DESIGN & ELEKTRONIK MAKER TOOLS für Profis
KNOW-HOW FÜR ENTWICKLER

Rapid Application Development

Das IoT ist ohne eingebettete Systeme undenkbar. Erst kleine und kleinste Mikrocontroller mit intelligenten Kommunikationsschnittstellen und Anbindung ans Internet ermöglichen sinnvolles und flächendeckendes Einsammeln von Daten. Doch wie kompliziert ist der Einstieg in die Embedded-Welt? Dieser Artikel gibt Einblick, wie die »Arduino-Plattform« die Einstiegshürden für eingebettete Systeme dramatisch reduzieren kann.

PROF. DR. JÖRG WOLLERT

Kaffeemaschine, E-Bike, Smartphone und Heizungssteuerung haben heute eines gemeinsam: Die Intelligenz wird mit Software auf einem Mikrocontroller zur Ausführung gebracht. Die Software entscheidet über das Verhalten, die Datenauswertung, eventuell anfallende Wartungszyklen oder vielleicht sogar über das Ende der Lebensdauer. Selbst Tintenpatronen werden zunehmend intelligenter, sie wissen ob sie vom Originalhersteller sind oder ob es sich

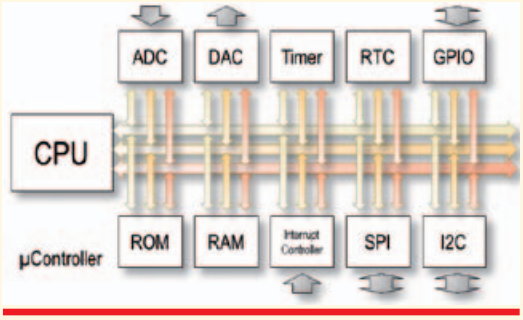

Bild 1: All-in-One – bei modernen Mikrocontrollern ist die gesamte Peripherie auf dem Die. *Bilder: Wollert*

um Plagiate handelt, sie kennen die Anzahl der Drucke und wissen wann sie leer sind. Und die Fitness-Industrie schafft eine kaum vorzustellende Wertschöpfungskette mit Apps für Smartphones, mit Sensoren, die den Vitalzustand des Probanden aufnehmen oder Geschwindigkeit und Trittfrequenz des Fahrrads aufzeichnen. Verbunden mit den GPS-Koordinaten der Geräte kann eine aussagefähige Analyse über den Fitness-Zustands eines Sportlers gemacht werden.

Zentrum der digitalen Welt sind häufig das Smartphone oder integrierte Modems, welche durch 3G- oder 4G-Schnittstellen einen uneingeschränkten Zugang in die weite Welt ermöglichen. Die Welt wird damit vernetzter, komplizierter und undurchsichtiger – gilt das auch für die Entwicklung der teilnehmenden Geräte? Für einen gewissen Grad sicherlich, auf der anderen Seite werden die Tools immer besser, um leistungsfähige Komponenten mit einem Minimum an Aufwand zu entwickeln.

Eingebettete Systeme

In der Regel sind eingebettete Systeme kompakte Geräte, die nicht unbedingt als »Computer« erkennbar sind, aber dennoch mit einer leistungsmäßig optimal skalierten Hardware ausgerüstet sind. Je nach Komplexität kann auf ein Betriebssystem verzichtet werden oder es werden spezielle Echtzeitbetriebssysteme oder Derivate von Android, Linux oder Windows verwendet.

Insgesamt kann man feststellen, dass eingebettete Systeme immer auf eine spezifische Aufgabe hin optimiert sind. Das spiegelt sich in den Hardwareschnittstellen, den Kommunikationsschnittstellen, in der Benutzerschnittstelle und dem Betriebsverhalten wieder. Die Optimierung zeigt sich dann auch in den Systemkosten oder der Codegröße, dem benötigten Speicher und was auch sonst noch denkbar ist. Darüber hinaus spielt das Echtzeitverhalten bei Embedded-Systemen eine herausragende Rolle.

Während konventionelle Computersysteme aus einer Vielzahl unterschiedlichster Komponenten von der CPU über Speicher und Peripheriekomponenten bestehen, basieren eingebettete Systeme auf hoch integrierten Mikrocontrollern. Diese haben alle notwendigen Komponenten quasi auf dem Chip (*Bild 1*). Das hat viele Vorteile. Während bei konventionellen Computersystemen ein ausgeklügeltes Board-Design notwendig ist, kann der Mikrocontroller als monolitisches System mit Schnittstellen verstanden werden, wobei die Systemschnittstellen in der Regel unkritisch im Platinendesign sind. Der Aufbau eigener Systeme wird dadurch deutlich vereinfacht. Der Systemarchitekt muss lediglich einen Mikrocontroller mit den notwendigen Systemressourcen auswählen und auf ein akzeptables EMV-Verhalten achten, der Rest läuft gewissermaßen von allein.

Möchte man sogar auf das Design eigener Hardware verzichten, kann man auf eine große Anzahl von Lösungen zurückgreifen. Eine Vielzahl von Open-Source-Communities stellt sehr gut standardisierte Hard- und Softwarekonfigurationen zusammen, die den Einstieg in die Embedded-Welt einfach machen. Zwei Plattformen gewinnen hierbei zunehmend an Bedeutung. Das untere Ende der skalierbaren Embedded-Plattformen wird durch die Arduino-Systeme realisiert. Hier wird bewusst der Begriff der Arduino-Systeme gewählt, weil sich unter einer einheitlichen Entwicklungsumgebung verschiedenste skalierbare Mikrocontroller-Plattformen von 8-bit-CPUs mit wenigen MHz Taktfrequenz bis hin zu 32-bit-i586-kompatiblen Multicore-Systemen befindet. Für Arduino spricht:

- Heute ist keine Embedded-Plattform für IoT-Systeme preiswerter verfügbar. Für unter 3 € sind vollständige IoT-Systeme mit Kommunikationsschnittstelle realisierbar oder über den chinesischen Distributor der Wahl lieferbar.
- Die Entwicklungssystematik von Arduino ermöglicht die extrem einfache und ko-

stengünstige Realisierung eines Arduino-Derivates und damit die Nutzung der gesamten Entwicklungs-Toolchain.

- Es gibt derzeit keine Hardwarebasis, die eine umfangreichere und skalierbare Komponentenbasis mit einer standardisierten Hardwareschnittstelle (Shield) bietet. Das führt dazu, dass das Shield-Interface von fast allen µC-Herstellern als Standardschnittstelle für Development-Boards verwendet wird.
- Daraus, dass Arduino Open-Source ist, folgt als logische Konsequenz, Platinen-Layout und Software sind verfügbar und dürfen für eigene Anwendung verwendet werden. Als »Mutter aller Arduinos« kann der Arduino UNO angesehen werden. Dieser zeigt sehr deutlich die Philosophie von eingebetteten Systemen und die Einschränkungen, die selbst durch minimale Betriebssysteme in Kauf genommen werden müssen.

Arduino UNO ist ATmega 328p-Minimaldesign

Der Arduino UNO ist ein Minimaldesign rund um den ATmega 328p-Mikrocontroller. Die verfügbaren Hardwareports werden konfiguriert, ein USB-UART-Konverter ermöglicht die Kommunikation mit dem Hostsystem oder einem weiteren Computer. Der Konverter wird so verwendet, dass dieser auf der einen Seite als Programmieradapter und auf der anderen Seite als Kommunikationsschnittstelle verwendet werden kann.

Leuchtdioden zeigen die RX/TX-Kommunikation und die Betriebsbereitschaft an. Die Spannungsversorgung erfolgt entweder über die USB-Schnittstelle oder eine externe Energiequelle. Der ATmega-µController wird mit einem speziellen Bootloader und einer genau definierten Programmierung der drei programmierbaren Ports zum Arduino (*Bild 2*). Wegen dieses definierten Setups ist es so schön einfach, einen Arduino zu verwenden.

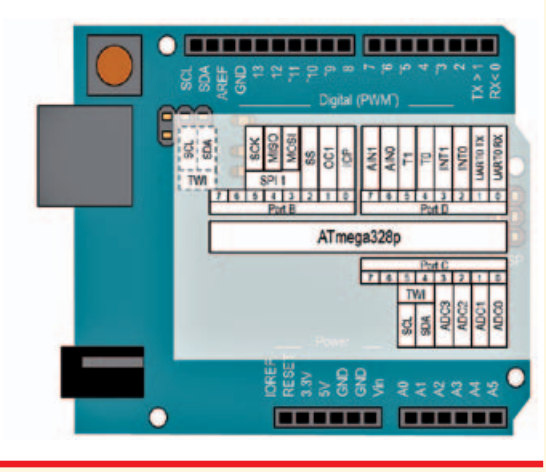

Bild 2: Nichts als ein spezifisches Systemsetup des ATmega 328p – Arduino UNO.

Während bei üblichen Development-Boards die spätere Anwendung im Vordergrund steht, gibt es beim ATmega ein wenn auch mit Einschränkungen behaftetes Muster-Design, das durch Bibliotheken die wesentlichen Funktionen unterstützt. Für den ATmega wird Port C immer als Analog-Eingang definiert, Port B unterstützt immer eine SPI-Schnittstelle und einige GPIOs (General Purpose Input Output) und Port D nutzt immer die UART-Schnittstelle und einige GPIOs. Alle Signale werden über PIN-Header zur Verfügung gestellt. Hierdurch entsteht quasi ein standardisiertes Bussystem, an dem Hardwareerweiterungen angesteckt werden können. Erweiterungsplatinen, die den Konventionen folgen, werden in der Arduino-Welt Shield genannt. Neben der Hardware gehört es für ein Shield zum guten Ton, eine 100 % Arduino kompatible Softwarebibliothek zur Verfügung zu stellen, so

dass ein Shield auf allen Systemen der Arduino-Familie verwendet werden kann.

Aktuell gibt es eine unvorstellbare große Anzahl von Shields, die nahezu alle Anwendungsfälle abdecken: Von Positionierungsaufgaben über GPS, Lage- und Rotationssensoren, Kommunikationsschnittstellen von kabelgebundenen Systemen wie Ethernet, CAN oder RS485 bis hin zu Funksystemen von Bluetooth und Bluetooth Smart über Enocean und Wi-Fi bis ZigBee. Auch rüsten die Chiphersteller ihre Development-Boards zunehmend mit Arduino-Shield-Schnittstellen aus, um unmittelbar an dem bereits existierenden Software- und Hardwarepool zu partizipieren. Damit wird die Wertschöpfungskette rund um die Arduino-Schnittstellen zum Standard für das Prototyping von Systemen. Kaum eine andere Umgebung kann eine derart leistungsfähige RAD-Umgebung bieten. Eine Luxusvariante des Arduino ist Intels Galileo 2 mit einer x86-Pentium-CPU und allem, was man für das IoT benötigt

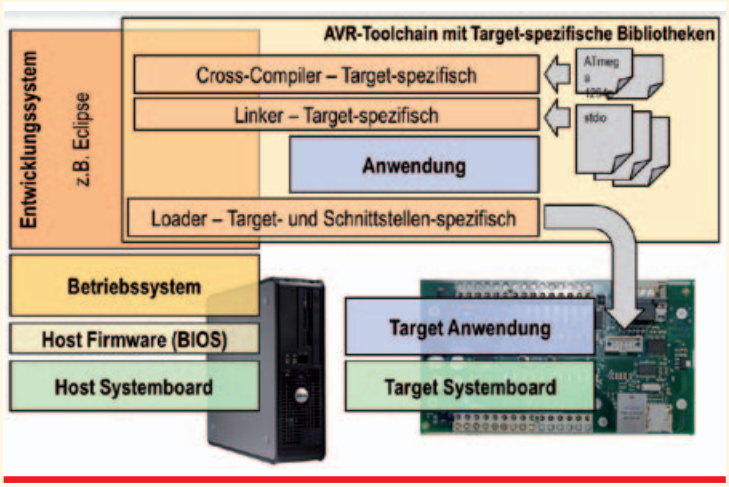

Bild 3: Cross-Development ist Voraussetzung für Embedded-Entwicklung.

Cross-Entwicklung eingebetteter Systeme

In den bisherigen Ausführungen lag der Fokus auf der Hardware von eingebetteten Systemen. Es hat sich gezeigt, dass gut definierte Hardware, unterstützt von geeigneten Bibliotheken, ideal ist für »Rapid Application Development«. Einen vergleichbaren Stellenwert hat die Softwareumgebung. Im Gegensatz zu einer klassischen »self-hosted-Entwicklung« erfolgt die Programmierung von eingebetteten Systemen immer über eine Cross-Entwicklungsplattform (*Bild 3*). Hierbei werden auf einem Hostsystem die Entwicklungswerkzeuge vom Editor über den Compiler bis hin zu Linker und Debugger gehalten. Der gesamte Coding-Prozess läuft auf dem Host, die erzeugte Software ist dort jedoch nicht lauffähig.

Zum Überprüfen der Lauffähigkeit muss entweder ein Emulator oder das Zielsystem verwendet werden. Bei der Entwicklung von Software für beispielsweise Smartphones ist die Nutzung eines Simulators selbstverständlich – bei der Anwendung konventioneller Embedded-Targets eher nicht. Über einen Target-spezifischen Treiber oder eine dedizierte Programmierschnittstelle kann die Software auf das Zielsystem übertragen werden. Mit einem Bootloader auf dem Target kommt das Programm schließlich zur Ausführung.

Eine der größten Herausforderung in diesem Cross-Entwicklungsprozess ist das Konfigurationsmanagement. Hierunter versteht man die »richtige« Versionierung der eingesetzten Tools und Bibliotheken für das spezifische Target. Das ist alles andere als trivial und führt zu einer Abhängigkeit zwischen Hostsystem und Target. Bei den Entwicklungssystemen kann man zwei Trends deutlich erkennen:

Microsoft hat das Visual Studio so weit geöffnet, dass durch Plugins das gesamte System von Drittanbietern als eigene Entwicklungsplattform genutzt werden kann. Vorausgesetzt man mag Windows als Host-Betriebssystem, lässt sich das gewohnte Programmiersystem auch für die Embedded-Entwicklung verwendet. Das AVR Studio ist beispielsweise eine derartige Plattform.

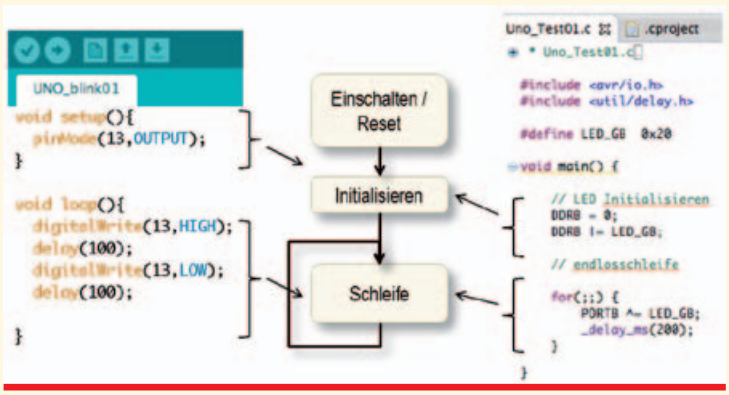

Bild 4: Arduino Sketch – Eigentlich nur eine Basisstruktur für einen Programmrumpf.

Der zweite Trend ist die Nutzung einer Java-basierten Entwicklungsumgebung. Diese ist vom Hostsystem unabhängig und kann damit auf Hosts mit unterschiedlichsten Betriebssystemen verwendet werden. Als typische Vertreter dieser Gattung gelten Eclipse und die Arduino-Workbench.

Egal für welche Entwicklungsumgebung man sich entscheidet, so ganz vom Betriebssystem unabhängig ist eine Entwicklung niemals. Immer sind spezifische hardwarenahe Werkzeuge notwendig, die für das jeweilige Betriebssystem adaptiert werden müssen. Häufig sind diese dann auch von den jeweiligen Versionen der Betriebssysteme abhängig. Und genau diese Zuordnung, welches Target mit welcher Konfiguration tatsächlich erstellt worden ist, stellt manchmal eine echte Herausforderung dar – oder man hat die Unterstützung von entsprechenden Managementsystemen.

Arduino-Softwareentwicklung

Anders ist es bei dem Arduino-Entwicklungssystem. Es basiert auf Java und kommt ohne herstellerspezifische Tools und Treiber aus. Die Verwaltung von Bibliotheken und Treibern für Programmieradapter ist integriert. Ebenfalls sind die unterschiedlichen Distributionen für die jeweiligen Betriebssysteme identisch – inklusive der Projektverwaltung. Bestenfalls sind die USB-Treiber für den auf dem Target verwendeten USB-Seriell-Adapter nachzuladen. Das macht diese Umgebung für ein »mal-eben«-Erstellen für Prototypen ideal.

Gegenüber konventioneller Programmierung abweichend ist der spezielle Arduino-Programmierframework, den man als Sketch bezeichnet. C-Programmierer sind es gewöhnt eine main()-Methode zu programmieren. Doch beim Arduino gibt es keine main. Statt dessen gibt es eine setup()-Methode, in welcher der gesamte Initialisierungscode ausgeführt werden soll. Hierzu gehört beispielsweise die Initialisierung der Ports und der verwendeten Variablen. In der zyklisch aufgerufenen loop()-Methode er-

folgt dann die Bearbeitung des eigentlichen Programmflusses. Formal ist die loop() eine in einer Endlosschleife aufgerufene Methode. In *Bild 4* ist der Unterschied zwischen einem Sketch und einer konventionellen C-Implementierung dargestellt.

Die Entwicklung von Embedded-Controllern mit Arduino als alleinige Lösung anzusehen ist sicherlich falsch. Die Debug-Möglichkeiten sind unzureichend, eine Versionsverwaltung für eigenen Code fehlt und die angebotenen Bibliotheken genügen bei weitem nicht professionellem Anspruch. Darüber hinaus ist der erzeugte Code alles andere als schlank und Laufzeit-optimiert. Für das RAD aber gibt es keine produktivere Lösung.

Gerade beim Umstieg von der klassischen Host-Programmierung auf die Entwicklung von Software für eingebettete Systeme sind die Hürden bei den klassischen Entwicklungsmodellen häufig so hoch, dass eine Systemlösung als schlanke, vom Betriebssystem freie Lösung, erst gar nicht in Betracht gezogen wird. Gerade bei IoT-Systemen spielt das eine dramatische Rolle. Der Griff zu einem Raspberry PI wird in aller Regel aufgrund der »gewohnten« Programmierung gemacht – nicht weil die Rechenleistung dieses Ein-Chip-Einplatinencomputers notwendig wäre. Hier bieten schlanke Systemlösungen ein erhebliches Reduktionspotential, so dass Kosten, Energieaufnahme und Footprint dramatisch reduziert werden können. Darüber hinaus ist eine schier unvorstellbar große Anzahl von Shields oder Modulen für Arduino mit funktionsfähiger Software verfügbar. Die Implementierung eines Wi-Fi-CAN-Adapters wird damit innerhalb eines Nachmittags möglich.

Doch wie so oft bekommt man nicht alles geschenkt. Üblicherweise ist der Bibliothekscode für reale Produkte nicht anwendbar. Es sind wirklich nur Prototypen. In *Bild 5* wird ein kleiner Überblick gegeben, wie sich die Codegröße in einem Arduino-Sketch im Vergleich zu einem gut geschriebenen C-Code verhält. Der Faktor 7 ist hier nicht geschönt, komplexere Programme weisen häufig einen noch höheren Reduktionsfaktor auf. Auch sind die Lizenzbedingungen, wenn man nur vorgefertigten Code verwendet weder produktfreundlich noch produktfähig. Aber sie bieten eine extrem gute Basis für schnelle und effektive Machbarkeitsstudien, Funktionsmuster und Prototypen.

Zusammenfassung

Vor einigen Jahren waren eingebettete Systeme ausschließlich Spezialisten vorbehalten. Die Masse der Programmierer hat sich auf die universellen Windows- und Linux-Plattformen gestürzt. Das hat sich geändert. Mit vorgefertigten Embedded-Plattformen von Arduino, über Beaglebone bis Raspberry PI haben sich Communities etabliert, die eine stressfreie und einfache Embedded Programmierung ermöglichen. Im Besonderen bei der Arduino-Entwicklungsumgebung und der Vielfalt der Arduino-Systeme zeigt sich wie eine Multiplattformentwicklung für Embedded-Systems möglich werden kann. Vorgefertigte Module und Bibliotheken mit einem reduzierten Framework ermöglichen effektives »Rapid Application Development«. Die Einstiegshürde für eingebettete Systeme wird dramatisch reduziert. Best Practices zeigen, dass die Entwicklung auch kleinster kostengünstiger Systeme möglich sind. (fr)

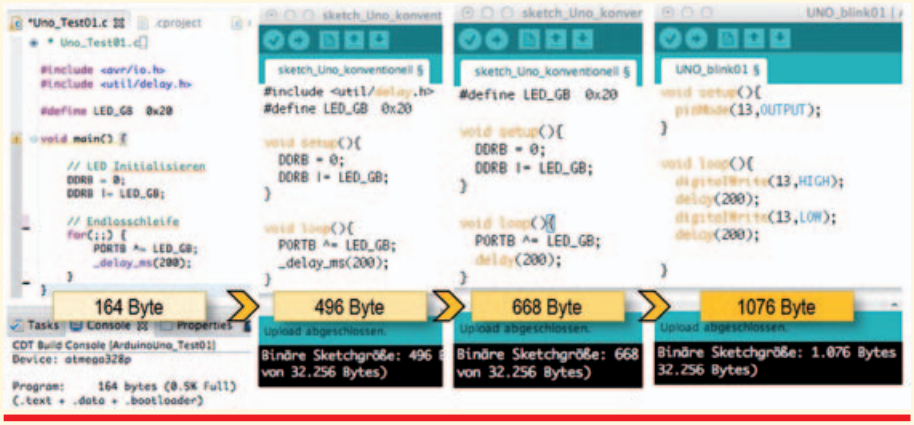

Bild 5: »Rapid Application Development« ist nicht codeoptimiert – aber entwicklungszeitoptimiert.

PROF DR.-ING. JÖRG WOLLERT

Professor für eingebettete Systeme und Mechatronik an der FH Aachen. Berät Unternehmen zu verteilten Automatisierungssystemen, eingebetteten Systemen (im besonderen mit Funktechnik) und zu Industrie 4.0.