Was sind Embedded Systems?

Der Versuch eines FAQ

Peter Balog

Was sind Embedded Systems ?
Embedded Systems (eingebettete Systeme), sind applikationsspezifische Mikrocomputerschaltungen mit applikationsspezifischer Software innerhalb einer zu steuernden (zu beeinflussenden) Umgebung.

Wo findet man *Embedded Systems* ?
In jeder „intelligenten" Anlage,- vom modernen Telefonapparat, Faxgerät, Videorekorder, Laserprinter über die Systeme zum Motormanagement in modernen Autos bis hin zu Hochleistungsanlagen im Bereich der Telekommunikation und Automatisierungstechnik. Sowohl die Geräte der modernen Meß-technik, als auch die Apparaturen im Medizinbereich wären ohne den Einsatz von leistungsfähigen *Embedded Systems* nicht denkbar.

Welche Hardware wird verwendet ?
Die Komplexität der Hardware von *Embedded Systems* richtet sich nach der spezifischen Anwendung. Von der einfachen Mikrocontrollerlösung (Singlechip) über Singleboardcomputer, bestückt mit Standardprozessoren, Standardperipheriebausteinen und spezifischen ASICs bis hin zu Multiboardsystemen basierend auf lose und eng gekoppelten Multimikroprozessormoduln, deren Rechnerleistung oft über denen eines Workstation-Clusters oder Mainframes liegt.

Nach welchen Gesichtspunkten erfolgt der Hardwareentwurf ?
Der Hardware-Designer von *Embedded Systems* hat die Aufgabe aus den definierten zeitlichen, „räumlichen" und sonstigen (firmenspezifischen) Randbedingungen die geeignete Mikrocomputerplattform zu finden. Das Zusammenspiel zwischen Prozessor(en), Standardperipherie, Speicher und anwendungsspezifischen (meist hochintegrierten *Standard-Cell* oder *Full-Custom*)-ASICs, die sogenannte *Glue-Logic*, erfolgt nach den Richtlinien für einen konstruktiven, streng synchronen Digitalentwurfs und wird (meist) in FPL (*Field Programmable Logic*) implementiert. Formale Spezifikation, Verifikation und Simulation unterstützen den Designer. Die „*Design for Testability*"-Strategien, gepaart mit der modernen Meßtechnik nehmen der Inbetriebnahme des Komplettsystems den Schrecken. Neben all der Digitaltechnik dürfen die analogen Aspekte, wie EMV, Stromversorgung und *Ground-Bounce*-Probleme, um nur einige zu nennen, nicht außer Acht gelassen werden.

Welche Software wird verwendet ?
Zufolge der Applikationsabhängigkeit steht i.A. keine „Standardsoftware" zur Verfügung,- sehr wohl gibt es jedoch Standardansätze im Softwaredesign.

Wie erfolgt die Softwareentwicklung für *Embedded Systems* ?
Die Softwareentwicklung läßt sich in die beiden Bereiche Systemsoftware und Applikationssoftware gliedern. Fast die gesamte Zeit der Softwareentwicklung wird zur Erstellung der Applikationssoftware benötigt,- diese wird plattform-unabhängig in einer höheren Programmiersprache (C, C++), basierend auf einem auf einem *Application Programming Interface* mit *Realtime*-Funktionalität (*preemptive multitasking with asynchronous IO-capability*), welches meist in Form einer Hochsprachenbibliothek vorliegt, entwickelt.

Lediglich 5-10% der Softwareentwicklungszeit entfallen auf die Systemprogrammierung, deren Aufgabe es ist, einen standard *Real-Time-Kernel* an die Hardwareplattform anzupassen und ein entsprechendes Laufzeit- und IO-System zu implementieren. Obwohl hier sehr hardwarenahe programmiert werden muß (*Device-Driver, Interrupt-Handler*), wird doch ein Großteil in einer höheren Programmiersprache (C) erledigt.

Was ist ein Real-Time-Kernel ?
Ein RTK ist der Kern eines Multitaskingbetriebssystems speziell für eingebettete Systeme (*preemptive multitasking with asynchronous IO-capability*). Der RTK stellt, plattformunabhängig, folgende Funktionen zur Verfügung:

- Task-Management
- Critical Ressource Management
- Task-Synchronization (Semaphores, Signals)
- Intertask-Communication (Mailboxes, Message-Passing, Pipes)
- IO-Management (IO-Driverinterface)
- Interrupt-Management (Interrupt-Handler-Interface, Interrupt-Tasks)
- Memory-Management (static, dynamic)
- Error-Handling (optional)
- Monitoring, Debugging (optional)

Für fast alle Mikroprozessoren und Mikrocontroller gibt es RTKs, die in kürzester Zeit an die gegebene Hardware angepaßt werden können. Zudem bieten RTK-Hersteller für gängige Peripheriebausteine *IO-Driver* und *Interrupt-Handler* (meist in C-Source) an,- der Systemprogrammierer muß meist nur die aktuelle Hardwareadresse des Chips in eine C-Header-Datei eintragen.

Wann sollte ein RTK verwendet werden ?
Immer dann, wenn im System mehr als eine Hardwarekomponente einen Interrupt generieren kann, oder wenn im System mehr als ein Eingang und ein Ausgang angesteuert werden muß.

Warum soll die Applikationssoftware nicht „direkt auf der Hardware" implementiert werden ?
Für Kleinstsysteme (1 Eingang, 1 Ausgang, kein Interrupt) ist die Anwendung eines RTKs sicher ein „Overkill". In allen anderen Systemen würde die Software-entwicklung mindestens doppelt so lange dauern, wobei die Verifikation des Systems ungleich schwieriger ist, wenn keine „ordnende Instanz" (RTK) verwendet wird.

Wird diese „ordnende Instanz", welche das „Interrupt-Monster" zähmt, innerhalb der Applikation implementiert, so hat man seinen eigenen RTK entwickelt ! Und es gibt eigentlich keinen Grund einen RTK selbst zu entwickeln, oder doch ?

Gibt es Gründe, die für die Eigenentwicklung eines RTKs sprechen ?
Die beste Antwort auf diese Frage gibt eine Overheadfolie der Probitas Corporation, die im Rahmen des „1995 Embedded System Design Symposiums" von HP gezeigt wurde:

Reasons to Write a Real-Time-Kernel

- You enjoy re-inventing the wheel
- You like chasing difficult bugs
- There's no rush to get your product out
- Porting software to new platforms is your idea of really living it up
- Your firm has the NIH(1) syndrome
- You want to be in the business of sellings RTKs yourself

(1) Not Invented Here

Warum gewinnt heute objekt-orientierter Softwareentwurf zunehmend an Bedeutung ?
Der objektorientierte Ansatz (Kapselung, Hierarchien, Vererbung) zwingt zu einer noch abstrakteren Spezifikation des Problems. Dadurch wird die „ordnende Instanz" im System mächtiger.

Wie sieht die Zeitaufteilung für den Entwurf des Gesamtsystems aus ?
Unter der Annahme, daß die Hardware auf Standard-Prozessoren und Peripherie-einheiten basiert und daß die Entwicklung etwaig verwendeter, hochintegrierter *Standard-Cell*- bzw. *Fullcustom*-ASICs (speziell im Hi-Speed-Telekommunikationsbereich) in die Hardwareentwicklungszeit nicht miteingerechnet wird, und daß außerdem die notwendige *Glue-Logic* zeitgemäß entworfen (synchron, HDL, Verifikation, Simulation) und implemetiert (FPL) wurde, teilt sich die Gesamtzeit für die Entwicklung von Embedded Systems (typisch) wie folgt auf: 25% Hardware, 5% Systemsoftware, 70% Applikationssoftware.□