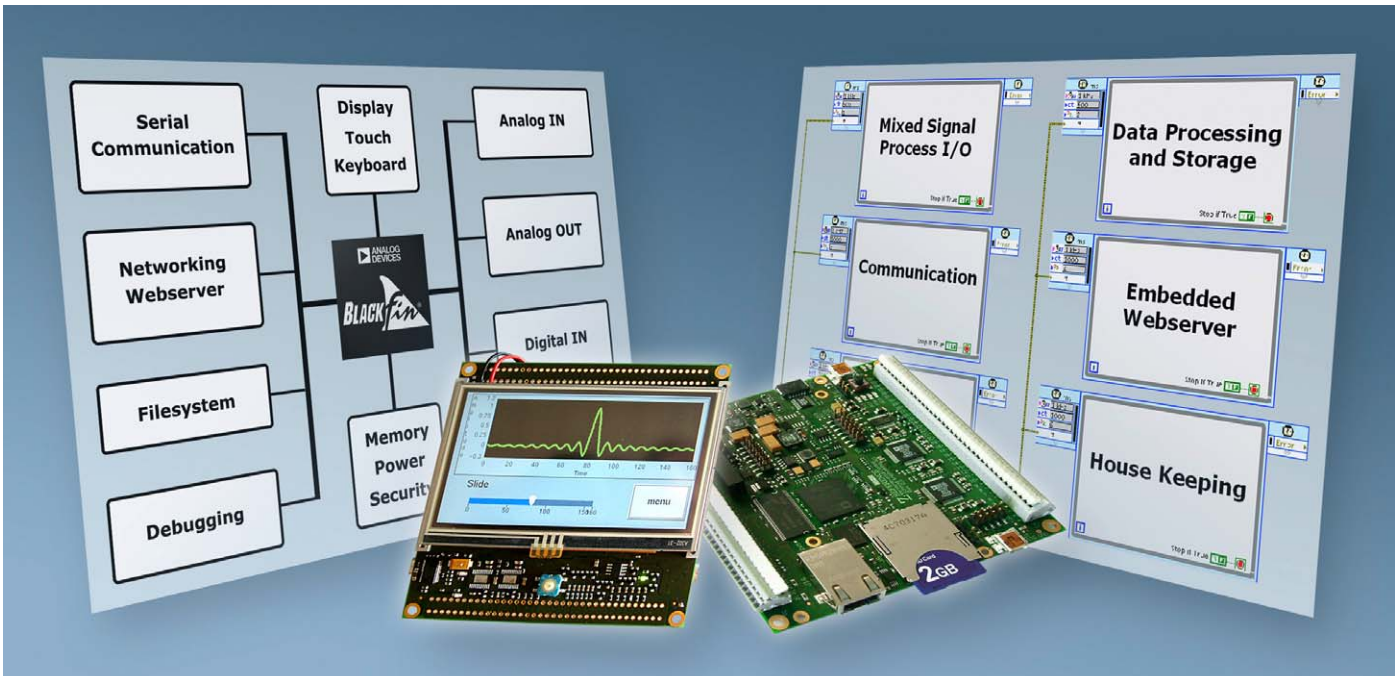

Bild 1: Schneller zu Ergebnissen mit der Systemlösung ZBrain und funktionellem Programmieren mit LabView Embedded.

Embedded Echtzeitanwendungen schneller entwickeln – mit LabView auf Mikroprozessoren

Mit funktionellem Programmieren auf Systemebene Ideen schnell umsetzen

Von Rapid Prototyping bis zur Serienentwicklung. Ob Panel-Rechner oder Messhandheld, intelligenter Sensor oder Tiefsee-Messnetzwerk, Outdoor-Feldeinsatz oder Webserver. Mit funktionellem Programmieren auf Systemebene lassen sich Ideen schnell und wirkungsvoll umsetzen. Die Programmierung erfolgt per Drag-and-Drop standardisierter Funktionsblöcke – bei voller Transparenz auf Prozessor und I/O.

Die zunehmende Komplexität und Leistung von Embedded Systemen stellt wachsende Anforderungen an die Anwendungsentwicklung. Deshalb verzögern sich viele Embedded Softwareprojekte oder scheitern sogar. Nicht zuletzt auch deshalb, weil diese Herausforderungen oft nur von Spezialisten gelöst werden können. Grafische Programmierung mit der domänenspezifischen Systemsprache National Instruments LabView ist in der Branche eine anerkannte Methode, um in kurzer Zeit sehr effektiv komplexe Probleme zu lösen *(Bild 1)*. Dieser Artikel beschreibt, wie komplette Mess- und Automatisierungsanwendungen vom Prozess-I/O übers GUI (siehe Glossar) bis zum Webserver auf der Basis von LabView auf Mikroprozessoren entworfen und implementiert werden können. Abstrakte Programmierung mit C-Code-Generierung und die Kombination von Echtzeitsoftware auf leistungsfähiger Low-Power-Hardware bietet die entscheidenden Vorteile, die Komplexität zu überblicken und Zeitpläne in den Entwicklungsphasen «Konzeptprüfung», «Prototyping» und «Serieentwicklung» einzuhalten.

AUTOR Marco Schmid
CEO Schmid Engineering

INFOS
Schmid Engineering AG
9542 Münchwilen
Tel. 0 71 969 35 90
www.schmid-engineering.ch
marco@schmid-engineering.ch

Funktionelles Programmieren auf Systemebene. Traditionell bringt ein Ingenieur seine Ideen oder Gedankenmodelle auf ein Blatt Papier, zum Beispiel in Form eines Blockschaltbildes. Anschliessend wird dieses Bild mit einer textbasierten Programmiersprache so codiert, dass es der Prozessor ausführen kann. Der Ansatz grafischer Systemprogrammierung beginnt einen Schritt früher. Das Gedankenmodell wird direkt als «live» Blockschaltbild auf dem Bildschirm grafisch entworfen. Daraus generiert das Entwicklungswerkzeug ausführbaren Prozessorcode über einen C-Code Generator und standardisierte Compiler/Linker *(Bild 8)*, der auf der Zielhardware live mit Prozess-I/O getestet wird.
Die Programmierung erfolgt per Drag-and-Drop standardisierter Funktionsblöcke und führt so ohne grossen Aufwand schnell zu verwertbaren Ergebnissen. Grundlage hierzu ist eine grosse Zahl unterschiedlichster virtueller Instrumente, die mit Verbindungslinien verdrahtet werden. Diese VIs reichen

von mathematischen Analyse- und Signalverarbeitungsalgorithmen bis zu komplexen Mess-, Steuer- und Regelfunktionen. Je nach Aufgabenstellung wählt der Entwickler das passende Programmiermodell *(Bild 3)* und fügt diese Elemente im grafischen Kontext zusammen. Für Signalverarbeitungen wie zum Beispiel digitale Filter eignet sich das klassische Datenflussmodell. Für Sortieralgorithmen eignet sich eher der textorientierte Ansatz. Ein PID-Regler oder eine Übertragungsfunktion hingegen sind mit Simulationsknoten am verständlichsten. Das übergeordnete Systemverhalten kann übersichtlich und selbstdokumentierend als Zustandsdiagramm visualisiert werden.

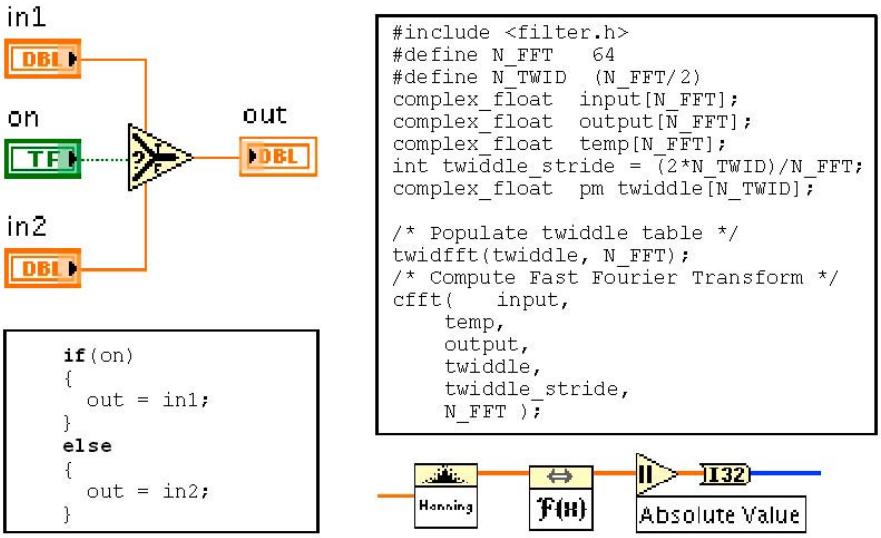

Bild 2: Der Vorteil grafischer Abstraktion gegenüber textbasierten Ansätzen zeigt sich speziell bei komplexen Aufgaben wie zum Beispiel einer Spektralanalyse (rechts).

Vom Prozess-I/O über GUI und Filesystem zum Webserver. Software für Embedded Mess- und Automatisierungssysteme gliedert sich in etwa folgende allgemeine Hauptfunktionen, die als parallele Prozesse unabhängig oder synchronisiert zueinander ablaufen:
Prozess-I/O: Punktuelles, kontinuierliches oder blockweises Erfassen von Sensordaten oder Steuern von Aktoren über analoge, digitale oder serielle Schnittstellen. Angesprochen wird die Hardware über plattformunabhängige Treiber-VIs, die unterlegte C-/Assemblerroutinen so abstrahieren, dass die Low-Level-Komplexität im Blockschaltbild verborgen bleibt. Da Prozess-I/O meistens parallel zu anderen Buszugriffen am Prozessor erfolgt, ist die sogenannte Threadsicherheit (engl. thread safety) Voraussetzung für lückenlosen Datentransfer im Multitaskingbetrieb. Prozess-I/O findet letztendlich auf Interrupt- und Registerebene statt und bedient sich Mechanismen wie DMA für maximalen Datendurchsatz.
Digitale Signalverarbeitung: Die meisten LabView-Analysefunktionen stehen in der «Embedded Welt» ebenfalls zur Verfügung, teilen sich aber die CPU-Ressourcen. Darüber hinaus fehlt meistens die FPU, also werden diese Operationen emuliert. Zur Optimierung bieten sich verschiedene Tricks wie zum Beispiel Shiften statt Multiplizieren, Verlegen von Code/Daten in schnelle Speicher, der Einsatz von Fixed-Point Arithmetik, eingebettete C-Funktionen, handoptimierte Prozessorbibliotheken oder Caching an. Zyklisch durchgeführte Benchmarks belegen, ob die unterlegte Hardware die Signalverarbeitungsfunktionen überhaupt in der geforderten Echtzeit ausführen kann.

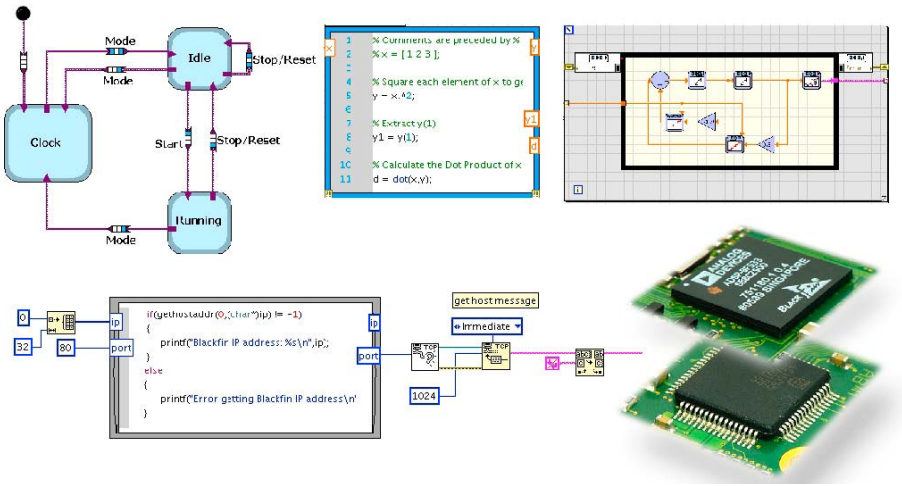

Bild 3: Verschiedene Programmiermodelle auf Systemlevel bei gleichzeitiger Transparenz auf Prozessor und Low-Level I/O.

Datenspeicherung: Die Applikation greift über standardisierte Dateifunktionen auf Messdaten, Konfigurationsdateien oder Bitmaps zu *(Bild 4)*. Im Gegensatz zum PC liegen sie auf Flashbausteinen oder mobilen Speichermedien wie SD-Karten. DMA wird genutzt, um Daten von LabView oder direkt von den Treibern auf die Speichermedien zu streamen. Kalibrationswerte oder Programmzustände werden häufig auf nichtflüchtigem Speicher wie FRAMs abgelegt.
Grafische Bedieneroberfläche: Das LabView-Frontpanel mit seinen fertigen Steuerungselementen, Schiebereglern oder Tachos eignet sich für Embedded-GUI-Konzeptstudien. In wenigen Stunden lässt sich so zum Beispiel ein Prozessmonitor entwerfen, realisieren und testen. Im fertigen Produkt sind meistens individuelle Grafiken mit CI-Elementen wie Logos gewünscht ➔

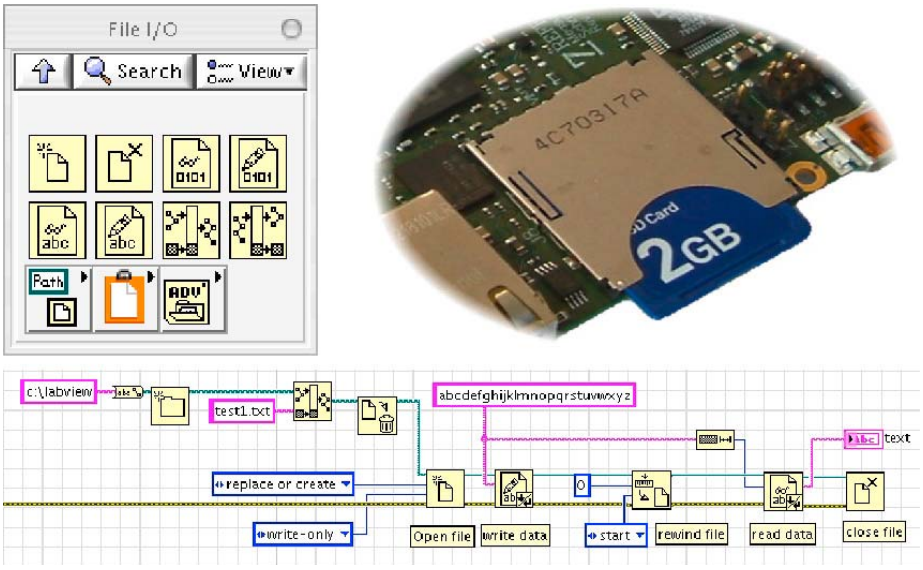

Bild 4: Embedded Filesysteme mit Drag-and-Drop standardisierter Funktionsblöcke.

Bild 5: Individuelle grafische Bedieneroberflächen und Touch auf dem PC entwerfen und ins Zielsystem laden.

(Bild 5). Eine gängige Methode dazu sind verschiedene Bitmaps, die dann entsprechend der Benutzerinteraktion geladen und auf dem Display positioniert werden. Interessante Effekte können mit der Overlaytechnik, also Bitmaps mit überlagerten Grafikprimitiven wie Linien und Text, erzielt werden.

Kommunikation: Neben klassischem Seriellen-I/O wie RS232, RS422 oder RS485 sind Ethernet, CAN oder USB gängige Kommunikationskanäle. Mächtige Mechanismen wie übers Netz zugreifbare (shared) Variablen basieren auf TCP/IP und sind eine einfache und schnelle Möglichkeit, zwei Zielsysteme via Ethernet zu verbinden. USB ist auch im Zusammenhang mit Direktzugriff vom PC auf das Embedded Speichermedium eine komfortable Lösung, zu den Daten zu gelangen.

Embedded Webserver: Auf der Basis von TCP/IP-Funktionsblöcken mit eingebettetem HTML-Code lässt sich sehr effizient ein Webserver implementieren (Bild 6). DHCP oder Static IP sowie die MAC-Adresse sind komfortabel im Werkzeug konfigurierbar.

Housekeeping: Fehlerbehandlung (Detektion und Korrektur), Online-Speicherüberwachung, Überprüfung der CPU-Auslastung, Überwachen des Multitaskingbetriebs, Watchdoghandling, Loggen von Ereignissen zusammen mit Zeitstempel aus der RTC sind zentrale Funktionen für stabilen und autarken 24/365-Betrieb.

Multitasking, Timing, Echtzeit. Anstelle des früher üblichen «Mainloop/Interrupt»-Gerüsts erfreut sich heute die Multitasking-Topologie grosser Beliebtheit, denn sie ist skalierbarer, übersichtlicher und damit wartungsfreundlicher. Idealerweise betreibt je ein Task eine im vorherigen Abschnitt beschriebene Hauptfunktion wie beispielsweise einen Webserver. Standardisierte LabView-Bordmittel sorgen für synchrones und asynchrones Timing, zum Beispiel mit Semaphoren, oder ermöglichen den gepufferten Datenaustausch zwischen den Tasks.

Auf der Mikroprozessorzielhardware läuft kein klassisches Betriebssystem, sondern ein präemptiver Multitaskingkernel. Dieses rudimentäre RTOS bietet die nötigen Ressourcen für Multitaskingbetrieb, I/O-Zugriff, Timing und Systemdienste. Bild 7 zeigt, wie diese Kerneldienste durch einfach anzuwendende grafische Strukturen oder Funktionsblöcke abstrahiert werden. Der «Timed-Loop» (Zeitgesteuerte Schleife) beispielsweise ist die grafische Repräsentation eines Tasks und lässt sich intuitiv konfigurieren: Timingsource, Ausführungsperiode, Priorität, Timeout und Phasenverschiebung. Ergänzend dazu bieten BSPs von den Boardherstellern erweiterte Echtzeitdienste oder Ereignisbehandlung. In der Laufzeitumgebung «untertunneln» diese Dienste LabView und erlauben Interrupt-Antwortzeiten im Mikrosekundenbereich. So lassen sich zum Beispiel digitale Triggersignale direkt mit weiteren Prozess-I/O-Ressourcen wie Analogwandlern verknüpfen.

Vom High-Level Blockschaltbild zum Low-Level Maschinencode. Das LabView-Blockschaltbild mit oder ohne Frontpanel definiert die Anwendung mit den Haupt- und den Unterfunktionen und abstrahiert den unterlegten Kernel, die Low-Level Treiber, Multitasking, Timing sowie Echtzeitdienste und die Mathematik. Der wichtigste Prozess besteht im automatisierten Generieren von plattformunabhängigem C-Code (Bild 8), dessen Qualität ausschlaggebend für performante und stabile Ausführung zur Laufzeit ist. Mapping-Dateien verbinden generische Definitionen wie zum Beispiel den Datentyp «Signed Integer» mit prozessorspezifischen Ressourcen, verknüpfen verschiedene Tools und verlinken den C-Code mit Echtzeitkernel, Bibliotheken sowie BSPs. Standardisierte IDEs der Prozessorhersteller erzeugen via Compiler, Linker und Loader Maschinencode, der als deterministische Firmware in den Flashspeicher des Zielsystems gebrannt und direkt von dort geladen wird, meistens im Zusammenspiel mit einem Bootloader.

Das Ziel ist hundertprozentige Konsistenz zwischen der grafischen Abstrahierung und dem eigentlichen Embedded-Code, sodass der autogenerierte C-Code nicht mehr von Hand geändert werden muss. Je nach Typ oder Komplexität der Anwendung oder Entwicklungsphase besteht die Möglichkeit, den C-Code-Generator entsprechend zu tunen:

- Softwareschalter (#defines) steuern Compiler und Linker
- Eliminierung von überflüssigem Code, zum Beispiel überflüssige Variablen
- Aktivieren und deaktivieren von Instruktions- oder Datacaching
- Generieren eines GUIs aus dem LabView-Frontpanel
- Aktivieren und Einstellen der Debugginghilfsmittel sowie eines Profilers
- Schutzmechanismen (zum Beispiel Division durch Null, ungültige Array Indizes)
- Wechsel zwischen kooperativem und präemptiven Multitasking
- Optimieren auf Codegrösse oder Ausführungsgeschwindigkeit
- Plazieren von Code und Daten auf verschiedene Memorybänke (zum Beispiel externes SDRAM oder schnelles on-chip-Memory)

Volle Transparenz auf Prozessor und I/O. Den Kern jedes Embedded Systems bildet der Mikrocontroller, Mikroprozessor oder Signalprozessor. Speicher, CPU-Clock und Energieverbrauch sind dabei die limitierenden Faktoren, weshalb die Programmierer bislang auf

textbasierte Sprachen der zweiten oder dritten Generation (Assembler, C/C++) setzten, um möglichst alles aus der CPU herauszuholen. Die Akzeptanz grafischer Tools, die der 4. Generation (4GL) angehören, ist bei Entwicklern von Embedded-Software deshalb direkt gekoppelt mit den Möglichkeiten, auch weiterhin transparenten Zugriff auf Ablauf und Timing der Software sowie auf die I/Os der Hardware zu haben. Und zwar bis hinunter auf Assembler- und Registerebene, indem zum Beispiel C-Code oder Assemblerroutinen im grafischen Kontext eingebettet werden können. Das schlägt Brücken zu Low-Level-Ressourcen wie Interrupthandling, Memory Mapping oder DMA-Transfers und erweitert LabView um typische Embedded Funktionen wie kundenspezifische Analoges- und Digitale-I/Os, Timer, Counter, GPIO, On-Chip und On-Board Memory (SRAM, SDRAM, Flash), Speichermedien (SD-Karten), RS232/RS485/RS422 Kommunikation, Batteriebetrieb, CPU Powermanagement, Sleep Modes, RTCs, Watchdogs und viele mehr. Über denselben Mechanismus lässt sich via standardisierte Bussysteme wie zum Beispiel SPORT, PPI, SPI, I2C, TWI, CAN, Ethernet oder parallelem Adress-/Data-/Controlbus dezentrale I/Os an den Prozessor anbinden.

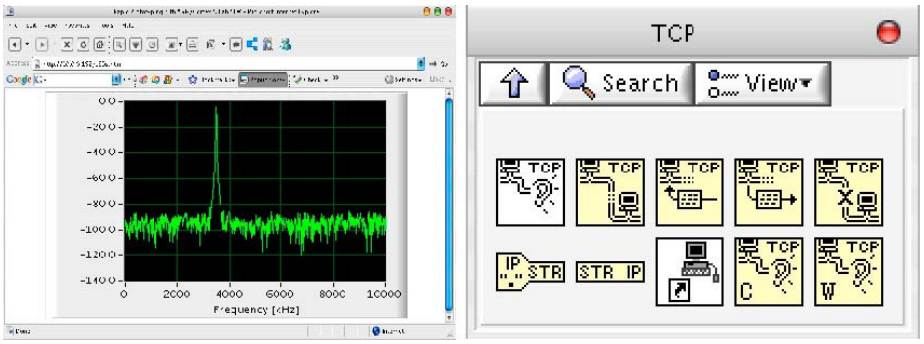

Bild 6: Daten über TCP/IP Dienste vernetzen – von der Shared Variable bis zum Webserver.

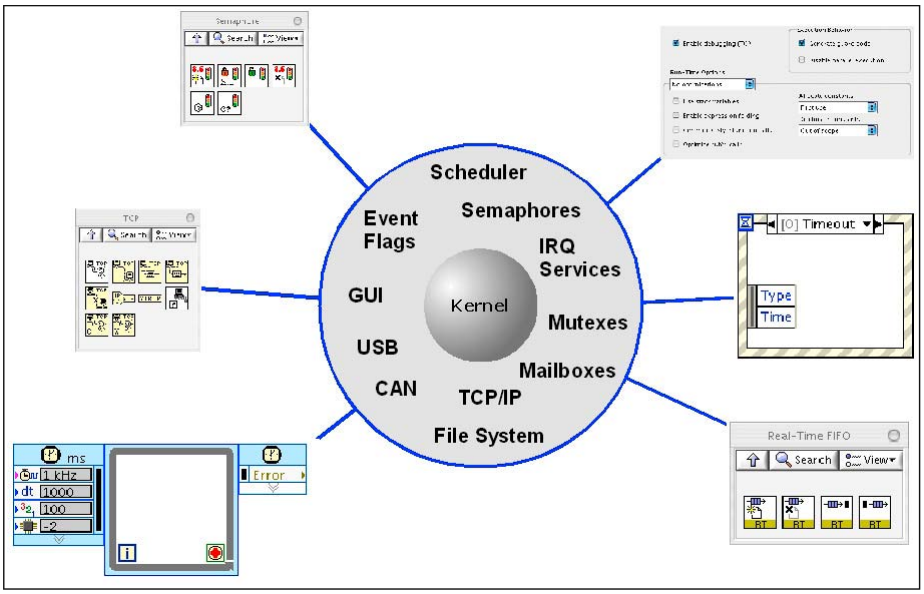

Bild 7: Multitasking und Kerneldienste des unterlegten RTOS werden von grafischen LabView Elementen abstrahiert.

Skalierbare Hardware wächst flexibel mit der Aufgabe. In der Praxis gehören Änderungen zum täglichen Brot. Mit National Instruments Konzept von rekonfigurierbarem I/O (cRIO/SBRIO) oder Schmid Engineerings skalierbarem ZBrain System lässt sich die Embedded Hardware schnell und einfach an diese Änderungen adaptieren, da die Hardwarefunktionen über plattformunabhängige Funktionsblöcke angesprochen werden können. Damit lässt sich auch die Wahl der CPU weitgehend ohne Einfluss auf die Software hinauszögern.
Standardisierte Referenzplattformen sind für bestimmte Anwendungsgebiete wie

- Mess- oder Regeltechnik
- Sound & Vibration
- Highspeed-Signalverarbeitung
- Mobile Messtechnik
- SPS-Automatisierung für rauhe Umgebungen optimiert.

Sie bieten unter anderem analoges I/O (von Hz über kHz bis MHz, 14 bis 24 Bit), digitales I/O, Encoder, PWM, Counter, Ethernet, USB, TFT, SD-Card Interfaces und intelligentes Powermanagement. Bei Bedarf lassen sich kundenspezifische Formfaktoren ableiten, indem Funktionen zugefügt oder entfernt werden.

Im System Fehler finden. Jede Anwendung enthält während der Entwicklung unweigerlich Fehler. Das gilt auch für die Verwendung grafischer Entwicklungswerkzeuge. Mit derartigen Werkzeugen lässt sich zwar die Komplexität einfacher abbilden und besser abstrahieren, das heisst aber nicht, dass die Anwendungen per se fehlerfrei sind. Damit Entwickler diese Softwarefehler beheben können, müssen ihnen leistungsfähige Werkzeuge zur Verfügung stehen, die effizientes Arbeiten und Eingreifen in den Betrieb der Anwendung ermöglichen:

- On-target JTAG Debugging mit Breakpoints, Pause, Singlestepping
- Setzen und Lesen von Variablen zur Laufzeit
- Simultandebuggen in LabView und generiertem C-Code bis auf die Assembler/Registerebene sowie Beobachten von Variablen mit minimal invasiven Sonden
- Ausgabe interner Grössen über den seriellen Port (Terminal) oder die Standardkonsole.

➔

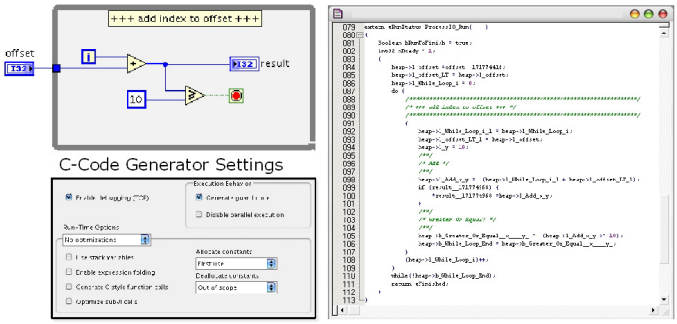

Bild 8: Gedankenmodelle als «live»-Blockschaltbild entwerfen (oben links) und über einen C-Code Generator (unten links) in echtzeitfähigen Prozessorcode (rechts) übersetzen lassen.

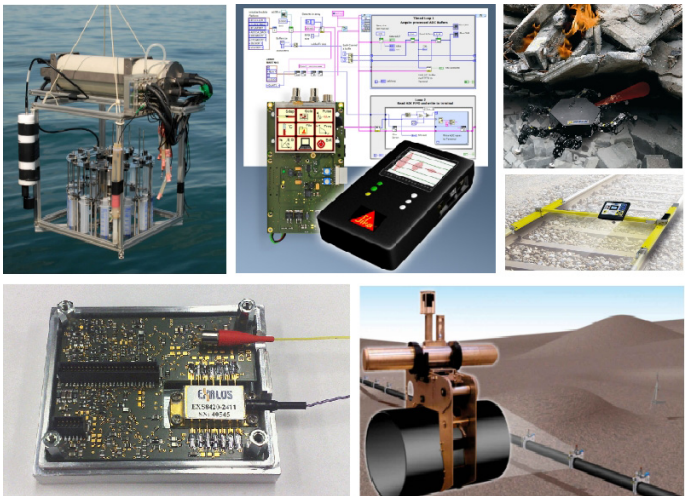

Bild 9: Verschiedene Projekte unterschiedlicher Komplexität erreichen den Markt dank LabView auf Mikroprozessoren in Rekordzeit.

- Hardware-In-The-Loop-Tests im Standalone Betrieb mit «live»-Zugriff auf Prozessor und I/O
- Debuggen des Multitaskingbetriebs direkt auf dem Zielsystem
- Timingkontrolle mit Oszilloskopen an I/O-Pins

Zeit- und kostensparend zum Embedded System. Bislang war die Entwicklung kompletter Embedded Systeme den Spezialisten vorbehalten. Die hier vorgestellte Lösung ermöglicht es nun auch Systemingenieuren, Prozessspezialisten und Domänenexperten, Mikroprozessortechnologie für ihre Embedded Anwendungen zu nutzen. Der Schlüssel dazu ist die Grafische Systemsprache LabView Embedded von National Instruments im Zusammenhang mit Autocodegenerierung als Ergänzung zu textbasierten Programmiersprachen. Zusammen mit Schmid Engineerings ZBrain System als einer Kombination produktiver Entwicklungstools mit Standardhard- und -software bieten sich jetzt zeit- und kostensparende Lösungwege, sich den kommenden Herausforderungen am Markt erfolgreich zu stellen *(Bild 9). (pm)* ■

Glossar

BSP	Board Support Packages. Auf eine Embedded Plattform angepasstes Softwarepaket mit Betriebssystem, Compiler und Treiber.
CAN	Controller Area Network. Asynchroner, serieller Feldbus.
CI	Corporate-Identity. Unternehmensidentität. Ist der abgestimmte Einsatz von Verhalten, Kommunikation und Erscheinungsbild nach innen und aussen.
CompactRIO	Compact Reconfigurable Input Output. Rekonfigurierbares Steuer-, Regel- und Erfassungssystem von National Instruments.
CPU	Central Processing Unit. Hauptprozessor, der die Programme in einem Computer ausführt.
DHCP	Dynamic Host Configuration Protocol. Ermöglicht die Zuweisung der Netzwerkkonfiguration an Geräte durch einen Server.
DMA	Direct Memory Access. Technik mit der Peripheriegeräte wie Netzwerkkarten ohne Umweg über die CPU direkt mit dem Arbeitsspeicher kommunizieren können.
FPU	Floating Point Unit. Gleitkommaeinheit in einer CPU, die mathematische Funktionen oder Gleitkommazahlen verarbeitet.
FRAMs	Ferroelectric Random Access Memory. Nichtflüchtiger elektronischer Speichertyp auf der Basis von Kristallen mit ferroelektrischen Eigenschaften.
GPIO	General Purpose Input/Output, frei programmierbare Ein- und Ausgänge.
GUI	Graphical User Interface, grafische Benutzerschnittstelle.
HTML	Hypertext Markup Language. Sprache zur Strukturierung von Inhalten wie Texten, Bildern und Hyperlinks in Dokumenten.
I2C	Inter-Integrated Circuit, meistens gesprochen als I-Quadrat-C oder englisch I-squared-C. Ein von Philips Semiconductors entwickelter serieller Datenbus.
IDE	Integrated Development Environment. Eine integrierte Entwicklungsumgebung ist ein Anwendungsprogramm zur Entwicklung von Software.
JTAG	Joint Test Action Group. Standard, der ein Verfahren zum Testen und Debuggen von elektronischer Hardware direkt in der Schaltung beschreibt.
MAC	Media Access Control. Standard für Netzwerkprotokolle und Bauteile, die regeln, wie sich mehrere Rechner das gemeinsam genutzte physikalische Übertragungsmedium teilen.
PPI	Parallel Peripheral Interface. Synchrone, 16-bit breite Schnittstelle des Blackfin Prozessors von Analog Devices, die sich zum direkten Anschluss von grafischen Displays und CMOS-Sensoren eignet.
PWM	Pulsweitenmodulation
RTC	Real Time Clock. Echtzeituhr
RTOS	Realtime Operating System. Ein Echtzeitbetriebssystem verfügt über spezielle Echtzeit-Funktionen für die Einhaltung von Zeitbedingungen und die Vorhersagbarkeit des Prozessverhaltens.
sbRIO	Single-Board Reconfigurable Input Output. CompactRIO ohne Gehäuse.
SD-Karten	Secure Digital Memory Card. Digitales Speichermedium, das nach dem Prinzip der Flash-Speicherung arbeitet.
SDRAM	Synchronous Dynamic Random Access Memory.
SPI	Serial Peripheral Interface. Ein von Motorola entwickeltes serielles Bussystem.
SPORT	Serial Port. Serielle Schnittstellen des Blackfin-Prozessors von Analog Devices.
SRAM	Static Random Access Memory. Statisches RAM.
TCP/IP	Transmission Control Protocol / Internet Protocol. Familie von Netzwerkprotokollen. Wird wegen ihrer grossen Bedeutung für das Internet auch als Internetprotokollfamilie bezeichnet.
TFT	Thin-film-Transistor. Dünnschichttransistor. Spezieller Feldeffekttransistor, auf dem TFT-Bildschirme basieren.
TWI	Two-wire Interface, siehe I2C.
USB	Universal Serial Bus. Serielles Bussystem.
VI	Virtuelles Instrument. Ein per Software definiertes Mess-, Steuer- und Regelsystem.