MCUs werden immer komplexer, was tun?

Rolf Segger mit D&E-Chefredakteur Frank Riemenschneider.

Bild: DESIGN&ELEKTRONIK

Nachdem im Jahr 2016 die Halbleiterindustrie zu Wort kam, diskutierten im DESIGN&ELEKTRONIK TechTalk nunmehr die Firmengründer der vier führenden deutschen Tool-Hersteller mit Chefredakteur Frank Riemenschneider, wie die immer komplexer werdenden Mikrocontroller noch mit einem vertretbaren Aufwand beherrscht werden können.

Moore's Law sei es gedankt: Mehr Funktionen auf dem Chip, immer intelligentere Peripherie und immer mehr Rechenleistung aufgrund von 32-bit-CPUs kennzeichnen den Mikrocontroller-Markt im Jahr 2016.

Die wachsende Komplexität der Chips und somit auch der darauf laufenden Software führte jedoch auch zu Verzögerungen in Projekten, zu ausufernden Budgets und zu immer größeren Projektteams. Mit Thomas Bauch (PLS), Reinhard Keil (ARM), Stephan Lauterbach (Lauterbach) und Rolf Segger (Segger) diskutierte die DESIGN&ELEKTRONIK mit den vier deutschen Tool-Hersteller-Legenden, welche Wege die Software-Entwicklung zukünftig beschreiten wird. Auf der Grundlage von insgesamt acht Stunden Rohmaterial wurden die Ausführungen auf diese und die folgenden drei Ausgaben verteilt, den Anfang macht Rolf Segger, der 1992 die gleichnamige Firma gegründet hat.

Herr Segger, die Komplexität moderner Mikrocontroller nimmt immer weiter zu, die Frage ist, wie das für Entwickler noch zu beherrschen ist. Ein Ansatz ist, immer mehr Fertigkomponenten zu liefern. Ist das eine Strategie, die sinnvoll ist?

Mehr Fertigkomponenten von Halbleiterherstellern anzubieten, wie wir es schon seit Jahren sehen, ist grundsätzlich eine gute Sache für die Entwickler. Das Problem ist nur, daß man sich damit auf den Halbleiterhersteller festlegt, da der Code in der Regel in portabel ist oder die Lizenz nicht erlaubt diesen „mitzunehmen".

Eigene kostenlose Tools von Halbleiterherstellern »sind qualitativ eben wie sie sind«. Des Weiteren wollen sie die Kunden auf ihre Umgebung festnageln, auch durch Codebeispiele. Das kann doch niemand ernsthaft wollen?

Von einem Core auf einen anderen zu wechseln, ist für unsere Middleware nicht das Problem, da unterstützen wir 20 bis 30 Toolhersteller. Das Problem ist die Peripherie und natürlich eine gewisse Gewöhnung an die Tools. Was die Qualität von frei verfügbarer Software angeht, muss jeder für sich selbst entscheiden.

Nichtsdestotrotz wird es ja auch beim Einsatz von Profi-Tools immer schwieriger, eine MCU in einer optimierten Weise zu programmieren, nehmen Sie

mal einen Cortex-M7 mit seinen diversen Speichersystemen. Wie wird das denn unterstützt?

Als es noch keine Caches gab, war das Leben natürlich einfacher. Wir geben bei unserer Middleware Performance-Werte an zum Beispiel für eine AES-Verschlüsselung auf einem Cortex-M mit 200 MHz mit rund 2 MB/s. Dann haben die Kunden eine Richtschnur, und wenn sie davon grob abweichen, können sie sich bei uns melden. Ein Hauptproblem sind falsch konfigurierte Caches, das ist für viele Entwickler schwierig.

Die Datenblätter tragen ja auch nicht zur Vereinfachung bei…

Das stimmt, 2000 bis 3000 Seiten mit vielen Abkürzungen, wo eine Abkürzung mit drei anderen erklärt wird, helfen natürlich nicht weiter. Sie können studiert haben, kommen von der Uni und verstehen gar nichts.

Dazu kommt noch, dass immer noch Entwicklern der Overhead zum Beispiel für eine Abstrahierung der Peripherie-Register über Bibliotheken, die dann zusätzliches RAM benötigen, zu hoch ist. Was sagen Sie denen denn?

> *» Ein bisschen mehr RAM darf heute nicht mehr stören «*

Das bisschen RAM darf nicht mehr stören. Etwas mehr Code und mehr RAM ist bei heutigen Chips meines Erachtens nicht mehr so tragisch.

Ein Ansatz für eine vereinfachte Entwicklung sind modellbasierte Methoden. Ist das die Lösung?

Ich kann derzeit nicht erkennen, dass sich das in der Masse durchsetzt. Vor 10 Jahren hatte es mal geheißen, dass in 2 Jahren 80 Prozent aller Anwendungen UML-basiert sein werden. Eingetreten ist das nicht. Dass C-Programmierung verdrängt wird, sehe ich nicht. Davon abgesehen kann UML ja ohnehin nur auf der obersten Schicht funktionieren, wo noch viel C-Code drunter liegt. Unsere Idee ist, dass sich die Kunden nicht im Detail mit dem System auskennen müssen, indem wir beispielsweise die Systemkonfiguration richtig einstellen und über unsere Bibliotheken das Cachemanagement anbieten.

Wie sieht es Ihrer Ansicht nach mit der Akzeptanz von ARMs CMSIS aus?

CMSIS-Core hat sich bei den Halbleiter-Herstellern durchgesetzt. Sicher ist es gut, die Startup-Routine zu standardisieren – Einschalten der PLL, Initialisierung der Caches beispielsweise und ein sicheres Ankommen bei main(), wo ich anfangen kann zu programmieren.

Wären die Applikationsprozessoren Cortex-A mit Linux eine Alternative zu den komplexen MCUs?

Das hängt von Ihren Anforderungen ab. Wenn Sie preissensitive Waschmaschinen bauen und externes RAM und eine 6- oder 8-Schichten-Platine brauchen und dazu gegebenenfalls Sicherheitslücken in Linux stopfen müssen, ist das vielleicht doch nicht der richtige Ansatz. Bei größeren Stückzahlen funktioniert das aus Kostengründen nicht.

Nichtsdestotrotz müssen Sie manchem Kunden erklären, dass beispielsweise für ein GUI ein Cortex-M4 mit externem nicht gecachtem Speicher nicht dieselbe Rechenleistung liefern kann wie ein Cortex-A9. Ein vergleichbares API können Sie doch in entsprechenden Bibliotheken auch ohne Linux bekommen, beispielsweise TCP/IP, Filesystem, GUI, Security und USB. Cortex-A wird doch primär wegen der Rechenleistung genommen, wenn Cortex-M nicht mehr ausreicht. Zudem wird es bei Linux schwierig wenn es um harte Echtzeitanforderungen geht.

Wie sieht es denn mit Eclipse aus?

> *» Eclispe ist zu träge «*

Eclipse ist nicht primär für den Embedded Einsatz gemacht. Die verfügbaren Eclipse-Tools sind unterschiedlich gut. Generell ist Eclipse sehr leistungsfähig. Wir setzen es jedoch wir nicht ein, weil es zu träge ist und unsere Produktivität dadurch bremst. Auch ein PC der neuesten Generation mit 12 Cores neue Chips nützt Ihnen hier nichts, da viele Dinge bei Eclipse nicht parallelisiert sind. Es wird einfach nicht schneller.

Eine weitere Variante, nicht immer von Grund auf alles neu programmieren zu müssen, sind Embedded-OS. Wie sieht es denn mit deren Einsatz aus?

Der Einsatz von Embedded OS (RTOS + Middleware) macht meines Erachtens generell immer Sinn. Registerinitialisierung, Interrupt-Handling und Peripherie-Management bekommt der Programmierer frei Haus geliefert. In der Zukunft sehen wir natürlich immer mehr Kommunikationsfähigkeiten wie MQTT, aber auch Protokolle wie Secure FTP oder Dropbox für die Cloudanbindung.

Sehen Sie einen Durchbruch für neue Software-Entwicklungsmethoden?

Ich sehe, dass man mehr auf PCs entwickelt und eine Simulation macht.

Unabhängig davon bleibt erstmal alles wie es ist und die heutigen Tools werden besser, aber vom Grundsatz her sind keine revolutionären Änderungen in der Software-Entwicklung zu erwarten?

Das sehe ich so. Es wird weitestgehend bei C/C++-Programmierung bleiben, zumindest auf den unteren und mittleren Ebenen des Programmes. Auf oberen Ebenen, also der echten Anwendungsprogrammierung wie etwa Benutzerinterface und Ablaufsteuerung machen in vielen Fällen auch GUI-Builder oder UML ähnliche Tools Sinn. Das hängt allerdings stark von der eigentlichen Applikation ab. Der Embedded-Bereich ist und bleibt sehr vielschichtig, und die Anforderungen an Tools hängen stark vom Einsatzgebiet und Stückzahlen ab.

Herr Segger, vielen Dank für das Gespräch!

Rolf Segger
studierte Physik an der Universität Düsseldorf, bevor er 1992 die gleichnamige Firma in Hilden gründete. Im Jahr 2012 zog er sich aus dem operativen Geschäft zurück und verlegte seinen Dienstsitz als strategischer Berater auf die Bahamas. Die Motivation für die Gründung seiner Firma entstand aus der damaligen Tätigkeit als Software-Entwickler, in der er feststellte, dass in unterschiedlichen Projekten immer die gleichen Aufgaben anstanden – es folgte das erste Produkt, das RTOS. Internationale Bekanntheit erlangte die Firma, als sie für Mitsubishi einen Flash-Programmer entwickelte. Die Japaner hatten es geschafft, einen flash-programmierbaren Chip zu entwickeln, jedoch keinen Programmer.