

Embedded Systems

Einführung und Entwicklung

Sven Gregori
<gregori@hs-furtwangen.de>

Computer Networking
Fakultät Informatik
Hochschule Furtwangen

24. Mai 2006

Überblick

- 1 Einführung
- 2 Anatomie eines Embedded Systems
- 3 Aspekte die zu beachten sind
- 4 Development
- 5 Quellen und weiterführende Links

Überblick

- 1 Einführung
- 2 Anatomie eines Embedded Systems
- 3 Aspekte die zu beachten sind
- 4 Development
- 5 Quellen und weiterführende Links

Defintition Embedded System

- Special Purpose Computersystem
- für (eine) bestimmte Aufgabe konstruiert
- Teil des Geräts, das damit gesteuert wird
- Größe ist nicht entscheidend

Historisches

Apollo Guidance Computer

- MIT / Raytheon, 1961
- erster Einsatz: Apollo 11, 1969
- 4100 3-Input NOR Gatter ICs
- Core Rope Memory ROM
- Magnetic Core Memory RAM

Einsatzgebiete

- Mobiltelefone, PDAs
- Automobiltechnik
- Unterhaltungselektronik
- Netzwerktechnik
- Tastatur, Drucker, Scanner
- Kaffee-Automaten
- Fernbedienung
- ...

Überblick

- 1 Einführung
- 2 Anatomie eines Embedded Systems**
- 3 Aspekte die zu beachten sind
- 4 Development
- 5 Quellen und weiterführende Links

Beispiele

- Embedded Cronjob (Wecker)
→ LCD Anzeige, paar wenige Bedienelemente
- Router
→ Netzwerkverbindung, Status-LEDs
- Fahrkartenautomat
→ Display, Touchscreen, Datenbankbindung, ...
- Digitaler Satellitenreceiver
→ Video-Prozessor, IrDA Kommunikation, Menu, ...

Auswahl von Architekturen

- Embedded PC Systeme
 - PC/104
 - DIL/NetPC
- Microcontroller
 - 8051
 - 80C166
- ASIC, FPGA, CPLD

PC/104

- x86 Architektur
- 80386 bis P4, Centrino, ...
- Abmessung 90x96mm
- PC/104: ISA Bus
- PC/104-Plus: ISA Bus plus PCI Bus
- PCI-104: nur PCI Bus
- Aufeinanderstecken einzelner Module
(Mainboard, A/D, I/O, PCMCIA, ...)

DIL/NetPC

- Programmierbares Embedded Network System
- 32 Bit CPUs mit 50 bis 400MHz
(AMD 486, Intel StrongARM, Motorola ColdFire, ...)
- bis 64MB SDRAM
- bis 16MB Flash-ROM
- 10/100Mbit Ethernet
- Linux mit Embedded Webserver
- 55x23mm, 82x28mm, 82x33mm, 45x45mm

Microcontroller

- Single Chip Computer
- breites Architektur-Spektrum (4-32 Bit)
- Vereinigung von
 - CPU
 - Speicher (RAM, ROM)
 - I/O Controller
 - zusätzliche Komponenten (A/D Wandler, CAN Controller, ...)auf einem Chip
- in der Regel verzicht auf Cache

Microcontroller 8051

- Intel, 1980
- 8 Bit Architektur
- von diversen Firmen hergestellt
- heute noch mit am weitesten verbreitet
- 128 Byte On-Chip RAM
- 4KB On-Chip ROM
- ...

Microcontroller 80C166

- Siemens, 1986-1989
- 16 Bit Architektur
- 4-stufige Pipeline
- 16MB Adressraum
- 2KB interner RAM
- variabler interner Stack (64 - 2048 Byte)
- ...

FPGA

- Programmierbare Logikschaltung
- besitzt konfigurierbare Logikblöcke (CLP), Ein- und Ausgänge
- CLP besitzt Look-up Table, Register, MUX und I/O Matrix
- werden mit Hardwarebeschreibungssprachen programmiert
→ System direkt in Hardware implementiert
- z.B. eigene Gatterschaltung, Communication Controller (Ethernet, IrDA, USB, ...), CPUs, FPU, Microcontroller, ...
- OpenCores: Open Source IP-Cores Projekt

Überblick Betriebssysteme

- Garkeins
 - Anwendung direkt oberhalb Hardware
 - oder schon in Hardware implementiert (FPGA)
- Monitor
 - minimalste Art eines Betriebssystems
 - Speicherinhalt lesen, schreiben und ausführen
- General Purpose OSs
 - Linux, *BSD, Plan9, FreeDOS, ...
- Realtime OSs
 - QNX, VxWorks, Symbian OS, RTLinux, ...

Linux im Embedded Bereich

- LFS
- Eagle Linux
 - Anleitung für Linux LiveCDs from scratch
 - minimalistisches (statisches) Linux System
- uClinux
 - für Systeme ohne MMU
- emdebian
 - stripped down Debian System
 - 3 Jahre Work-in-Progress - Ergebnisse?
- RTLinux
 - nur Kernel-Module für Realtime-Anforderungen

Echtzeit Betriebssysteme

- reagieren vorhersagbar auf unvorhersagbare Ereignisse
- Kennziffern sind Interruptantwortzeiten und Kontextwechselzeit
- Problem: Caches, TLBs, Pipelines, ...
→ ermöglichen nur worst-case Abschätzung
- RTOS allein garantiert aber noch kein Echtzeitverhalten
→ Anwendung muss überlegt programmiert sein
- RTLinux: Module schalten sich vor eigentlichen Kernel
→ normaler Kernel läuft als Idle-Task im RT Kernel

Betriebssysteme - Zusammenfassung

- Betriebssystem je nach Anwendungsgebiet sinnvoll
- Realtime-fähige Systeme genauso
- General Purpose Systeme können Entwicklung erleichtern
 - gewohntes Umfeld
 - Support
- Linux: Keine wirklich brauchbare Distribution vorhanden
 - lass mich da gern eines besseren belehren!
 - ansonsten LFS
 - RTLinux Erweiterung wieder je nach Anwendung

Überblick

- 1 Einführung
- 2 Anatomie eines Embedded Systems
- 3 Aspekte die zu beachten sind**
- 4 Development
- 5 Quellen und weiterführende Links

Allgemeine Aspekte

- Zuverlässigkeit
- System muss Fehler entsprechend abfangen/behandeln
- Lebensdauer (Robustheit)
- System läuft meist dauerhaft und unbeaufsichtigt
- Einbau muss bedacht sein
(Abwärme, Energiezufuhr, Diagnose, ...)

verwendete Hardware

- Auf rotierende Komponenten generell verzichten
 - Festplatten → FlashDisk, (E(E))PROM
 - CD-/DVD-Laufwerke → USB-Stick
 - Lüfter → passiv gekühlte Komponenten
- passendes Dateisystem für Flash Speicher
 - Flash hat begrenzte Anzahl Schreibzyklen
 - `mv` verschiebt Datei erst in RAM
 - worst case: Dateiverlust bei Stromausfall
 - JFFS2
 - ggf. Ramdisk zu Hilfe nehmen

Einsatzumgebung

- Temperatur
→ Sahara, Antarktis, Weltall
- Erschütterung
→ deutsche Autobahnen
- Bedienbarkeit/Erreichbarkeit
→ Meeresgrund, Weltall

Sicherheitsaspekte

- Nicht von aussen gefährdet
 - h4x0rZ
 - User
 - Störungen
- Nicht nach aussen gefährdend
 - falsche Ausgabewerte abfangen (Herzschrittmacher)
 - falsches Timing verhindern (ABS)

Überblick

- 1 Einführung
- 2 Anatomie eines Embedded Systems
- 3 Aspekte die zu beachten sind
- 4 Development**
- 5 Quellen und weiterführende Links

Programmiersprachen Kategorien

Je nach Anforderung in Frage kommende Kategorien

- General Purpose Sprachen
 - eben für alle möglichen Anwendungen
 - z.B. Assembler, C, C++, Java, ...
- Hardwarebeschreibungssprachen
 - direkt Chips Programmieren (z.B. FPGAs)
 - z.B. VHDL, Verilog
- Synchroner Programmiersprachen
 - für reaktive Systeme (Avionik, Kernkraftwerke, ...)
 - z.B. Esterel, Lustre

Gängige Programmiersprachen I

- **Assembler**
 - nur bedingt sinnvoll
 - C Compiler liefert meist besseren Code
 - exotische Hardware oder Low Range Systeme
- **C**
 - kleiner Footprint (dietlibc)
 - Systemnah
 - inline Assembler (Zeitkritische Abschnitte)
 - ausgereifte, gut optimierende Compiler
 - allerdings übliche C-Probleme

Gängige Programmiersprachen I

- **Assembler**
 - nur bedingt sinnvoll
 - C Compiler liefert meist besseren Code
 - exotische Hardware oder Low Range Systeme
- **C**
 - kleiner Footprint (dietlibc)
 - Systemnah
 - inline Assembler (Zeitkritische Abschnitte)
 - ausgereifte, gut optimierende Compiler
 - allerdings übliche C-Probleme

Gängige Programmiersprachen II

- C++
 - prinzipiell wie bei C
 - zzgl. Option der Objektorientierung
 - Standard Template Library
 - bietet Exceptions Konzept
 - jedoch relativ großer Footprint
- EC++
 - C++ Dialekt fuer Embedded Bereich
 - reines Subset von C++ (keine Erweiterungen)
 - Verzicht auf Exceptions, Templates, Namespaces, STL, ...

Gängige Programmiersprachen II

- C++
 - prinzipiell wie bei C
 - zzgl. Option der Objektorientierung
 - Standard Template Library
 - bietet Exceptions Konzept
 - jedoch relativ großer Footprint
- EC++
 - C++ Dialekt fuer Embedded Bereich
 - reines Subset von C++ (keine Erweiterungen)
 - Verzicht auf Exceptions, Templates, Namespaces, STL, ...

Gängige Programmiersprachen III

- Ada
 - Versuch schwächen von C zu umgehen
 - Entwicklung vom DoD gefördert
 - vor allem für sicherheitskritische Systeme gedacht
 - typsichere Sprache
 - Runtime checkings
 - Free Content ISO Standard
- PEARL
 - Realtime Sprache
 - einstiger Gegenspieler von Ada

Gängige Programmiersprachen III

- Ada
 - Versuch schwächen von C zu umgehen
 - Entwicklung vom DoD gefördert
 - vor allem für sicherheitskritische Systeme gedacht
 - typsichere Sprache
 - Runtime checkings
 - Free Content ISO Standard
- PEARL
 - Realtime Sprache
 - einstiger Gegenspieler von Ada

Gängige Programmiersprachen IV

- Java
 - kleiner Footprint möglich (KVM, J2ME)
 - Hardwarenähe durch JNI möglich
 - Exception Konzept
 - Byte Code Verification
 - Portabel
 - Geschwindigkeit ist Gerücht
 - eigentliches Problem ist undefiniertes Schedulingverhalten
 - ..und ist halt ne Schlips-Sprache

Gängige Programmiersprachen - Zusammenfassung

- Hardware entscheidet indirekt die Sprache mit
- auch entscheidend, ob OS vorhanden ist oder nicht
- Assembler nur wenn es die einzige Möglichkeit ist
- Persönlich: C oder C++, je nach Paradigma-Vorliebe
- Java sollte nicht ausser Acht gelassen werden

C Programmierung μ C C167

- Spezieller C Compiler erforderlich (z.B Keil)
- Controllerspezifische Datentyp
 - bit: Bitvariable
 - bdata: definiert Bitadressierbare Variable (int, char)
 - sbit: Zugriff auf Bit im SFR oder bdata Variablen
 - sfr: definiert Variable im SFR (reg166.h)
- `void int_foo(void) interrupt 0x23 using reg_foo`

spezieller Fall: kein Betriebssystem

- Keine Umgebung
- `void main(void)` statt
`int main(int argc, char **argv)`
- `while (1)`, `pause()`, `sleep()`, ... statt `return`
- Programm wird z.B. über Monitor oder Bootloader gestartet

FPGA Programmierung

- Schaltung wird in Hardwarebeschreibungssprache definiert
 - VHDL
 - Verilog
 - (SystemC)
- Simulation der Schaltung
- generieren einer Netzliste
- physikalische Implementierung auf dem Chip
- beliebige Wiederholung

Debugging

- Cross-Debugger
- einfache Ausgabe (printf, printk, ...)
- digitale Ausgabe (Bit setzen, LED ansteuern)

Problem:

- Zeitbelastung z.B. bei Übertragung via RS232
- verändern von Echtzeitverhalten

Realtime Debugging

- In Circuit Emulation (ICE)
 - Systemspezifischer Chip (Bound-out Chip)
 - ersetzt eigentliche CPU
 - Möglichkeit direkt auf Register zuzugreifen
 - Problem: nur bis Taktraten von 50-100MHz machbar
- JTAG
 - IEEE 1149.1-1990 Norm
 - Chip enthält spezielle Testzellen an den Ein- und Ausgängen
 - aneinander gekettet (Schieberegister)
 - Werte können gesetzt und gelesen werden
 - Auch möglich um CPU/Controller zu programmieren

Zusammenfassung

- Wie gesagt: Special Purpose Systeme, d.h.
 - Architektur,
 - (Echtzeit-) Betriebssystem
 - Sprache für die Anwendung
 - ...

hängt von der geplanten Anwendung ab

- "Alles-könnende" Systeme nicht unbedingt von Vorteil
- gewünschte Echtzeit bedarf besondere Aspekte

Überblick

- 1 Einführung
- 2 Anatomie eines Embedded Systems
- 3 Aspekte die zu beachten sind
- 4 Development
- 5 Quellen und weiterführende Links

Quellen und weiterführende Links I

- Allgemein
 - WPV "Softwareentwicklung für Embedded Systems" bei Prof. Dr. R. Müller
 - Walter, Klaus-Dieter: Messen, Steuern, Regeln mit Linux. Franzis Verlag 2001
- Microcontroller
 - WPV "Microcontroller" bei Dipl.-Ing. T. Milcsevics
 - Schmitt, v. Wendorff, Westerholz: Embedded Control Architekturen. Hanser Verlag 1999 (Bibliothek: EU615)

Quellen und weiterführende Links II

DIL/NetPC <http://www.dilnetpc.com>

uclinux <http://www.uclinux.org/>

LFS <http://www.linuxfromscratch.org/>

Eagle Linux <http://www.safedesksolutions.com/eaglelinux/>

RTLlinux <http://www.fsmlabs.com/>

EC++ <http://www.caravan.net/ec2plus/>

OpenCores <http://www.opencores.org/>

whoohoo!

Fragen?