oder andere wiederkehrende Ereignisse wie z.B. Interrupts von Kurbelwellensensoren. Wenn der Counter einen bestimmten Wert erreicht hat, setzt der Alarm ein Event oder aktiviert eine Task.

Die OSEK-Spezifikation beinhaltet auch eine Message-basierte Inter-Task-Kommunikation. Messages dienen zum Datenaustausch sowohl zwischen Tasks auf dem selben Steuergerät, als auch zwischen Tasks, die sich auf verschiedenen Steuergeräten befinden.

Zur Task-Synchronisation können Events genutzt werden. Dabei wartet eine Extended Task auf ein Event, welches von einer Task, einem Alarm oder einer Interrupt Service Routine gesetzt werden kann.

Modellierung von Echtzeit- (OSEK-) Anwendungen in Simulink/TargetLink

Allgemeine Modellierungsstile

Simulink ist heute *das* Standardwerkzeug für Modellierung und Offline-Simulation regelungstechnischer Probleme. Sein Anwendungsspektrum ist stetig gewachsen und neben der klassischen Simulation dynamischer Systeme wird es zunehmend als Spezifikationsbasis aller Phasen im Entwicklungsprozess elektronischer Steuergeräte verwendet. Als grafische und ausführbare Spezifikation ersetzt es dabei die klassischen verbalen Ansätze. Dieser erweiterte Anwendungsbereich bringt neue Anforderungen mit sich. In Verbindung mit der seit kurzem verfügbaren automatischen Generierung von hocheffizientem, produktionstauglichem C-Code, wird Simulink zur Beschreibung der gesamten Applikationssoftware von Steuergeräten benutzt. In diesem Kontext ist die Verwendung eines Echtzeitbetriebssystemes von großer Bedeutung und für automotive Anwendungen beginnt sich hier der OSEK-Standard durchzusetzen. Die neue Version des Codegenerators TargetLink der Firma dSPACE bietet deshalb eine nahtlose Integration von OSEK-konformen Betriebssystemen in die Simulink-Umgebung und den hieraus generierten Code. Im Gegensatz zu vielen CASE-Tools, die ihren Ursprung direkt in der Softwareentwicklung haben und in deren Welt Objekte wie Tasks, Messages, Ressourcen usw. einen natürlichen Teil darstellen, ist die klassische Domäne von Simulink die Funktionsentwicklung mit einer komplett verschiedenen Abstraktionsebene. Automatische Codegenerierung aus Simulink für Echtzeitbetriebssysteme sieht sich zwei grundsätzlichen Problemen gegenübergestellt:

- der Abbildung von Simulink-Modellierungskonzepten auf verfügbare Betriebssystemobjekte
- der Schaffung neuer Spezifikationsmöglichkeiten für wichtige Betriebssystemeigenschaften

Dabei muß der Codegenerator sowohl ein *unverändertes* Simulink-Modell automatisch - also durch wohlüberlegte Standardeinstellungen - optimal in Echtzeit implementieren können, als auch umfangreiche zusätzliche Optionen bieten, die eine exakte Anpassung an spezielle Anforderungen erlauben.

Eine Echtzeitanwendung, die auf einem Echtzeitbetriebssystem (RTOS) basiert, ist in unterschiedliche Software-Einheiten aufgeteilt. Diese sogenannten Tasks fassen Softwareabschnitte mit gemeinsamen Echtzeitanforderungen wie Timing, Priorität usw. zusammen. Sie werden vom RTOS automatisch zu definierten Zeiten aufgerufen oder durch asynchrone Hard- oder Softwareereignisse aktiviert. Ein Simulink-Modell kennt ursprünglich keine Tasks, sondern ist aus anderen Objekten aufgebaut, die der Codegenerator auf Betriebssystemeinheiten abbilden muß.

Simulink nutzt zur hierarchischen Strukturierung des Modells sogenannte Subsysteme, die zu einer gewissen Funktionalität gehörende Basisblöcke zusammenfassen. Diese Basisblöcke werden entweder zyklisch mit einer bestimmten Abtastrate oder ereignisgetrieben abgearbeitet.

Bild 2 zeigt ein typisches Simulink/Stateflow Diagramm, das zyklische und ereignisgetriebene Modellteile enthält. Ein automotives Motorsteuergerät beispielsweise würde die

Drosselklappenregelung periodisch und die Berechnung von Zündung und Einspritzung von Kurbelwelleninterrupts getrieben ausführen. Um dieses Modell als OSEK-Echtzeitanwendung zu implementieren, werden bei der Codegenerierung folgende Schritte durchlaufen.

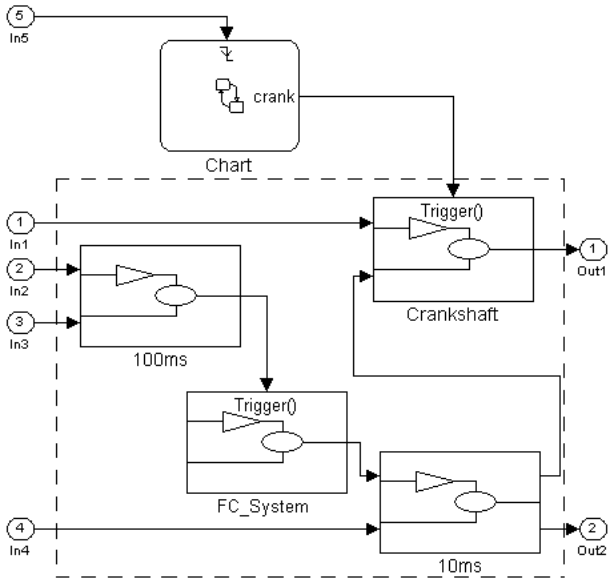

Bild 2: Typisches Simulink/Stateflow Diagram

Das zehn und das einhundert Millisekunden Subsystem wird jeweils als separate Task implementiert. Die zugehörigen C-Funktionen werden durch entsprechende OSEK-Schlüsselwörter gekennzeichnet. Zusätzlich wird das Betriebssystem so konfiguriert, dass diese Tasks automatisch in der korrekten Periodendauer aufgerufen werden. Hierfür werden OSEK-Alarme definiert, geeigneten Timern zugeordnet und in einer speziellen Startup-Routine durch entsprechende Betriebssystemaufrufe initialisiert.

Das zum Kurbelwellensignal synchrone Subsystem wird ebenfalls als eigene Task implementiert. In Bild 2 übernimmt ein Statechart den Aufruf dieses Subsystems. Das ist aber nur für die Offline-Simulation so modelliert, im realen Steuergerät soll die Task üblicherweise direkt aus einem Kurbelwelleninterrupt heraus aktiviert werden. Der Codegenerator muß also das Statechart ignorieren. Simulink-Diagramme enthalten häufig Teile, die nur in der Simulation benötigt werden, z.B. Streckenmodelle um die Regelschleife zu schließen. Um dieses Konzept zu unterstützen, erzeugt TargetLink nur für einen definierten Modellabschnitt, in Bild 2 durch eine gestrichelte Linie gekennzeichnet, Code.

Asynchrone Funktionen werden in Simulink durch getriggerte Subsysteme modelliert. Sollen sie als separate Tasks implementiert werden, müssen sie von der im Modell spezifizierten Triggerquelle aktiviert werden. *FC_System* in Bild 2 wird aus Task *100ms* heraus aktiviert. TargetLink sorgt automatisch für entsprechende Betriebssystemaufrufe. Liegt die Triggerquelle außerhalb des vom Codegenerator bearbeiteten Bereichs, wie die für *Crankshaft*, muß der Anwender die entsprechende Taskaktivierung manuell der gewünschten Triggerquelle, also z.B. einem Interrupt, zuordnen.

Der nächste wichtige Schritt ist die Umsetzung der Datenverbindungen zwischen den Tasks. Dabei muß berücksichtigt werden, dass Tasks sich ggf. gegenseitig unterbrechen können und dass daher einige Datenzugriffe geschützt werden müssen, um Inkonsistenzen zu verhindern. TargetLink legt z.B. für bestimmte globale Daten lokale Kopien für die weitere Bearbeitung an. Datenverbindungen, die im Modell den Sichtbarkeitsbereich des Codegenerators verlassen, können z.B. durch Betriebssystem-'Messages' implementiert werden. Dies betrifft in der realen Applikation z.B. Verbindungen zum Datenaustausch mit handgeschriebenem Code auf dem gleichen Steuergerät oder zur Kommunikation mit anderen Steuergeräten.

Im letzten Schritt wird eine Offline-Beschreibung der Gesamtapplikation in Form einer OIL-Datei generiert. Mit den hierin enthaltenen Informationen kann der sogenannte 'System Builder' des Betriebssystemherstellers einen optimal angepaßten Echtzeitkern erzeugen.

In den folgenden Abschnitten werden die genannten Schritte der Codegenerierung detaillierter beschrieben.

Tasks

Wie im letzten Abschnitt beschrieben, wird das Simulink-Modell in einzelne Tasks aufgeteilt. Dabei verwendet TargetLink entweder eine Standardaufteilung oder der Anwender bestimmt selbst die Zuordnung von Subsyste-

men zu Tasks.

Standardmäßig faßt TargetLink alle periodischen Subsysteme gleicher Abtastrate zu gemeinsamen Tasks zusammen. Getriggerte Subsysteme mit gemeinsamer Triggerquelle werden ebenfalls zusammengefaßt. Ihr Code wird entweder der Task zugeordnet, die den Trigger auslöst, wenn diese im Sichtbarkeitsbereich des Codegenerators liegt (*FC_System* in Bild 2 wird Task *100ms* zugeordnet), oder einer separaten Task (*Crankshaft* in Bild 2).

Durch Plazieren eines speziellen *Task Block* in ein Subsystem überschreibt der Anwender die Standardaufteilung des Codegenerators. Dieser Block ordnet ein Subsystem explizit einer auswählbaren Task zu. *FC_System* in Bild 3 wird nun als separate *Task C* implementiert und nicht mehr direkt in *100ms* ausgeführt. Dort wird nur noch das *Task C* zugehörige OSEK-Kommando 'activateTask' aufgerufen und das Betriebssystem entscheidet anhand der unterschiedlichen Taskprioritäten und Unterbrechbarkeiten, wann *Task C* wirklich ausgeführt wird.

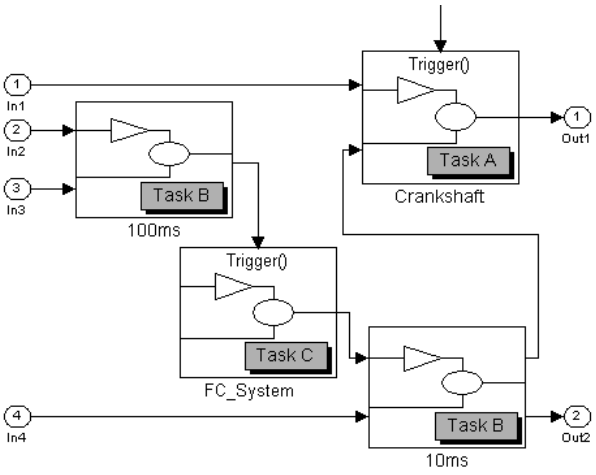

Bild 3: Task-Block im Simulink-Diagram

Wie in Bild 3 gezeigt, können selbst Subsysteme unterschiedlicher Abtastrate der gleichen Task zugeordnet werden. Der Code für *100ms* und *10ms* wird nun in die gemeinsame *Task B* eingebettet, die periodisch alle 10 Millisekunden - das ist der größte gemeinsame Teiler beider Abtastraten - aufgerufen wird. Ein Zähler kapselt den Code von *100ms* und führt ihn nur bei jedem zehnten Taskaufruf aus. Eine solche Konstruktion wird jedoch wegen bestimmter Prozessorauslastungsnachteile eher die Ausnahme sein.

Diese Möglichkeit der Zusammenfassung unterschiedlicher, über das Modell verteilter Subsysteme hat noch einen anderen wichtigen Aspekt. Es ist nicht notwendig, das Modell auf oberster Ebene im Hinblick auf Abtastraten oder Zugehörigkeit zu bestimmten Ereignissen aufzuteilen, man kann stattdessen auch eine funktionsorientierte Strukturierung wählen. Ein Steuergerät enthält üblicherweise eine größere Zahl sogenannter *Features* (z.B. Drosselklappenregelung, Leerlaufregelung, Abgasrückführung etc.), die häufig von verschiedenen Teams entwickelt und sinnvollerweise als getrennte Subsysteme dargestellt werden. Jedes dieser Features kann wiederum Teile enthalten, die z.B. im 10-Millisekunden-Raster, synchron zur Kurbelwelle usw. abgearbeitet werden müssen. Faßt man diese verteilten Einzelsysteme gleicher Periodizität bzw. Eventzugehörigkeit - häufig Prozesse genannt - zusammen, spart man Task-Overhead.

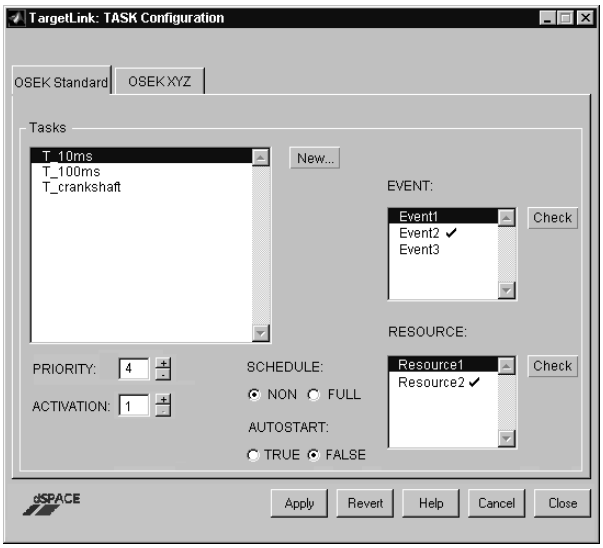

Bild 4: Task Block Dialog

Mit dem Task-Block kann man nicht nur Subsysteme bestimmten Tasks zuordnen, man kann mit ihm auch deren Eigenschaften einstellen. Bild 4 zeigt den Dialog des Task-Blockes. Es können existierende Tasks ausgewählt oder neue Tasks angelegt werden und es werden die im Standard von OSEK vorgeschriebenen Attribute wie Priorität, Anzahl der Aktivierungen, Unterbrechbarkeit, benutzte Resourcen oder zugehörige Events, spezifiziert. Die hier eingestellten Eigenschaften werden in die OIL-Beschreibung übernommen und sie werden z.T. auch genutzt, um den generierten Code zu optimieren. Die Priorität

und das Scheduling beeinflussen z.B. den Datenaustausch zwischen Tasks. Bild 4 zeigt, dass es neben der *OSEK Standard* Seite des Dialogs auch noch eine *OSEK XYZ* Seite gibt, die hier stellvertretend für eine spezielle OSEK-Implementierung steht. Der OSEK-Standard erlaubt neben den vorgeschriebenen Attributen auch implementierungsspezifische Optionen, die für eine Reihe von kommerziellen OSEK-Produkten direkt in der TargetLink-Umgebung spezifiziert werden können. TargetLink ist aber nicht auf diese direkt unterstützten OSEK-Implementierungen beschränkt. In einem späteren Abschnitt wird gezeigt, wie TargetLink mit jedem beliebigen Produkt und den zugehörigen Konfigurationswerkzeugen zusammenarbeiten kann.

Wenn Tasks erzeugt werden, entweder durch die Standardpartitionierung von TargetLink oder durch explizite Anwenderwünsche, werden die notwendigen Taskdeklarationen in den Code generiert (z.B. `Task (T_100ms) {...}` und die Taskaktivierung wird initiiert. Periodische Tasks werden automatisch passenden OSEK-Alarmen zugeordnet, die in einer Startup-Funktion initialisiert werden. Getriggerte Subsysteme werden von Code aktiviert, der in die Task der Triggerquelle generiert wird. Ist diese Quelle außerhalb des Sichtbarkeitsbereiches von TargetLink, wird ein Stück Code (Makro) erzeugt, das die Taskaktivierung enthält und das vom Anwender an die zugehörige Triggerquelle, z.B. einen Interrupt, gehängt werden muß.

Inter-Task-Kommunikation

In einer preemptiven Multitaskingumgebung stellt der Datenaustausch zwischen Tasks eine der wichtigen Aufgaben des Betriebssystems dar. Im Folgenden soll nur auf die Kommunikation zwischen Tasks auf dem gleichen Prozessor eingegangen werden.

Bild 5 zeigt ein Szenario, in dem Tasks mit unterschiedlichen Abtastraten, Prioritäten und Unterbrechbarkeiten Daten austauschen. *Task A* sendet *signal1* und *signal2* an *Task B*. Für diesen Datenaustausch werden üblicherweise globale Variablen benutzt. Da *Task A* unterbrechbar ist (in OSEK-Terminologie SCHEDULE = FULL) und *Task B* eine höhere Priorität besitzt, kann es vorkommen, dass *Task A* unterbrochen wird, nachdem *signal1* aktualisiert wurde, aber *signal2* noch seinen alten Wert hat. *Task B* würde nun mit Mischdaten aus unterschiedlichen Zeitschritten arbeiten, was zu ungewolltem Verhalten führen kann und daher in den meisten Fällen vermieden werden muß. In diesem Beispiel ist *Task A* dafür verantwortlich, einen geeigneten Konsistenzsicherungsschutz einzubauen. So kann *Task A* z.B. lokale Kopien der Daten nutzen, die dann am Ende den zugehörigen globalen Variablen zugewiesen werden. Dieser Kopiervorgang muß gegen Unterbrechung geschützt werden, z.B. durch Sperren von Interrupts oder Belegen von OSEK-Ressourcen. Durch die Einführung lokaler Arbeitskopien wird dieser kritische Abschnitt aber so kurz wie möglich gehalten.

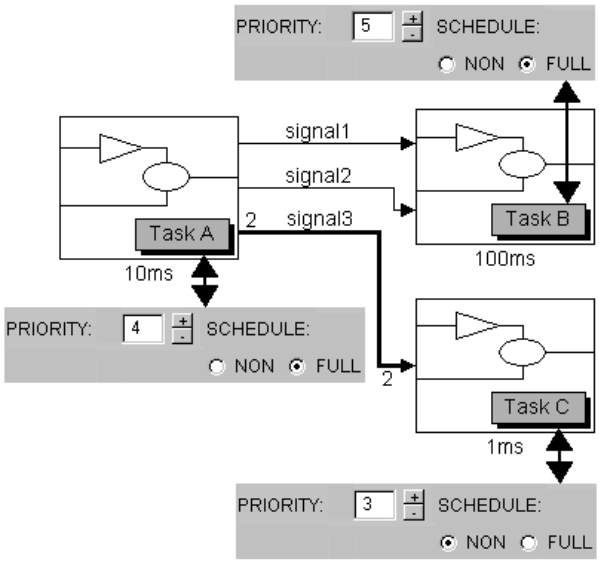

Bild 5: Kommunikation zwischen Tasks

Es hängt von den Taskeinstellungen und der damit verbundenen Frage 'wer kann wen unterbrechen' ab, ob überhaupt Konsistenzsicherung nötig ist, und wenn ja, welche Task diese implementieren muß. In Bild 5 ist auch das vektorielle *signal3* potenziell der Gefahr von Inkonsistenzen ausgesetzt. Die speziellen Taskeinstellungen in Bild 5 verhindern in diesem Fall jedoch eine Inkonsistenz. Der Sender kann nicht vom Empfänger unterbrochen werden (*Task A* hat eine höhere Priorität als *Task C*) und umgekehrt (*Task C* ist nicht unterbrechbar, da SCHEDULE = NON). In diesem Fall können beide Tasks direkt, ohne Konsistenzsicherung, mit der zu *signal3* gehörigen globalen Variablen arbeiten, was die effizienteste Art des Datenaustausches dar-

stellt.

Betriebssysteme bieten üblicherweise Dienste zum geschützten Datenaustausch zwischen Tasks. OSEK definiert dafür sogenannte Messages, die allerdings für den Datenverkehr zwischen Tasks auf dem gleichen Steuergerät nicht immer die beste Lösung sind, da Konsistenzsicherung, wie das letzte Beispiel gezeigt hat, nicht immer notwendig ist. TargetLink wählt daher, abhängig von den Taskeinstellungen, automatisch den effizientesten Weg des Datenaustauschs (globale Variablen mit oder ohne lokale Kopien, Interrupt-Sperre, Ressourcen-Belegung, usw.). Dieses voreingestellte Verhalten kann jederzeit durch explizite Anwenderangaben überschrieben werden.

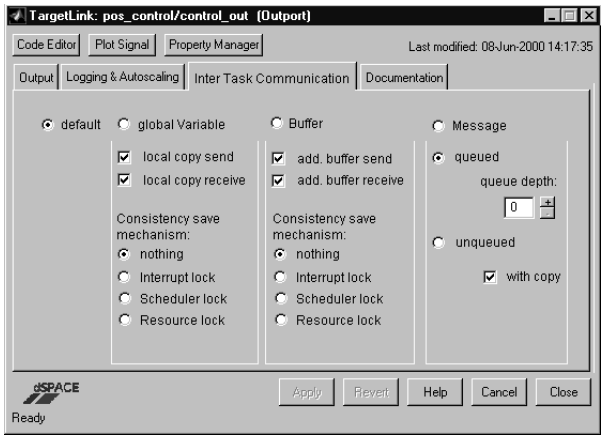

Bild 6: Dialog Inter-Task-Kommunikation

Spezialblöcke

TargetLink stellt Spezialblöcke zur Verfügung, die es erlauben, bestimmte OSEK-Eigenschaften zu spezifizieren und deren Verhalten in der Simulation nachzubilden. Als ein Beispiel dafür, soll im Folgenden der Alarm/Counter-Block näher beschrieben werden.

Alarme werden von TargetLink schon automatisch benutzt, um periodische Tasks aufzusetzen. Darüber hinaus wird ein Spezialblock angeboten, der die Funktionalität des OSEK-Alarms dem Anwender ganz allgemein im Modell verfügbar macht. Da in OSEK ein Alarm immer einem Counter zugeordnet ist, wird der TargetLink Alarm-Block immer zusammen mit einem speziellen Counter-Block verwendet (siehe Bild 7). Für den Zähler wird nicht wirklich Code generiert, da dieser ja in der realen Applikation durch einen Hardware- oder Betriebssystemzähler implementiert wird. Der Counter-Block wird zur Offline-Simulation benutzt. Der Blockdialog erlaubt die Einstellung der OSEK-Attribute und des daraus resultierenden Simulationsverhaltens. Der Ausgang eines Counters ist mit einem oder mehreren Alarm-Blöcken verbunden. In diesem wird spezifiziert, bei welchem Zählerstand der Alarm (zum ersten Mal) ausgelöst und damit die Task aktiviert wird, die mit dem Alarm per Triggerverbindung verbunden ist. *Cycle* definiert optional die Periodizität des Alarms. Alarme werden entweder bei der Systeminitialisierung aufgesetzt, z.B. die zum Aktivieren der Subsysteme mit fester Abtastrate, oder wenn zur Laufzeit ein bestimmtes Ereignis eintritt. Der zweite Fall wird durch einen optionalen Triggereingang des Alarmblocks modelliert, der von beliebiger Stelle innerhalb des Modells gespeist werden kann. Absolute Alarme lösen bei absoluten Zählerständen aus, relative Alarme nach der spezifizierten Anzahl von Zählerinkrementen nachdem der Alarm aufgesetzt wurde.

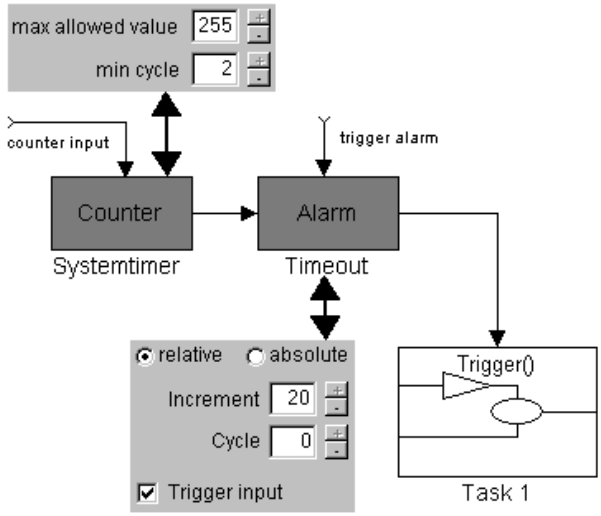

Bild 7: Alarm/Counter-Block

OSEK-Konfiguration und Zusammenspiel verschiedener Werkzeuge

Eine typische Steuergeräteapplikation besteht aus mehreren C-Quelldateien, von denen einige beispielsweise von TargetLink erzeugt, andere von Handprogrammierern geliefert wurden. Eine gemeinsame OIL-Beschreibung dient als Eingabe für den System-Generator, der das Betriebssystem optimal für die Applikation konfiguriert. Der so angepaßte

Echtzeitkern wird zusammen mit den anderen Sourcen zum endgültigen ausführbaren Programm gelinkt (siehe Bild 8).

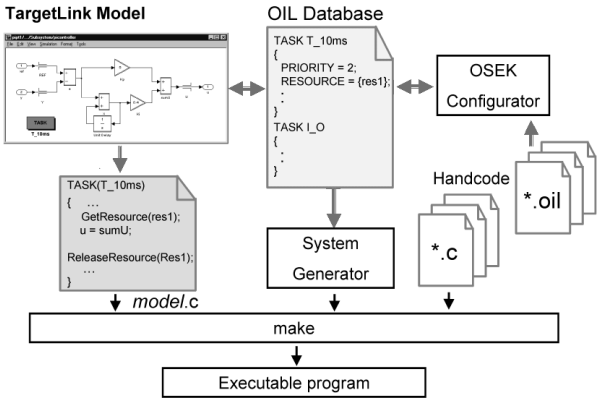

Bild 8: Zusammenspiel der Entwicklungswerkzeuge

Zur Erzeugung der textuellen OIL-Datei gibt es verschiedene Möglichkeiten. Zum einen sollten die Eigenschaften der aus dem Simulink-Modell stammenden Betriebssystemobjekte sinnvollerweise direkt auf Modellebene spezifiziert, sowie die zugehörigen OIL-Abschnitte automatisch generiert werden können. Zum anderen liefern die meisten OSEK-Anbieter komfortable grafische Tools zur Konfiguration der OIL-Datei. Deren Nutzung ist sicherlich für die handgeschriebenen Programmteile sinnvoll. Für ein nahtloses Zusammenspiel beider Werkzeuge benutzt TargetLink eine externe OIL-Datei als Datenbasis für die Betriebssystemeinstellungen, aus der heraus importiert und in die hinein exportiert werden kann. Simulink-Komponenten (z.B. Subsysteme) können OSEK-Objekten (z.B. Tasks) zugewiesen werden, die bereits in der Datenbasis existieren, oder es können neue Einträge angelegt werden. Der Anwender kann frei entscheiden, was das für ihn geeignetste Tool zum Spezifizieren der Attribute ist.

Es ist z.B. möglich, die Basiseinstellungen einer Task (das sind die im Standard vorgeschriebenen) in der TargetLink Umgebung einzustellen und die Beschreibung später mit Hilfe des OIL-Konfigurators mit den implementierungsspezifischen Attributen zu komplettieren.

Die Verwendung der gemeinsamen, externen OIL-Datenbasis gewährleistet Konsistenz, d.h. Änderungen, die in einem Tool vorgenommen werden, werden automatisch vom anderen übernommen.

Simulation von OSEK-Applikationen

Eine der großen Stärken von Simulink ist, dass das Verhalten eines entworfenen Steuergerätes direkt auf dem PC simuliert werden kann. Dazu wird häufig ein Streckenmodell benutzt, um den Regelkreis zu schließen (siehe Bild 9).

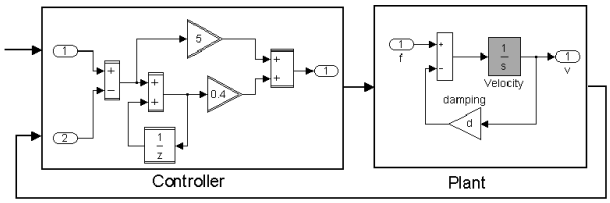

Bild 9: Floating-Point Simulation

Neben diesem Floating-Point Simulations-Modus in Simulink bietet TargetLink zwei weitere Simulations-Modi:

- Production Code Host Simulations-Modus
- Production Code Target Simulations-Modus

Bei einer Production Code Host Simulation wird der von TargetLink generierte Code anstelle der Simulink-Blöcke verwendet (Software-in-the-loop). In Bild 9 würde Subsystem *Controller* durch einen einzigen Block ersetzt werden (S-Function Block), der den generierten Code enthält. Mit diesem Simulations-Modus kann der generierte Code getestet werden. Effekte, wie zum Beispiel Quantisierungsfehler, Sättigung oder Überläufe, die durch Festkomma-Arithmetik entstehen, können untersucht werden. Diese Effekte erscheinen auf dem PC genauso wie auf dem Seriensteuergerät.

Bei einer Production Code Target Simulation wird der von TargetLink erzeugte Seriencode auf einem Evaluierungsboard gerechnet, das den gleichen Microcontroller enthält wie das Seriensteuergerät (processor-in-the-loop). Simulink liefert weiterhin die - vom Streckenmodell errechneten - Stimuli für das Steuergerät und wertet die Steuergeräteausgänge aus. Diese Signale werden vom Microcontroller über die serielle Schnittstelle vom Host-PC empfangen bzw. zum Host-PC gesendet. Mit

diesem Simulations-Modus kann der generierte Code unter realistischen ‚closed-loop'-Bedingungen getestet werden. Da der Compiler Teil der Testschleife ist, wird er in der Simulation mit überprüft.

Simulation ohne echtes OSEK RTOS

Um die Vorteile der beschriebenen Production Code Simulationen voll nutzen zu können, muß der unveränderte Produktionscode, der ja im allgemeinen auch OSEK-Funktionsaufrufe enthält, verwendet werden. Da nicht in allen Fällen davon ausgegangen werden kann, dass der Anwender für die Simulation eine kommerzielle OSEK-Implementierung zur Verfügung hat, kann TargetLink zusätzlichen Code erzeugen, der die verwendeten OSEK-Makros und OSEK-Betriebssystemfunktionen in einer einfachen Art nachbildet. Diese müssen nicht ein wirkliches Echtzeitbetriebssystem bilden, sie können auf die speziellen Anforderungen dieser ‚closed-loop'-Simulation zugeschnitten sein, die nicht in Echtzeit ablaufen kann.

Während der Simulation steht genau fest, welche Task in welchem Zeitschritt ausgeführt wird. Für die Simulation auf dem Evaluierungsboard wird vom Host-PC ein ‚ActivateTask'-Kommando zur Zielhardware gesandt. Ein einfacher Scheduler-Ersatz startet dann die Tasks durch einfachen Funktionsaufruf. Komplizierte Taskumschaltungen eines preemptiven Schedulers mit Sicherung des Taskkontextes usw. sind nicht notwendig, da die durch die Ausführungszeit einer Task auftretenden Effekte wie z.B. gegenseitiges Unterbrechen von quasi gleichzeitig gestarteten Tasks, in einer solchen Offline-Simulation nicht nachgebildet werden können. Während die Zielhardware den Task-Code abarbeitet, wartet der Host-PC auf die Antwort und die Simulationszeit wird angehalten so dass keine andere Task lauffähig wird.

Das Vorgehen im Production Code Host Simulations-Modus ist analog. Der Produktionscode und der Simulations-Frame, der die OSEK-Dienste nachbildet, werden in diesem Modus auf dem Host-PC gerechnet.

Simulation mit echtem OSEK RTOS

Die besten Ergebnisse werden erzielt, wenn schon für die Simulation alle Software-Module der fertigen Applikation eingebunden werden. Das schließt die originale OSEK-Implementierung mit ein und ist daher nur auf dem Microcontroller möglich. Das korrekte Zusammenspiel zwischen dem generierten Produktionscode und dem Betriebssystem kann auf diese Weise getestet werden. Da der Fortschritt der Simulation auch von Modellteilen abhängt, die auf dem Host-PC gerechnet werden (Streckenmodell) und daher nicht Echtzeit entspricht, muß die Zielhardware von ihren Echtzeit-Eingängen, z.B. Hardwarecountern oder Interrupts, abgetrennt werden. Diese Stimuli werden vom Host-PC (Simulink) entsprechend der aktuellen Simulationszeit geliefert (siehe Bild 10).

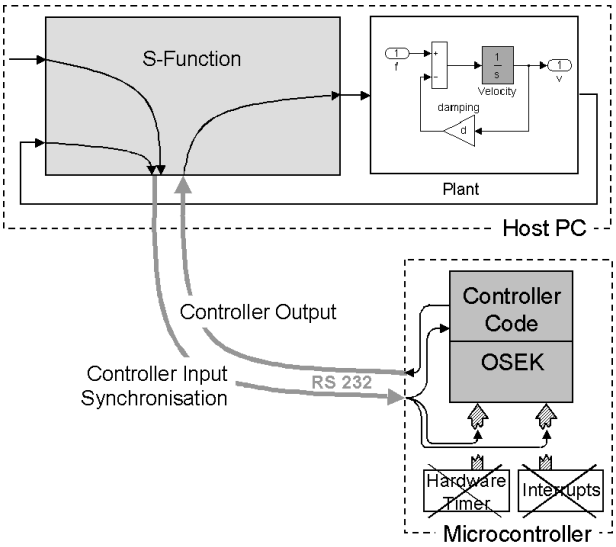

Bild 10: Target Simulation mit OSEK RTOS

Man stelle sich eine Anwendung mit zwei periodischen Tasks vor. Die Tasks haben Abtastraten von 10ms bzw. 100ms. Die entsprechenden Alarme sind mit dem System-Timer verbunden. Wenn nun der System-Timer in der realen Applikation ein Vielfaches von 10ms bzw. 100ms erreicht, wird die entsprechende Task über einen Alarm aktiviert. Für die Simulation wird der System-Timer von der Timer-Hardware getrennt und ohne Eingriff von außen würde keine Task aktiviert. Wenn die Simulationszeit in Simulink nun entsprechende Werte erreicht (10ms, 20ms, 30ms, ...), bekommt die Zielhardware vom Host-PC das Kommando, den System-Timer auf den jeweiligen Zeitwert zu erhöhen. Dadurch löst der OSEK-Alarm aus und die Aktivierung der Tasks erfolgt wie im Seriensteuergerät. Nachdem der Task-Code abgearbeitet wurde, wer-

den die errechneten Ergebnisse zum Host-PC gesandt. Da die Systemzeit auf der Zielhardware von Simulink vorgegeben wird, können natürlich nicht alle Echtzeiteffekte nachgebildet werden. Mit dieser neuen Art der Simulation kann jedoch das Counter- und Alarm-Handling, das Message- und Event-Handling und die Task-Aktivierung überprüft werden.

Spezifische Codegenerierung für kommerzielle OSEK-Implementierungen

Wie bereits erwähnt, läßt die OSEK-Spezifikation eine ausreichende Flexibilität zu, um die Eigenschaften der verschiedenen Microcontroller optimal auszunutzen. Deshalb läßt sich nicht vermeiden, dass bestimmte Funktionen in verschiedenen OSEK-Implementierungen unterschiedliche Benutzerschnittstellen haben.

Als Beispiel seien hier die API-Funktionen für Counter erwähnt. Diese sind nicht durch OSEK festgelegt. Deshalb existieren leichte Unterschiede im Code für ein und die selbe Anwendung, wenn sie auf zwei unterschiedlichen OSEK-Implementierungen laufen soll. Wenn immer es möglich ist, verwendet TargetLink standardisierte Betriebssystemfunktionen. Falls es nötig ist, benutzt TargetLink jedoch auch implementierungsspezifische Betriebssystemfunktionen um effizienten und lesbaren Code zu erzeugen.

Kooperationen zwischen OSEK-Anbietern und dSPACE garantieren ein optimales Zusammenspiel zwischen den einzelnen OSEK-Implementierungen und TargetLink.

Ein anderes Beispiel für Unterschiede zwischen den OSEK-Implementierungen sind die OIL-Attribute. Viele davon sind im OSEK-Standard definiert, in jeder OSEK-Implementierung können vom Hersteller aber noch beliebig viele implementierungsspezifische Attribute definiert werden. TargetLink bietet einen Mechanismus, der diese Erweiterungen zum Standard erkennt und dem Anwender die Möglichkeit gibt, auch die implementierungsspezifischen Attribute einzustellen.

Möchte ein Anwender sein Programm von einer OSEK-Implementierung auf eine andere übertragen, ersetzt TargetLink automatisch die implementierungsspezifischen Funktionsaufrufe und OIL-Attribute. Das erhöht die Portierbarkeit und Wiederverwendbarkeit. Obwohl OSEK hauptsächlich aus diesem Grund spezifiziert wurde, müssen die gerade beschriebenen Schritte manuell durchgeführt werden, wenn keine automatischer Codegenerator verfügbar ist.

Zusammenfassung

Eine ständig wachsende Zahl von Anwendern in unterschiedlichen automotiven Entwicklungsbereichen nutzt MATLAB/Simulink, das Quasi-Standardwerkzeug des Regelungstechnikers, als Spezifikationsbasis für den gesamten Entwicklungszyklus elektronischer Steuergeräte. Das wird insbesondere möglich durch die Verfügbarkeit von Tools wie TargetLink, die aus Simulink-Modellen automatisch produktionstauglichen C-Code generieren können, der genauso effizient wie handgeschriebener Code ist. Für die Implementierung vollständiger Applikationen ist der Anschluß an ein Echtzeitbetriebssystem und dabei besonders an den weitverbreiteten Standard OSEK von großer Bedeutung. Dieser Artikel hat gezeigt, wie der Anschluß durch TargetLink gelöst wurde und wie hier die Modellierungskonzepte von Simulink auf OSEK-Funktionalität abgebildet werden. Die Funktionsentwickler können weiterhin in der vertrauten Umgebung arbeiten, die ihre Eignung in unzähligen Anwendungen weltweit unter Beweis gestellt hat, und bekommen zusätzlich all die Flexibilität und Anpaßbarkeit, die für die Softwareentwicklung eingebetteter Systeme notwendig ist. Es wurden beispielhaft einige Erweiterungen zum Simulink-Blocksatz vorgestellt, die OSEK-Dienste und Eigenschaften im Modell verfügbar machen und die optimal an die Simulink-Philosophie angepasst sind. Es wurde gezeigt, dass Simulink, entgegen einiger anders lautender Aussagen, mit den beschriebenen Erweiterungen sehr gut geeignet ist, um eingebettete Systeme zu modellieren.

Literaturverzeichnis

[1] OSEK/VDX Operating System, Version 2.1, Mai 2000

[2] OSEK/VDX System Generation OIL: OSEK Implementation Language, Version 2.2, Juli 2000

[3] OSEK/VDX Communication, Version 2.2.1, September 2000

[4] TargetLink Production Code Generation Guide, dSPACE GmbH Paderborn, Mai 2000

[5] Simulink User's Guide, The MathWorks, Nattick, MA USA 1999

[6] H. Hanselmann, U. Kiffmeier, L. Köster, M. Meyer, "Automatic Generation of Production Quality Code for ECUs", SAE 99, March 1-4, Detroit

Kontakt

Dr. Ing. Lutz Köster
dSPACE GmbH Technologiepark 25
33100 Paderborn
lkoester@dspace.de http://www.dspace.de

Dipl.- Ing. Thomas Thomsen
dSPACE GmbH Technologiepark 25
33100 Paderborn
tthomsen@dspace.de http://www.dspace.de

Dipl.- Inf. Ralf Stracke
dSPACE GmbH Technologiepark 25
33100 Paderborn
rstracke@dspace.de http://www.dspace.de

OSEK/VDX: http://www.osek-vdx.org

Headquarters in Germany

dSPACE GmbH
Technologiepark 25
33100 Paderborn
Tel.: +49 5251 1638-0
Fax: +49 5251 66529
info@dspace.de
www.dspace.de

USA and Canada

dSPACE Inc.
28700 Cabot Drive · Suite 1100
Novi · MI 48377
Tel.: +1 248 567 1300
Fax: +1 248 567 0130
info@dspaceinc.com
www.dspaceinc.com

United Kingdom

dSPACE Ltd.
2nd Floor Westminster House
Spitfire Close
Ermine Business Park
Huntingdon
Cambridgeshire PE29 6XY
Tel.: +44 1480 410700
Fax: +44 1480 410701
info@dspace.ltd.uk
www.dspace.ltd.uk

France

dSPACE SARL
Parc Burospace
Bâtiment 17
Route de la Plaine de Gisy
91573 Bièvres Cedex
Tel.: +33 1 6935 5060
Fax: +33 1 6935 5061
info@dspace.fr
www.dspace.fr