Embedded-Softwareentwicklung für Cyber-Physical Systems

Kreativität entfesseln

Marco Schmid

Embedded-System-Entwickler sehen sich im Zuge von Industrie 4.0 und deren Technologietreibern Cyber-Physical Systems und Internet of Things mit neuen Herausforderungen konfrontiert. Wie sich diese bei Ersteren meistern lassen, wird anhand der grafischen Systemdesign-Plattform LabVIEW aufgezeigt.

Bei Cyber-Physical Systems (CPS) geht es nicht wie bisher „nur" um Hardwareschnittstellen, Interrupts, RTOS, Multitasking und Memory-Management. Im Zentrum stehen Informationen: aus physikalischen Vorgängen gewonnen, in Echtzeit berechnet, via Netzwerk überall verfügbar und Teil dynamischer Regelprozesse. Die zwangsläufig höhere Komplexität bietet gleichzeitig die Chance, sich am Markt zu differenzieren. Entwickler sind aber keine Supermenschen. Teams, vor allem in KMUs, lassen sich nicht von heute auf morgen vergrößern, Geld wird auch nicht sofort in Strömen fließen, und der Termindruck nimmt eher zu als ab.

Bleibt also ein Verstärker für das menschliche Gehirn und die Beschleunigung der Entwicklungsmethode als Ausweg aus dem Dilemma. Vorgestellt wird eine Design- und Entwicklungsplattform, die sich einerseits für „smarte" Embedded Systems eignet, speziell jedoch für deren nächste Generation, den Cyber-Physical Systems (www.cyberphysicalsystems.org) (s. Abb. 1).

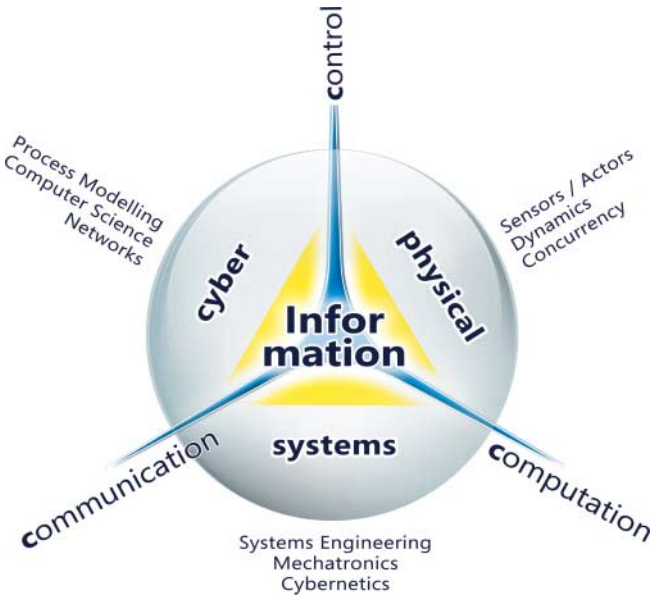
Mit visueller Programmierung Kreativität entfalten

Die Erkenntnis, dass das Hirn bildliche Ausdrücke besonders effizient verarbeiten kann, führte früh zum Konzept visueller Programmierung. Die auch unter 4GL (4th Generation Language) bekannten Ansätze sollten die klassischen textuellen Sprachen in ähnlicher Weise „ablösen" wie damals C den Assembler. Das ist ja nicht wirklich geschehen, es hat sich nur der Einsatzbereich verlagert. Programme sollten einfach durch Zusammenstecken und Ausprobieren entstehen und die Software-

entwicklung damit müheloser und intuitiver gestalten. Stefan Schiffer beschreibt Geschichte, Grundlagen, Klassifizierung, Potenziale und Grenzen jener Art der Programmierung, für die das visuelle Wahrnehmungssystem des Menschen unentbehrlich ist [1].

Heute ist man sich einig, dass der rein visuelle Ansatz für bestimmte Anwendungsbereiche und Zielgruppen gut passt, neben den Vorteilen aber auch Grenzen hat und deshalb konventionelle Programmierung nicht generell ablösen kann. Eine weitere Schlussfolgerung ist, dass vor dem Hintergrund der vielen etablierten Programmiersprachen und dem bestehendem Wissen der Erfolg eines neuen Paradigmas eher fraglich scheint. Die Zukunft wird vielmehr in heterogenen Sprachen gesehen, die Bild und Text zusammenfügen, damit die Vorteile beider Welten nutzen und so bestehende Defizite ausgleichen. Das ist wiederum kongruent zur Natur heterogener CPS.

Der grafische Teil der heterogenen Sprache soll vor allem die kreative, parallele Denkweise des Programmierers unterstützen: Darstellen komplexer Zusammenhänge, Prozessabläufe, Vermitteln von Ideen, bildliche Kommunikation und ein Gesamtüberblick über Hard- und Software eines Cyber-Physical Systems (s. Abb. 2). Dank dieses „Hirn-Verstärkers" können sich Entwickler an die CPS-Komplexität heranwagen oder im Prototyping „verrückte" Ideen einfach und schnell implementieren und testen. Gleichzeitig wird damit der analytische Bereich des Gehirns entlastet, und er lässt sich viel effizienter für das ebenso wichtige logische Denken im Softwareentwicklungsprozess nutzen.



Bei CPS fließen Embedded Systems, Echtzeit-Systeme, verteilte Sensornetzwerke und Regelsysteme zu einem „System der Systeme" zusammen. Aufgrund der Nähe zu physikalischen Prozessen sind Dynamik, Parallelität und Timing zentrale Anforderungen an ein CPS. Das macht es interdisziplinär und so deutlich komplexer als ein klassisches Embedded-System (Abb. 2).

Datenfluss als virtuelles Blockschaltbild

Aus den verschiedenen visuellen Programmiermodellen wurde für den hier beschriebenen CPS-Turbo der strukturierte Datenfluss gewählt. Das hat mehrere Gründe:

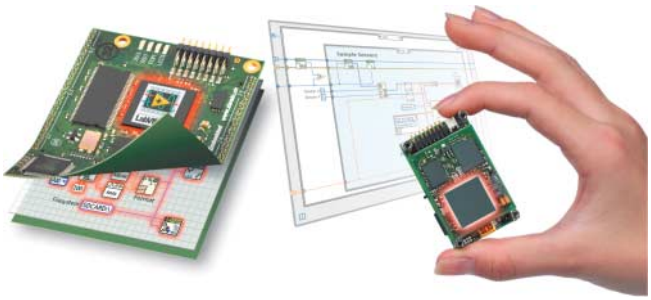
- Es geht um eine Weiterentwicklung der bekannten, funktionalen Programmierung.
- Auch Nicht-Embedded-Softwarespezialisten verstehen die grafische Notation auf Anhieb, da es sich im Wesentlichen um Blockschaltbilder handelt – der universellen und lösungsneutralen „Sprache" der Ingenieure und Wissenschaftler. Auch Fachleute aus dem Elektronikbereich werden im Datenfluss „ihr" Hardwareschema sofort wiedererkennen. Das hilft besonders als Kommunikationsinstrument im Team oder mit dem Kunden beim Prototyping.
- Das Datenflussmodell ist flexibel einsetzbar: von Design, Modellieren und Simulieren übers Implementieren bis hin zu Test und Validierung verteilter Systeme.
- Schwierige Konzepte der traditionellen Programmierung wie dynamische Datenstrukturen oder Variablen entfallen.
- Die Programmausführung erfolgt parallel, ohne dass der Entwickler dafür besondere Anweisungen vorsehen müsste – in Anwendungsbereichen wie numerischer Mathematik mit großen Datenmengen oder der Physik ein Plus.
- Strukturiertes Programmieren wird erleichtert, weil sich Programmteile dank klarer Schnittstellendefinition mühelos und, ohne auf Querbeziehungen zu achten, jederzeit trennen und verbinden lassen.
- Die Visualisierung der Programmausführung erhöht das Verständnis für komplexe Abläufe und erleichtert das Testen.

Cyber-Physical Systems sind „eins" mit ihrer Umgebung: Informationen aus physikalischen Vorgängen gewinnen, in Echtzeit berechnen, dynamische Regelprozesse steuern und per Netzwerk überall verfügbar machen (Abb. 1).

Im Folgenden wird eine konkrete Entwicklungsumgebung vorgestellt, die dieses Datenflussparadigma seit bald drei Jahrzehnten umsetzt und in mehreren Embedded-Anwendungsbereichen eingesetzt wird.

Das Spreadsheet für den CPS-Ingenieur

Die grafische Systemdesign-Plattform LabVIEW mit der visuellen Sprache G vereint die Stärken des theoretischen Datenflussmodells mit den praxisorientierten Prinzipien strukturierter und modularer Programmierung [2]. Sie weist damit zentrale Merkmale einer traditionellen, höheren Programmiersprache auf: einfache und zusammengesetzte Datentypen, statische Typisierung mit strenger Typenprüfung, hierarchische und polymorphe Operationen, Verzweigungen, Fallunterscheidungen, Sequenzen und Schleifen. Vor allem für CPS interessant sind



Eine mächtige, intuitive 4GL/DSL-Programmiersprache ist kompatibel zu Microcontrollern, DSPs und FPGAs, den Hardwarebausteinen von CPS (Abb. 3).

Timing und Synchronisierung als integraler Bestandteil der Sprache.

LabVIEW wurde von National Instruments (NI) mit dem Ziel lanciert, Wissenschaftlern und Ingenieuren ohne Programmiererfahrung den Zugang zu anspruchsvoller Applikationsentwicklung zu ermöglichen. In den meisten Fällen handelt es sich dabei um Mess-, Regel- oder Automatisierungsanwendungen auf Embedded-Hardware.

Die Entwickler sollten dank Blockschaltbild-Syntax eine intuitive Sicht auf diese Hardware erhalten und ihre Ideen umsetzen, ohne sich in zeitraubenden Low-Level-Details zu verlieren. Diesen Part übernimmt LabVIEW, weshalb Hardwarenähe, Echtzeit und Timing praktisch Hand in Hand mit der hohen Abstraktion der Sprache G gehen. Hier liegt der Hauptunterschied zu herkömmlichen Ansätzen im Embedded-Bereich. Der LabVIEW-Entwickler erhält über die Sprachelemente sofort Zugriff auf Techniken wie Interrupt-Handling, DMA-Transfers (Direct Memory Access), Registerprogrammierung, RTOS (Real-time Operating Systems) mit Multitasking, Memory-Management, Bootloader, Firmware-Update und Watchdog. Am Ende wird aus diesem Blockschaltbild Echtzeitcode erzeugt und auf einen Mikroprozessor, digitalen Signalprozessor oder FPGA (Field-Programmable Gate Array) geladen (s. Abb. 3). LabVIEW-Anwendungen sind weitgehend plattformunabhängig und lassen sich zwischen verschiedenen Prozessoren portieren. Ein Freiheitsgrad, der speziell im Rapid Prototyping von CPS große Vorteile hat.

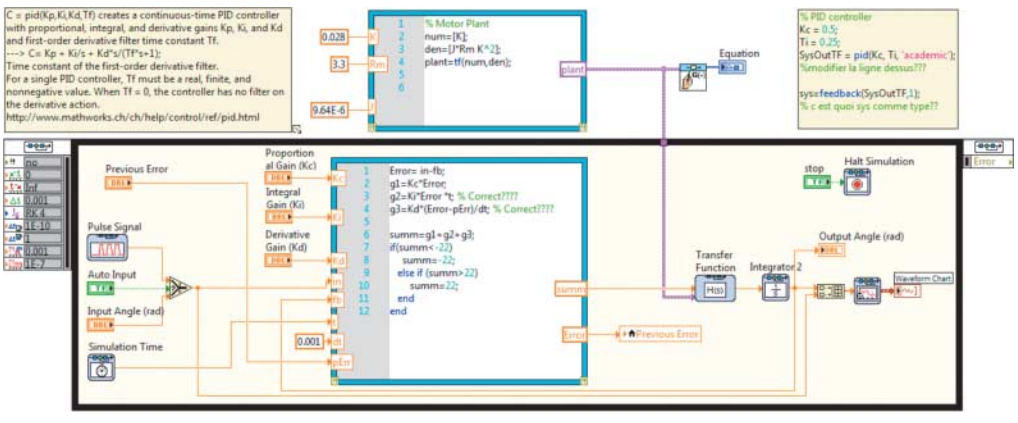
Als sogenannte DSL (Domain-Specific Language) ist LabVIEW eine fachspezifische Sprache und auf die Zielgruppe und Anwendungsbereiche der Technik ausgerichtet. Innerhalb dieses Fachbereichs erhält der Programmierer Unterstützung von Bibliotheken für Mathematik und Signalverarbeitung über Toolkits (u. a. Filter, Regler, Sound und Vibration, Vision, Motion) und Echtzeitsynchronisierung dezentraler Systeme bis zu modernen Bedienstrategien (Smartphone, Tablets) und Datenkommunikation (Cloud, Realtime-Ethernet). Hinzu kommt, dass ein LabVIEW-Diagramm dank Hintergrund-Compiler jederzeit ausführbar und mit grafischen Quellen und Senken sowie Profilern und Debugging-Instrumenten transparent testbar ist. Das kommt dem ursprünglichen Ideal visueller Programmierung („Zusammenstecken und Ausprobieren“) ziemlich nahe und weckt den Spieltrieb, was beim Rapid Prototyping und Cyber-Physical Systems das A und O ist.

Physikalische Programmiermodelle

Ausgehend vom strukturierten Datenfluss und seiner engen Beziehung zu Embedded-Hardware hat sich LabVIEW mittlerweile zu einer hybriden beziehungsweise heterogenen Sprache entwickelt, die mehrere Programmiermodelle unterstützt und standardisierte Datenschnittstellen anbietet. Das bekannteste LabVIEW zugrunde liegende Modell ist der strukturierte Datenfluss, bestehend aus virtuellen Instrumenten (VIs, Knoten), die zu einem Blockschaltbild verdrahtet (Kanten) werden. Eine Regeltechnik-Idee wiederum lässt sich in der üblichen Notation der Übertragungsfunktionen im Bildbereich effizient umsetzen und ausführen (s. Abb. 4).

Bei numerischen, rechenintensiven Algorithmen bietet sich Textmathematik (Matlab-Notation, Abb. 4 links) an. Sie lässt sich ebenso im LabVIEW-Diagramm einfügen wie Differentialgleichungen, die mit Model-Based Design [3] (Abb. 4 rechts) entworfen wurden. Statecharts (Abb. 7 links) vereinfachen vor allem komplizierte Ablauflogik zu einem einfachen, selbstdokumentierenden Systemdiagramm. Die C-Code-Schnittstelle (Abb. 6) schließlich ermöglicht das Einbinden bestehender C-Algorithmen oder den direkten Zugriff auf Mikroprozessoren bis auf Register- und Interrupt-Level. Dem Entwickler von Cyber-Physical Systems steht damit ein mächtiger Werkzeugkasten für unterschiedliche Aufgaben zur Verfügung.

Das für LabVIEW-Embedded-Anwendungen derzeit flexibelste Target ist das ZYNQ-SOC (System on Chip) mit Dual-Core-ARM9-Microcontroller und On-Chip-FPGA. Der LabVIEW-Support erfolgt über Linux, genauer gesagt über die für den Embedded-Bereich optimierte Ångström-Distribution. Als Bootloader dient (Das) „U-boot“, das nicht nur bei Embedded-Linux-Anwendungen weit verbreitet ist. Der Start und das Initialisieren des Kernels folgt dem System-V-Schema, mit dem sich zum Beispiel benutzerspezifisch Prozesse starten lassen. Über eine der Shells lässt sich das System mit externem Terminal, etwa PuTTY, einfach verwalten. Noch einfacher geht es mit



Für jede Aufgabe das richtige Programmiermodell im LabVIEW-Blockschaltbild, ob Datenfluss (links), Text-Mathematik (Mitte) oder Model-Based Design (rechts) zur Beschreibung physikalischer Prozesse (Abb. 4)

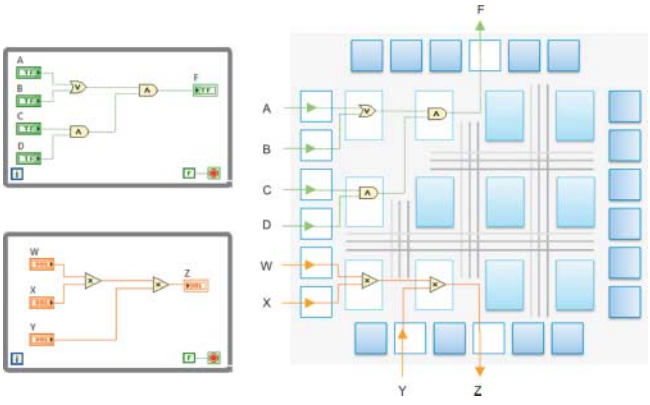
einem sich in den Webbrowser integrierten, grafischen Konfigurationswerkzeug. Daten werden anstelle von FTP über WebDAV ausgetauscht, einem auch von Dropbox genutzten Industriestandard für sichere Datenübertragung auf HTTP-Basis.

Entsprechend dem POSIX-Standard wird das LabVIEW-Diagramm eins zu eins auf das Linux-Betriebssystem abgebildet. Da lässt sich die „Spielwiese“ des Linux-Ökosystems nutzen, zum Beispiel beim Laden einer SQL-Datenbank über den Package-Manager opk. Zweitens steht mit der Busybox eine Art „Schweizer Armee-Taschenmesser“ für typische Embedded-Aufgaben zur Verfügung, von Filesystemzugriffen, Anzeigen der Systemzeit und kleinem DHCP-Client (Dynamic Host Configuration Protocol) bis zum Sleep-Modus und System-Reboot. Drittens erhält LabVIEW direkten Zugriff auf die Linux-Kommandozeile, womit sich direkt Systembefehle ausführen und so Filesystem- und User-Berechtigungen „live“ steuern lassen. Viertens steht mit Python eine mächtige Skriptsprache zur Verfügung. Fünftens kann LabVIEW mit TCP/IP über „localhost“ die Dienste weiterer Linux-Prozesse, sogenannte Deamons, anzapfen. Und sechstens besteht seit jeher die Möglichkeit, über LabVIEWs native C-API auf die Bibliotheken des Linux-Betriebssystems zuzugreifen. Diese sind entweder binär vorhanden oder lassen sich über Eclipse und Compiler/Linker aus C/C++-Sourcecode in eine Linux-Exe bauen, installieren und starten. Außerdem hat der versierte Linux-User jederzeit die Möglichkeit, den Kernel individuell zu konfigurieren und ihn neu zu kompilieren.

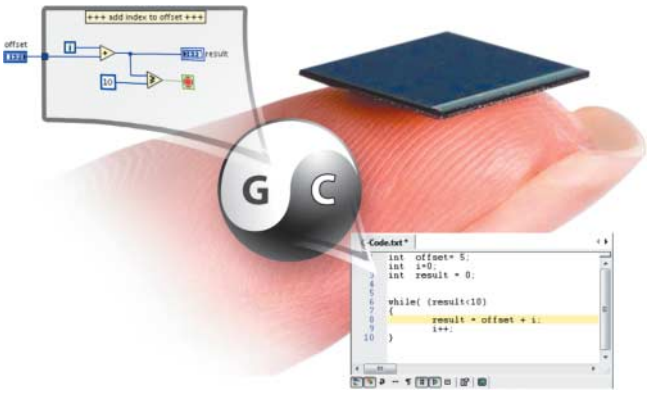
Beim Scheduling stehen sechs Schemata zur Verfügung. Mit *cron* lassen sich bis zu einer minütlichen Auflösung repetitive Tasks ausführen, zum Beispiel das Löschen von Logfiles oder regelmäßige E-Mail-Checks. Der CFS (Completely Fair Scheduler) dient vor allem zum Implementieren nicht zeitkritischer, aber trotzdem effizienter Work-Tasks. Werden Antwortzeiten in Millisekunden benötigt, lässt sich der Kernel entsprechend „preemptive“ konfigurieren. Dank LabVIEWs Multicore-Support kann der Programmierer grafische Tasks direkt einem Prozessorkern zuordnen. Bei zeitkritischen Tasks mit gefordertem Jitter zwischen 10…100 Mikrosekunden wird der Linux-Kernel mit *PREEMPT_RT* gepatcht. Für harte Echtzeit im einstelligen Mikro- oder sogar Nanosekunden-Bereich wird der FPGA genutzt, entweder über das variablenorientierte LabVIEW-Interface oder über eine C-API.

FPGAs mit der Maus programmieren

Ein FPGA ist eine im Kern mit Software rekonfigurierbare, parallel arbeitende Hardware. An ihn werden zeitkritische Aufgaben und I/O delegiert, um den Hauptprozessor zu entlasten. Dabei gibt es zwei entscheidende Eckdaten: die realisierbaren Funktionen und deren Timing. Bislang waren Spezialisten nötig, um FPGAs zu programmieren. Dank LabVIEW erhalten auch Ingenieure ohne diese Erfahrung Zugang zur mächtigen Technologie rekonfigurierbarer Logik (s. Abb. 5). Genauer betrachtet kommt die echte Parallelität der FPGAs dem LabVIEW-Datenflussparadigma am nächsten.



Die echte Parallelität der FPGAs kommt dem LabVIEW-Datenflussparadigma am nächsten (Abb. 5).



Der C-Code-Generator verbindet die abstrakte 4GL/DSL-Sprache LabVIEW mit der Welt von C, Echtzeitbetriebssystemen und Low-Power/-Cost-Boards (Abb. 6).

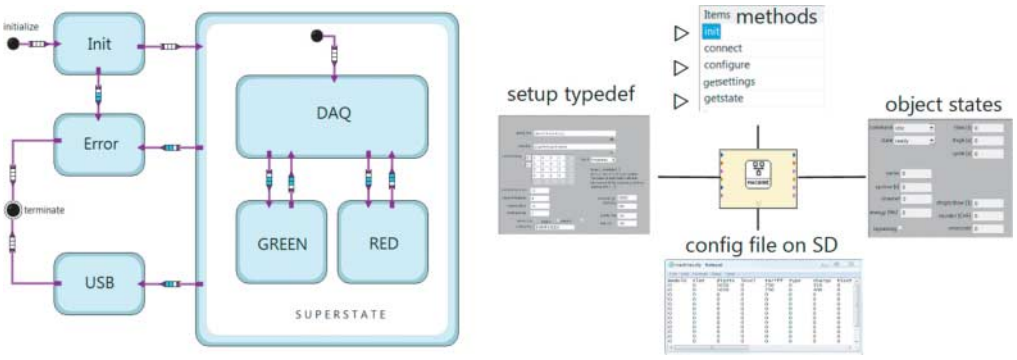
Über Funktionsblöcke lässt sich ein breites Spektrum analoger, digitaler und serieller Prozesssignale einbinden, verknüpfen und parallel vorverarbeiten, bevor sie in den Hauptprozessor weitergeleitet werden. Die Funktionen der abzubildenden Anwendung auf einem FPGA sind direkt begrenzt durch die zur Verfügung stehende Anzahl Gatter. Bekannte Kennwerte, welche Operation (Addition, Filter, FFT) wie viele Gatter benötigt, liefern deshalb wertvolle Entscheidungsgrundlagen bei der Auswahl der FPGA-Leistungsklasse. Beim Timing „denkt“ auch der grafisch orientierte Anwendungsprogrammierer in Ticks, dem kleinsten Zeitinkrement in der Größenordnung von Nanosekunden auf dem FPGA. Dabei ist definiert, wie viel Zeit bestimmte logische und mathematische Operationen und I/O-Zugriffe benötigen. Das gibt Richtwerte für das System-Timing.

Beim Softwaredesign gilt es, die Embedded-Anwendung sorgfältig auf Hauptprozessor und FPGA aufzuteilen. Der Hauptprozessor ist verantwortlich für die High-Level-Hauptfunktionen. Low-Level-Details wie zeitkritischer Code, digitale Filter, kombinatorische und sequenzielle Logik, Skalierungen, Fixed-Point- und Integer-Arithmetik werden auf den FPGA ausgelagert.

Das NI-LabVIEW-FPGA-Diagramm wird schließlich in VHDL-Code (Very High Speed Integrated Circuit Hardware Description Language) synthetisiert, mit den FPGA-Tools in eine Firmware/Bitfile kompiliert und in den FPGA geladen. Der gesamte Workflow wird abstrahiert und erfolgt auf Knopfdruck. Der Build mit Synthetisieren, Kompilieren und Flashen kann je nach Komplexität der FPGA-Anwendung mehrere Stunden dauern, was beim FPGA- und Hardwaredesign jedoch üblich ist. In diesem Moment wird Hardware „produziert“, weshalb der Vorgang direkt mit ähnlich zeitintensiven Routing-/Autorouting-Prozessen vergleichbar ist.

Grafisch auf 32-Bit-Microcontroller

In der Welt der Microcontroller, Mikroprozessoren und DSPs sind die imperative Sprache C und RTOS heute Quasi-Standard. Doch gerade das bei CPS so wichtige Timing fehlt im Sprach-



Zwei wirksame Elemente aus dem Software-Engineering sind die Statecharts (links) und Objektorientierung (rechts) – in LabVIEW grafisch (Abb. 7)

umfang von C und ist über Umwege wie Register-Programmierung, Interrupts oder Threads zu konstruieren. Das fügt eine künstliche Komplexität ein, statt sie zu reduzieren. Genau hier setzt der NI-ANSI-C-Code-Generator an (s. Abb. 6). Er ermöglicht dem Entwickler, seine Aufgaben komplett in der Sprache G als grafisches LabVIEW-Diagramm/Blockschaltbild umzusetzen und sich anschließend hardwareneutralen C-Code generieren zu lassen.

Die Embedded-Applikation umfasst Multitasking, Timing und Hardwarezugriffe direkt aus dem grafischen Code heraus. Darunter fallen beispielsweise analoges und digitales I/O, Embedded-Filesysteme, Kommunikation über Ethernet oder eine Multi-Touch-Bedienoberfläche. Damit setzt sich LabVIEW von ähnlichen Produkten, die nur „trockene" Anwendungslogik generieren können, ab. Der erzeugte Embedded-Applikations-C-Code wird mit dem Quellcode eines schlanken RT-Kernels verlinkt und mit gängigen Tools (Compiler, Linker, Loader) automatisch in eine echtzeitfähige Standalone-Firmware überführt.

Vom Flash-Speicher der Zielhardware startet die Anwendung direkt ohne Betriebssystem in Sekundenbruchteilen, geht in einen zuverlässigen 24/7-Betrieb über und ist gegen Einflüsse von außen weitgehend unempfindlich.

Software-Engineering wichtiger denn je

Dass auch Nicht-Softwareentwickler Anwendungen mit LabVIEW und G erstellen können, hat auch eine Schattenseite. Dieser Benutzergruppe sind die zentralen Konzepte des Software-Engineerings oft unbekannt, und es besteht die Gefahr, dass es G-Programmen an Architektur fehlt, sie organisch wachsen und so zu nicht mehr nachvollziehbarem Spaghetti-Code mutieren. LabVIEW ermutigt ja gerade, die Kreativität auszuleben, neue Ideen zu entwickeln und schnellstmöglich mit diesen zu „spielen". An diesem Punkt soll der Programmierer seinen Spieltrieb zähmen, sich auf seine logische Denkweise konzentrieren und den nötigen Rahmen schaffen.

Dazu gehören unter anderem eine gut durchdachte, skalierbare Softwarearchitektur in Form von State-Machines (s. Abb. 7 links), Modularisierung mit objektorientierten Ansätzen (s. Abb. 7 rechts), Versionskontrolle bei Arbeiten im Team, gezieltes Requirements Engineering und Unit- sowie Systemtests. Erst dann entfaltet LabVIEW sein volles Drehmoment für die Entwicklung von Cyber-Physical Systems.

Die in diesem Beitrag vorgestellte Entwicklungsmethode wird schon erfolgreich eingesetzt. Der Autor beschreibt beispielsweise an anderer Stelle [4] das Prototyping eines Messnetzwerks für dynamische Fahrzeug- und Reifenanalyse, das dem Cyber-Physical-Systems-Ansatz nahe kommt. In dieser verteilten Anwendung wird die individuelle Drehzahl der vier Räder, zum Beispiel von schnellen Roadstern, drahtlos gemessen, mit GPS-Daten synchronisiert und interner Positionsreferenz (Beschleunigungssensoren, Gyros) verknüpft, die Ergebnisse live auf einem 7"-TFT angezeigt, sie an den Tower gesendet und auf mobilem Medium abgespeichert. Interessant ist dabei, dass vier System-Tasks in ein und derselben Sprache auf vier unterschiedliche, dezentrale und synchronisierte Prozessorplattformen abgebildet wurden (ARM Core 1, ARM Core 2, FPGA, Mikroprozessor).

Weitere typische Einsatzgebiete sind ein Messnetzwerk 1000 Meter unter dem Meeresspiegel, auf der Schiene oder im 19"-Rack, in Analysen- und Medizintechnik ebenso wie in der Küche und im Abwasser. Dank geringem, skalierbarem Stromverbrauch ist die Methode auch für batterieunterstützte Aufgaben und für Mess-Handhelds geeignet. Ein erfolgreiches Beispiel dazu ist ein mobiles Gasanalysegerät in der Verteidigung. Der Anwendungsbereich erstreckt sich bis in die High-Tech-Branchen wie Automobil- und Luftfahrtindustrie, computerunterstützte Instrumentierung in der Medizin sowie Bioreaktorsteuerung. Auch bei Anforderungen wie der Steuerung des Partikelbeschleunigers im Large Hadron Collider (LHC) des CERN kam die Methode erfolgreich zum Einsatz. (ane)

Literatur

[1] Stefan Schiffer; Visuelle Programmierung – Grundlagen und Einsatzmöglichkeiten; Addison-Wesley Longman 1998

[2] Hugo A. Andrade; Software Synthesis from Dataflow Models for G and LabVIEW; National Instruments, Asilomar Conference on Signals, Systems and Computers; Nov. 1998

[3] Edward A. Lee et al.; A Model-Based Design Methodology for Cyber-Physical Systems with LabVIEW; IEEE Workshop on Design, Modeling and Evaluation of Cyber-Physical Systems 2011

[4] Marco Schmid; Radnabe als Dateneldorado/Messnetzwerk für dynamische Fahrzeug- und Reifenanalyse auf Basis von LabVIEW; Megalink Ausgabe 10/2013

Marco Schmid
(marco.schmid@schmid-elektronik.ch) ist in der Geschäftsleitung und im Verwaltungsrat eines mittelständischen, global tätigen Industriebetriebes für Elektronikentwicklung und -produktion, grafisch programmierbare Mikroprozessorlösungen und Messgeräte für die Schiene. Sein Fokus liegt auf der Hard- und Softwareentwicklung industrieller Embedded-Systeme – mit dem Hauptinteresse bei Software-Engineering, DSLs/4GLs und der Brücke zur „Low-Level-Welt" mit C-Code-Generatoren.