EMES: Eigenschaften mobiler und eingebetteter Systeme

Prozessoren für mobile und eingebettete Systeme: Die ARM-Architektur

Dr. Felix Salfner, Dr. Siegmar Sommer

Wintersemester 2010/2011

Prozessoren für mobile und eingebettete Systeme

- Anforderungen an Prozessoren sind anders als bei Desktop/Server-Systemen:
  - Niedriger Stromverbrauch wichtiger als Performance
  - Geringe Abwärme
  - Skalierbarkeit (gleicher Kern mit verschiedener Ausstattung)
  - Integrierte I/O-Komponenten ("System on a Chip")
  - Geringere Anforderungen an Kompatibilität
- Weit über 90% der Jahresproduktion an Prozessoren
- Breites Spektrum an Technologie und Leistungsfähigkeit
  - Von 4 Bit bis zu mehr als 64 Bit
  - Von n KB adressierbaren Speicher bis hin zu vielen TB

Prozessoren für mobile und eingebettete Systeme II

- Sehr große Vielfalt von Herstellern und Typen (kleine Auswahl):
  - Motorola: 68k-Architektur (CPU32)
  - Motorola und IBM: PowerPC
  - ARM: Verschiedene Versionen der ARM-Architektur und diverse Lizenznehmer (diese VL)
  - Hitachi: SH3, SH4 und SH5
  - NEC und andere: CPUs auf Basis der MIPS-Architektur
  - Microchip: PIC
  - Intel: i860, i960 und diverse andere (u.a. ARM)
  - Texas Instruments: MSP430 und andere
  - Infineon: C167 und andere
  - Atmel: AVR (VL vom 3.12.)
  - Viele weitere Architekturen und Lizenzproduktionen

Die ARM-Architektur

- Warum ARM (in dieser VL)?
  - Aktuell sehr bedeutend bei PDAs und eingebetteten Consumer-Geräten (80% im Mobilfunkbereich, 40% bei digitalen Kameras)
  - 2007: 98% der Mobiltelefone enthalten mindestens einen ARM-Prozessor
  - Bis Januar 2011 schon 15 Mrd ARM Cores verkauft
  - Von vielen Firmen lizensiert: Sony Walkman, Palm Pre, Telefone von Samsung / Sony / Siemens / Benq / HTC / Blackberry / Apple, Dell E-series, Garmin, TomTom, Nokia N900, Xerox-Drucker, Casio EX-1 Kamera, Canon Powershot, Microsoft Zune, Nintendo DSi, Amazon Kindle2, Apple iPod, ...

Was ist ARM?

- Eine Firma
  - Acorn RISC Machine
  - Advanced RISC Machine
  - Advanced RISC Machines, Ltd.
  - ARM Holdings
- Ein Prozessordesign (Inhalt dieser VL)
- Bezeichnung von Prozessorkernen

Geschichte und Ziele

- 1983-1985: Entwicklung der Acorn RISC Machine durch die Acorn, Ltd. unter Leitung von Steve Furber und Sophie (Roger) Wilson
  - Ziel war ein Prozessor für die Acorn Desktop-Computer
- Ab 1990: Weiterentwicklung und Vermarktung durch die ARM, Ltd.
- Entwicklung und Lizensierung von Designs, Implementation zu Mikrocontrollern und Prozessoren durch die Kunden

Designs: ARMv1 bis ARMv7 in verschiedenen Varianten
Implementationen: bspw. ARMv5TxM

Varianten

- T-Variante:
  Unterstützt Thumb-Modus (16-Bit Instruktionen, zur Laufzeit umschaltbar)

- M-Variante:
  Unterstützt Multiplikation zweier 32-Bit-Zahlen zu einer 64-Bit-Zahl (lange Multiplikation)

- E-Variante:
  Enhanced DSP — Instruktionen für Beschleunigung typischer DSP-Operationen

- J-Variante:
  Jazelle — bietet Soft- und Hardwarebeschleunigung für Java-Bytecode

Versionen 1 - 3

- Version 1 (1985) - Nie in einem kommerziellen Produkt verwendet, 26-Bit Adressraum, nur geringe Teilmenge des aktuellen Instruktionssatzes
- Version 2 (1986) - 16 32bit Register, Koprozessor-Unterstützung, 26-Bit Adressraum, Multiplikationsinstruktionen, 32-bit Datenbus
  - Extrem einfach gestalteter 32bit Prozessor - 30.000 Transistoren (Intel 286 - 134.000, Intel Nehalem - 383.000.000)
  - Minimaler Stromverbrauch
  - Kein Cache, kein Microcode; MMU, Grafik und I/O integriert
- Version 3 - 32-Bit Adressraum, erstmals Cache, zwei neue Prozessormodi, M-Variante

Versionen 4 - 6

- Version 4:
  zusätzlicher (System) Prozessormodus, Thumb-Modus (T-Variante)
- Version 5:
  Erweitert Version 4 um einige Instruktionen, verbessert Zusammenarbeit zwischen Thumb- und ARM-Modus, J- und E-Variante
- Version 6 (1992):
  Instruktionen für bessere Multimedia-Unterstützung (speziell SIMD-Instruktionen), verbesserte Interrupt-Behandlung, neue Statusbits, Thumb2 optional
  - Zusammenarbeit mit Apple (Newton PDA) und VSLI, DEC leitete StrongARM Architektur ab
  - Bei 233MHz nur 1W Leistungsaufnahme, noch immer nur 35.000 Transistoren

Version 7

- Version 7:
  Thumb2 obligatorisch, neue SIMD-Instruktionen (NEON - packed SIMD), DSP-Unterstützung, Profile für
  - Anwendungen (ARMv7-A): MMU, hohe Performance bei niedrigem Energieverbrauch, optimiert für Multitasking
  - Echtzeit (ARMv7-R): Geringe Latenz, Vorhersagbarkeit, geschützter Speicher
  - Microcontroller (ARMv7-M): Geringe Transistoranzahl, Vorhersagbarkeit, tief eingebettet
- Läuft auch unter der Bezeichnung Cortex

In dieser VL: Version 4 und aufwärts

Namensbildung für Implementationen

- Beginnen mit ARMv...
- ... gefolgt von der Version...
- ... und dann der Aufzählung der Varianten
- Besonderheit: Modelle ab Version 4 unterstützen in der Regel M, so daß dies nicht mehr aufgeführt wird. Ausnahmen ohne M werden als xM gekennzeichnet

Beispiel: ARMv5TxM

- Besonderheit: Bezeichnungen ohne "v" bezeichnen Implementationen, Zahl ist nicht direkt mit Architekturversion korreliert

Beispiel: ARM7TDMI Kern implementiert ARM Architektur v4T (siehe Wikipedia)

RISC-Design

- Bessere Bezeichnung ist 'Load/Store-Architektur' - keine anderen Möglichkeiten für Speicherzugriff, Logik für Verzögerungen minimal
- Vertreter: Alpha, ARM, AVR, MIPS, PowerPC, Sparc
- Instruktionen fester Länge, primär Registeroperationen
- 'Single clock throughput' bei hoher Frequenz
- Verringerung der einzelnen Auführungszeit, nicht der Anzahl der verfügbaren Instruktionen !
- Einfache Adressierungsmodi und wenige Datentypen in Hardware
- Guter Kandidat für Harvard-Architektur (Instruktionen und Daten getrennt, separate Caches)
- Moderne CISC-Prozessoren arbeiten häufig als RISC-Hardware auf Microcode-Ebene
- RISC übernimmt CISC-Ansätze (Bsp.: Cortex A8 Branch Prediktor)

Parameter der ARM-RISC-Architektur

- 32-Bit Adress- und Datenbus
- Instruktionen mit fester Länge (leichtere Dekodierung und Pipelining), die meisten in einem Takt ausführbar
- Diverse Prozessor-Modi
- 37 Register
- Datentypen:
  - Wort (32 Bit), auf 4-Byte-Grenzen aligned
  - Halbwort (16 Bit), auf 2-Byte-Grenzen aligned
  - Byte (8 Bit)
- Instruktionen
  - 54 ARM-Instruktionen: genau ein Wort breit
  - 38 Thumb-Instruktionen: genau ein Halbwort breit
  - Datenverarbeitende Operationen arbeiten alle auf Wörtern

Prozessor-Modi I

Prozessor-modus	Kurzform	Beschreibung	Exception
User	usr	Normale Programmausführung	-
System	sys	Privilegierte Programmausführung	-
Supervisor	svc	Privilegierter Exception-Modus	Software Interrupt
Abort	abt		Data Abort, Prefetch Abort
Undefined	und	Fängt undefinierte Instruktionen und unbehandelte Koprozessor-Instruktionen ab	Undefined instruction
IRQ	irq	Wird für die Interruptbehandlung benutzt	IRQ
FIQ	fiq	Wird für die Interruptbehandlung benutzt (Fast Interrupt)	FIQ

Prozessor-Modi II

- Alle Modi außer User-Mode sind privilegiert
- Alle Modi bis auf User und System sind Exception-Modi
- Programme laufen im User-Mode, bis Exception auftritt, die im betreffenden Modus behandelt wird
- Besonderheiten:
  - System-Mode: Arbeitet auf Registern des User-Mode
  - FIQ-Modus: Hat einen größeren eigenen Registersatz (neben R13, R14, R15 und CPSR auch R8-R12)

Register I

- 37 Register
- R0-R14 meist beliebig benutzbar
- Besondere Funktionen und Register:
  - R15: PC
  - R14: Link Register (Exception-Mode, BL-Instruktion)
  - R13: Stackpointer in Exception-Mode
  - CPSR: Current Program Status Register
  - SPSR: Saved Program Status Register

Register II

User	System	Supervisor	Abort	Undefined	IRQ	FIQ
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

Status-Register

31	30	29	28	27	26 … 8	7	6	5	4	3	2	1	0
N	Z	C	V	Q	Undefiniert	I	F	T	M	M	M	M	M

- N (Negativ), Z (Null), C (Übertrag) und V (Überlauf) sind Condition-Flags
- Q: Nur in E-Variante als Überlauf für erweiterte DSP-Instruktionen
- I und F: Für Interrupt- bzw. Fast Interrupt-Mode
- T: Thumb-Mode
- M kodiert den Modus der CPU

Ausnahmen (Exceptions) I

- Wenn eine Exception auftritt, wird folgender Code ausgeführt:

```
R14_<exception_mode> = return link
SPSR_<exception mode> = CPSR
CPSR[4:0] = exception mode number
CPSR[5] = 0           ; Execute in ARM state
If <exception_mode> = RESET or FIQ
  CPSR[6] = 1 ; Disable FIQs
CPSR[7] = 1           ; Disable IRQs
PC = exception vector address
```

Ausnahmen (Exceptions) II

Exception Vector:

Exception	**Modus**	**EV-Adresse**	**EV-Adresse (High Vector)**
Reset	Supervisor	0x00000000	0xFFFF0000
Undefinierte Instruktion	Undefined	0x00000004	0xFFFF0004
Software Interrupt (SWI)	Supervisor	0x00000008	0xFFFF0008
Prefetch Abort (Speicher signalisiert Abbruch beim Holen einer Instruktion)	Abort	0x0000000C	0xFFFF000C
Data Abort (Speicher signalisiert Abbruch beim Holen eines Datenwortes)	Abort	0x00000010	0xFFFF0010
IRQ (interrupt)	IRQ	0x00000018	0xFFFF0018
FIQ (fast interrupt)	FIQ	0x0000001C	0xFFFF001C

Ausnahmen (Exceptions) III

Priorisierung:

Priorität		Exception
höchste	1	Reset
	2	Data Abort
	3	FIQ
	4	IRQ
	5	Prefetch Abort
niedrigste	6	Undefined Instruction
	6	SWI

Instruktionsmodi

- ARM-Mode
  - Standard-Modus
  - Einzig möglicher Modus für Ausnahme-Behandlung
  - 54 Instruktionen
- Thumb-Modus
  - Höhere Code-Dichte durch 16-Bit-Instruktionen
  - 38 Basis-Instruktionen
  - Weniger mächtig als ARM
  - Wechsel jederzeit möglich: BX (Branch-and-Exchange) bei gesetztem untersten Bit der Adresse
- Jazelle-Modus
  - Beschleunigte Ausführung von Java-Bytecode

Instruktionssatz ARM I

- Alle Instruktionen sind 32 Bit lang
- Sehr strukturierte Kodierung: semantisch ähnliche Instruktionen werden auch ähnlich kodiert
- Bedingte Ausführung *aller* Instruktionen
  - Obere Bits (28-31) sind stets für die Bedingung reserviert
  - Keine Bedingung: AL (always)
  - Details: Nächste Folien
- Adressierungsmodi
  - Zwei oder drei Operanden (Ziel, Quelle, optionale Quelle)
  - Ziel und optionale Quelle immer Register
  - Quelle: “Shifter-Operand” — kann innerhalb eines Zyklus um in der Instruktion angegebenen Wert geshiftet werden
  - Details: Nächste Folien

Instruktionssatz ARM II

- Allgemeine Instruktionsform:

```
<opcode>{<cond>}{S} <Rd>, <Rn>, <shifter_operand>
```

- Branch mit Link:

```
BL{<cond>} <target_address>
```

  – R14 = Adresse der Instruction nach BL

Besonderheit: Bedingte Ausführung I

- Bedingte Ausführung aller Instruktionen spart Sprünge ein
- Verbessert Performance, kompensiert fehlende Sprungvorhersage
- Gilt im Thumb-Mode nur für Sprunganweisungen (kompakterer Code)
- Beispiel:

```
if (a == 0)
{
   b = 23;
}
else
{
   b = 42;
   c = a;
}
```

Besonderheit: Bedingte Ausführung II

In IA32-Code (ohne bedingte Ausführung aller Befehle)

```
        TEST EAX, EAX
        JNE _else
        MOV EBX, 23
        JMP _done
_else:  MOV EBX, 42
        MOV ECX, EAX
_done:
```

Besonderheit: Bedingte Ausführung III

In ARM-Code (mit bedingter Ausführung aller Befehle)

```
TEQ R0, #0
MOVEQ R1, #23
MOVNE R1, #42
MOVNE R2, R0
```

Besonderheit: Shifter-Operand

Innerhalb einer Operation kann ein Quelloperand geshiftet oder rotiert werden:

```
MOV R1, R0, LSL #3      ; R1 := R0 * 8 (ein Takt !)
ADD R3, R2, R1, LSR #2  ; R3 := R2 + R1 / 4
MOV R3, R2, ROR R1      ; R2 um R1 nach rechts rotieren und
                        ; Wert in R3 laden
```

ARM-Instruktionstypen

- Branch-Instruktionen
- Datenverarbeitende Instruktionen
- Multiplikations-Instruktionen
- Statusregister-Instruktionen
- Load/Store-Instruktionen
- Load/Store-Multiple-Instruktionen
- Semaphor-Instruktionen
- Instruktionen, die Exceptions generieren
- Koprozessor-Instruktionen

Load/Store-Multiple-Instruktionen

- Erlaubt das Laden von Registern mit dem Inhalt aufeinanderfolgender Speicheradressen bzw. das Speichern von Registern an aufeinanderfolgende Speicheradressen
- Mehrere Varianten
- Beispiel: STMDA R13!, {R0 - R12, LR}
  - STDMA — Store Multiply Decrement After
  - Indexregister wird nach jedem Schritt dekrementiert
  - R13 ist Indexregister mit der Basisspeicheradresse
  - ! besagt, daß R13 nach jedem Schritt aktualisiert wird
  - R0-R12 und LR sind die Register, die in den Speicher kopiert werden sollen
  - Ablauf: R0 nach Adresse, die in R13 steht, R13 um 4 dekrementieren, R1 an diese Adresse, usw.

Semaphor-Instruktionen

- SWP (Swap) und SWPB (Swap Byte) erlauben atomare Lese/Schreibe-Operationen
- Beispiel 1:
  - SWP R1, R2, [R3]
  - R3 enthält Speicheradresse
  - SWP lädt R1 mit dieser Speicheradresse und schreibt R2 an diese Adresse
- Beispiel 2:
  - SWP R1, R1, [R2]
  - R2 enthält Speicheradresse
  - SWP tauscht Inhalt von R1 und der Speicheradresse

Instruktionssatz Thumb I

- Motivation: 32-bit ARM Instruktionssatz einfach dekodierbar, aber geringe Code-Dichte. $\Rightarrow$ Sub-Befehlssatz mit 16-bit Instruktionen
- Seit ARM7TDMI (ARMv4T), 16 Bit Instruktionslänge; 38 Basis-Instruktionen, meist direkt abbildbar auf 32-bit Version (Standard-Argumente)
- Jederzeit Zugriff auf 8 Register (R0-R7)
- einzelne Instruktionen bieten auch Zugriff auf PC (R15), LR (R14), SP (R13) oder High Registers (R8-R15)
- Wechsel zur Laufzeit, Exceptions werden immer im ARM-Modus behandelt — in jedem ARM-System existiert damit gemischter Code
- Auch sinnvoll für Systeme, bei denen ein kleiner Teil des Speichers mit 32bit Bus, und der meiste Teil mit 16bit Bus angekoppelt ist (z.B. GameBoy Advance)

Instruktionssatz Thumb II

- Branch-Instruktionen
- Datenverarbeitende Instruktionen
- KEINE Multiplikations-Instruktionen
- KEINE Statusregister-Instruktionen
- Load/Store-Instruktionen
- Load/Store-Multiple-Instruktionen
- KEINE Semaphor-Instruktionen
- Instruktionen, die Exceptions generieren
- KEINE Koprozessor-Instruktionen

Jazelle

- Seit ARMv5TEJ Architektur
- Jazelle ist eine JVM, die spezielle Extensions in der ARM Hardware ausnutzt (zur Beschleunigung von JavaME Anwendungen)
- Extra Schritt zwischen 'fetch' und 'decode' in der Pipeline wandelt Java Bytecode Anweisung um, Zustand der JVM wird in Registern repräsentiert
- JVM muss Fallback für alle Instruktionen anbieten, Prozessor entscheidet selbst
- Seit ARMv7 nur noch triviale Unterstützung des Modus (keine Hardware-Instruktionen), stattdessen soll ThumbEE-Code durch JIT Compiler erzeugt werden (Java, C#, Python, Perl ,...)

Koprozessor-Architektur

ARM-Design ermöglicht einfache Erweiterungen durch die Koprozessor-Architektur

- Bis zu 16 Koprozessoren an einem ARM-Kern
- ARM ist für Flußsteuerung zuständig und lagert Berechnungen aus
- Kommunikation über Bus und Ausnahmen
- Wenn kein Koprozessor reagiert, gibt es eine undefined-Ausnahme
- Behandlung dieser Ausnahmen ermöglicht Software-Emulation
- Koprozessoren:
  - Wie ARM load/store
  - Bis zu 16 Register
  - Beliebige interne Verarbeitungsbreite
- Beispiel: Komplexer HDTV Transkoder, gesteuert durch kleinen ARM-Kern

Debugging

- Hardware-Unterstützung für Debugging - JTAG
  - Joint Test Action Group (JTAG) - IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture
  - Debugger für eingebettete Systeme basieren auf JTAG-Schnittstelle zum Prozessor
  - Auf den meisten Systemen ab der ersten Instruktion verfügbar - anhalten, Einzelschritt, laufen; Breakpoints; Flashen von angebundenen Speicher
  - 4 / 5 Pin Anbindung, Zugriff auf alle Chips, physikalischer Anschluß nicht standardisiert

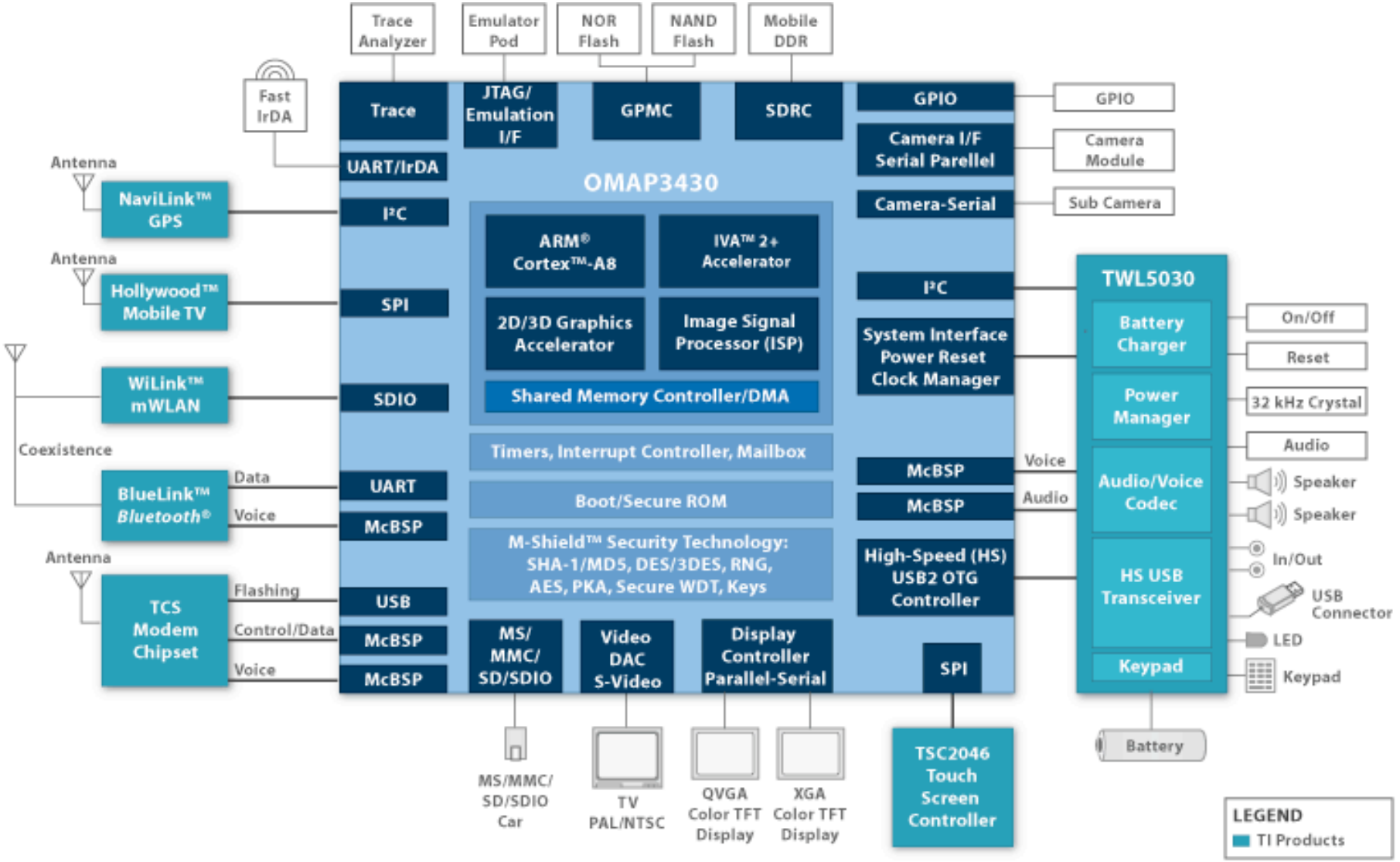
Implementationen

- Unterscheiden sich je nach Lizenznehmer stark
- Cortex-Familie: 69 Lizenznehmer (ARMv6-ARMv7)
- ARM11-Familie: 72 Lizenznehmer (ARMv6)
- ARM9-Familie: 253 Lizenznehmer (ARMv4-ARMv5)
- ARM7-Familie: 171 Lizenznehmer (ARMv3-ARMv5)
- Siehe auch: http://www.arm.com/products/licensing/licencees.html

Lizensierungen

- Für alle Lizenznehmer: Hardware-Beschreibung des Core, SDK, Recht zum Vertrieb von Silizium
- Hersteller ohne eigene Fab nur an verifizierten ’ semiconductor intellectual property core’ interessiert
  - ARM liefert Gate-Liste, Simulationsmodell und Testprogramme
- Chiphersteller erwerben Verilog-Quellen

Beispiel: Palm Pre mit TI OMAP3430



Entwicklungstools

- Abhängig von benutzter Hardware und Betriebssystem
- Allgemein:
  - ARMulator
  - SWARM (SoftWare ARM): C++ Simulator for ARM7
  - GNU toolchain für ARM
- Speziell:
  - Microsoft Visual Studio für Pocket PC
  - PalmOnes Entwicklungsumgebung für PalmOS
  - OpenEmbedded für linux-basierte PDAs und eingebettete Systeme