TECHNISCHE UNIVERSITÄT DRESDEN

Technische Universität Dresden • Fakultät Informatik • Institut Technische Informatik

HS Technische Informatik

Die AVR32-Architektur im Vergleich zur AVR8- und zur ARM-Architektur

Dresden, 11. Juni 2007

Atmel AVR - 8 Bit

- Entwicklung 1992 von *Alf Egil Bogen* und *Vegart Wollan*
- Einführung 1997 durch *Atmel* (Trondheim, Norwegen)

- 8-Bit-Mikrocontrollerfamilie
- weit über 100 aktive (binärkompatible) Modelle

- Harvard-Architektur
- RISC-Befehlssatz
- optimiert für C-Compiler
- umfangreiches Peripherieangebot

...

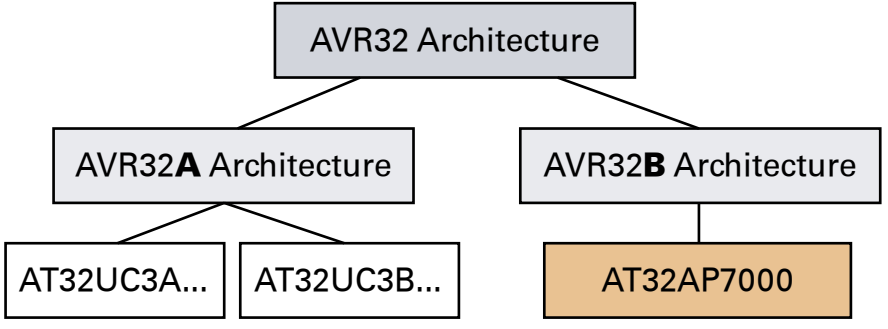
[C] embedit.de

- GCC-Port + kostenlose IDE vom Hersteller verfügbar

Atmel AVR - 32 Bit

- Einführung 2006 durch *Atmel*

- hochmoderne 32-Bit-Mikrocontrollerfamilie
- nicht kompatibel zur AVR-8-bit-Reihe



- GCC-Port + kostenfreie und plattformunabh. IDE verfügbar

ARM Architektur

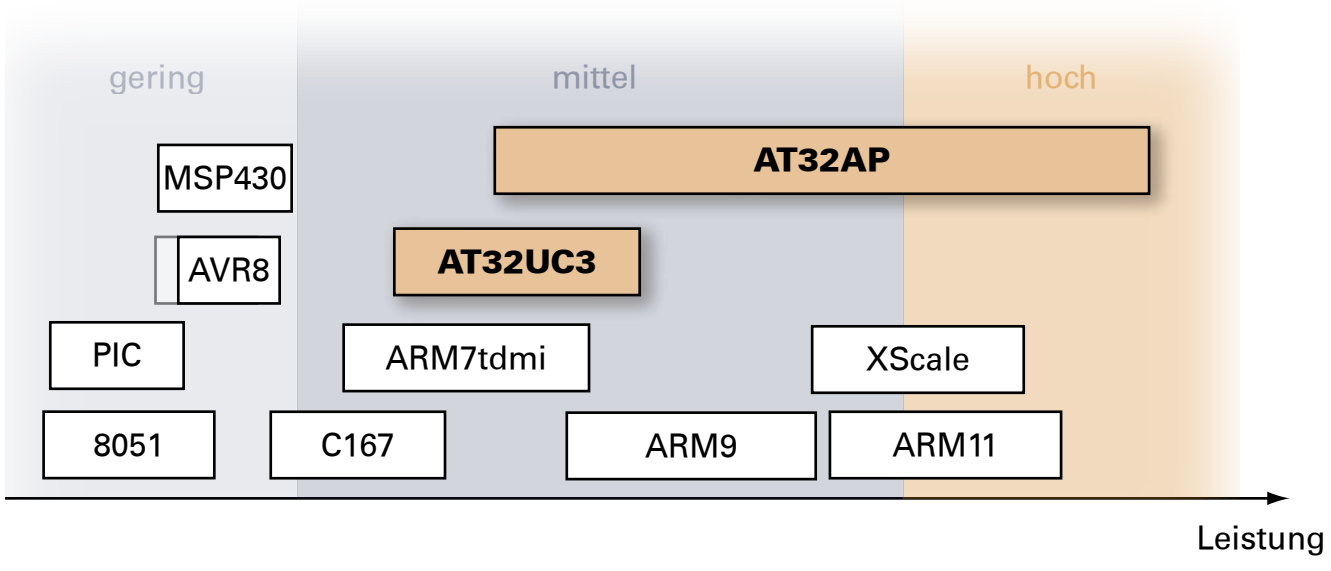
- 32-bit-RISC-Prozessorkern-Spezifikation
- Defacto Industriestandard

- entwickelt ab 1983 durch Acorn Ltd. (UK), Projekt „Acorn RISC Machine“
- lizenziert u.a. von Philips/NXP, Atmel, Intel, Motorola/Freescale, ST Micro, DEC...

- Spezifikation für unterschiedliche Leistungsklassen, aktuell *ARM7tdmi* ... *ARM9* ... *(XScale,) ARM11*

- verschiedene GCC-Ports verfügbar

Einordnung


gering
mittel
hoch
MSP430
AT32AP
AVR8
AT32UC3
PIC
ARM7tdmi
XScale
8051
C167
ARM9
ARM11
Leistung

Hinweis

Vergleichende Informationen in diesem Vortrag beziehen sich ausschließlich auf die „alten“ ARM-Kerne vor der Generation Cortex, da letztere zur ENTWICKLUNGSZEIT des AVR32 noch nicht vorlagen.

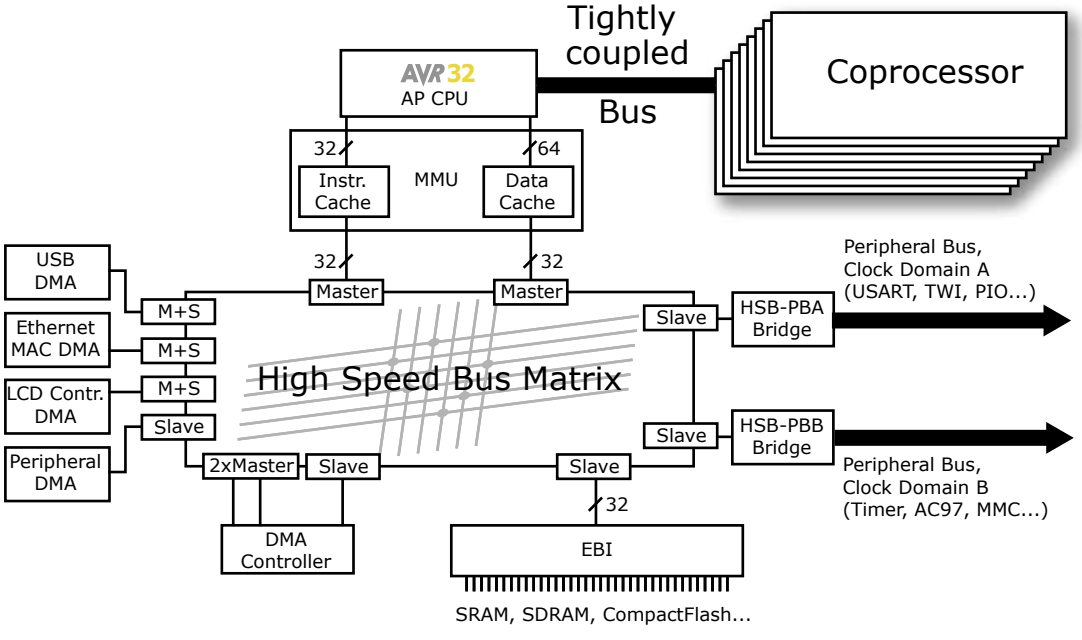
Da dieser Vortrag vornehmlich der Beschreibung des AVR32 gewidmet ist, sind nur vereinzelt Vergleiche zum ARM vorhanden. Sollte ein spezielles Feature nur in bezug auf den AVR32 erwähnt werden, bedeutet dies nicht, dass es beim ARM nicht in ähnlicher Art und Weise verfügbar ist.

Alle Informationen wurden nach bestem Gewissen aus den angegebenen Quellen zusammengetragen. Fehler sind selbstverständlich trotzdem nicht ausgeschlossen.

Gliederung

1. Speichersubsystem
2. Prozessorkern
3. Befehlssatz
4. Interrupts & Exceptions
5. Peripherie

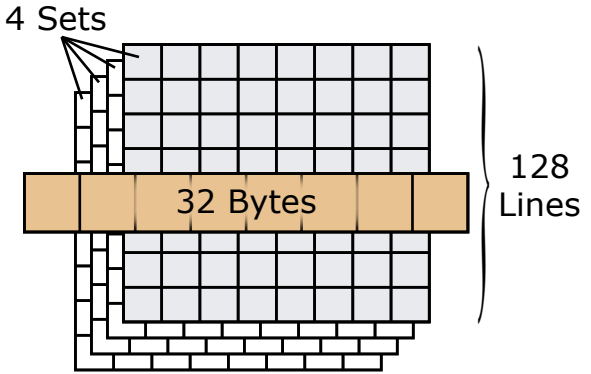
Bussystem


Tightly coupled Bus
AVR32
AP CPU
Coprocessor
32
64
Instr. Cache
MMU
Data Cache
32
32
Master
Master
USB DMA
Ethernet MAC DMA
LCD Contr. DMA
Peripheral DMA
M+S
M+S
M+S
Slave
High Speed Bus Matrix
Slave
HSB-PBA Bridge
Peripheral Bus, Clock Domain A (USART, TWI, PIO...)
Slave
HSB-PBB Bridge
Peripheral Bus, Clock Domain B (Timer, AC97, MMC...)
2xMaster
Slave
Slave
32
DMA Controller
EBI
SRAM, SDRAM, CompactFlash...

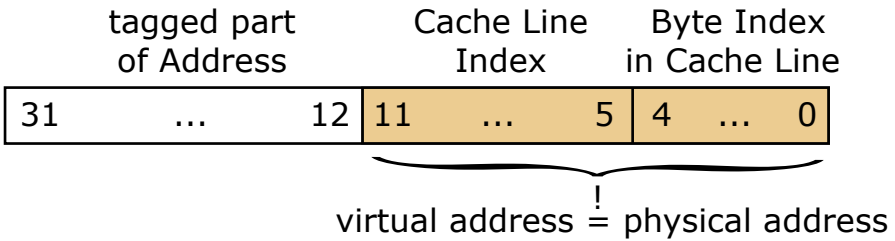
Instruction Cache

- 4-Wege-Assoziativ

  4 Sets
  × 128 Zeilen
  × 32 Bytes
  = 16 kByte



- indiziert über virt. Adresse, Tag-RAM enthält phys. Adresse



Data Cache

- Größe, Aufbau und Adressierung wie Instruction Cache
- WriteBack/Write Through Rückschreibeverhalten durch MMU für jede Seite getrennt einstellbar
- zwei Zeilen großer Puffer zwischen D-Cache und HMatrix
- Flush-Operationen für Cache und Write Buffer im Befehlssatz integriert

MMU(1)

- vollständige Unterstützung moderner Betriebssysteme
  - → virtuelle Adressräume
  - → Paging
  - → Schutzkonzepte
- realisiert durch Zusammenarbeit von HW und SW:

HW-Anteil:

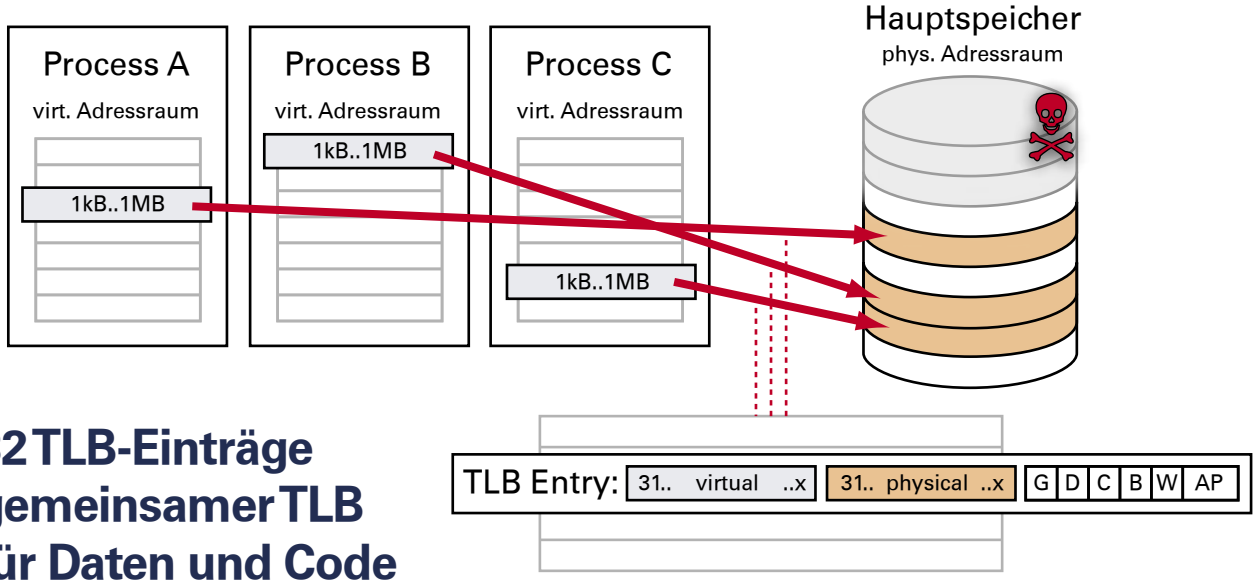
- Adressübersetzung...
- Rechteprüfung...
  ...durch TLB-Suche
- Ausnahme-Generierung

SW-Teil:

- Seitentabelle verwalten
- TLB-Einträge (nach-)laden
- Ersetzungs-Strategie
- Ausnahmebehandlung

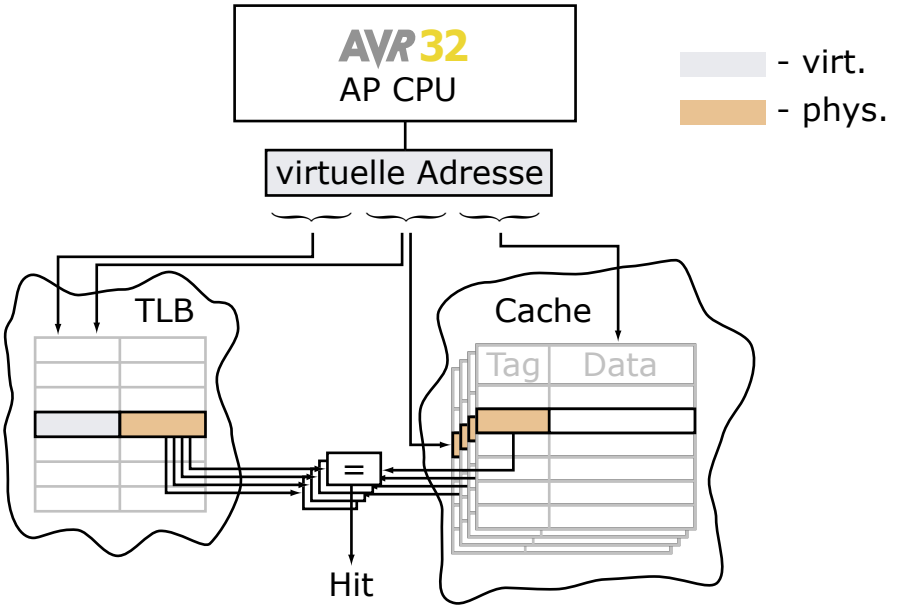
MMU(2)

- Abbildung virtueller auf physikalische Adressen



- 32 TLB-Einträge
- gemeinsamer TLB für Daten und Code

MMU & Cache



- **Achtung: mögliche Inkonsistenz von Cache und TLB bei Seitengröße < 4kB!**

Zusammenfassung Speicher

- Split Cache, je 4-Wege-assoziativ
- intelligenter Multi-Level-Crossbar *„HMatrix“* verbindet Kern, Speicher und restliche Systemkomponenten
- MMU unterstützt virtuelle Adressräume und Zugriffsschutz
- unified TLB zur Adressübersetzung und Rechteprüfung
- Software verwaltet Seitentabelle und füllt TLB

Gliederung

1. Speichersubsystem

2. Prozessorkern

3. Befehlssatz

4. Interrupts & Exceptions

5. Peripherie

Registersatz(1)

AVR32-Registersatz

R15 = PC
R14 = LR
R13 = SP
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0

- 16 x 32-bit Universalregister
- ein Statusregister

- R15: *Program Counter*
- R14: *Link Register* für Rücksprungadresse
- R13: *Stack Pointer*

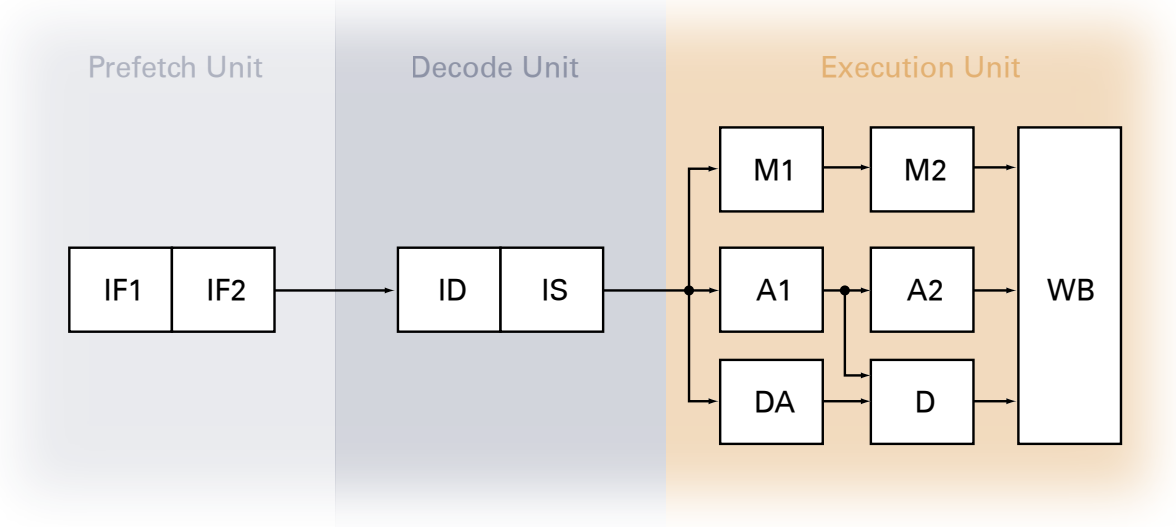
- Implementiert als Multiport-RAM mit 7 Read + 3 Write-Ports

- ARM-Registersatz identisch

Register-Shadowing

App	Supervisor	Int0	Int1	Int2	Int3	Exception	NMI
PC	PC	PC	PC	PC	PC	PC	PC
LR	LR	LR	LR	LR	LR_Int3	LR	LR
SP_App	SP_Sys	SP_Sys	SP_Sys	SP_Sys	SP_Int3	SP_Sys	SP_Sys
R12	R12	R12	R12	R12	R12_Int3	R12	R12
R11	R11	R11	R11	R11	R11_Int3	R11	R11
R10	R10	R10	R10	R10	R10_Int3	R10	R10
R9	R9	R9	R9	R9	R9_Int3	R9	R9
R8	R8	R8	R8	R8	R8_Int3	R8	R8
R7	R7	R7	R7	R7	R7	R7	R7
R6	R6	R6	R6	R6	R6	R6	R6
R5	R5	R5	R5	R5	R5	R5	R5
R4	R4	R4	R4	R4	R4	R4	R4
R3	R3	R3	R3	R3	R3	R3	R3
R2	R2	R2	R2	R2	R2	R2	R2
R1	R1	R1	R1	R1	R1	R1	R1
R0	R0	R0	R0	R0	R0	R0	R0
SR	SR	SR	SR	SR	SR	SR	SR
	RSR_Sup	RSR_Int0	RSR_Int1	RSR_Int2	RSR_Int3	RSR_Ex	RSR_Nmi
	RAR_Sup	RAR_Int0	RAR_Int1	RAR_Int2	RAR_Int3	RAR_Ex	RAR_Nmi

Pipeline



- 3 parallele Ausführungseinheiten: ALU, Multiplier (auch Division), Load/Store
- Transparente Hazard-Detection-Hardware, Behandlung durch Forwarding oder Pipeline-Stall

Prefetch Unit

- 96 Bit *Instruction Buffer* (FIFO) für 3-6 Befehle
- liefert 1 Befehl/Takt an Decode Unit

- *Branch Prediction* für bedingte Sprünge (transparent)
  → nutzt *Branch Target Buffer* (Sprungziel-Cache)
  + *Branch Folding* ersetzt Sprungbefehl durch Folgebefehl
    → Sprungbefehl benötigt 0 Zyklen!

- 4-Elemente *Return-Cache* führt Rücksprung ohne Stack-Zugriff bereits in der Prefetch-Phase aus

Decode Unit

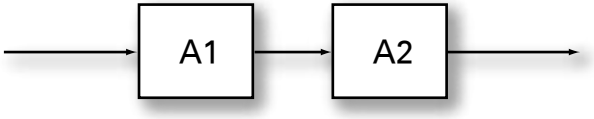
Instruction Decode:

- Dekodierung von RISC-Instruktionen oder Java-Bytecodes
- Rechteprüfung
- Adressgenerierung für Register-File

Instruction ISsue:

- Lesen von Register-File

ALU Subpipe



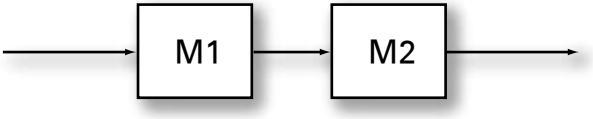
A1:

- Arithmetisch-/Logische Instruktionen
- Überprüfung der Sprungvorhersage
- Adressberechnung für indizierte Speicherzugriffe

A2:

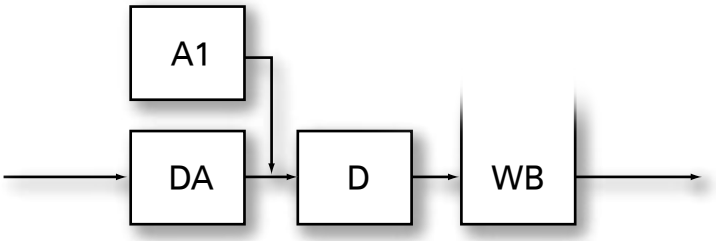
- Saturierung

Multiply Subpipe



- 16 × 32 Bit Hardware-Multiplizierer
- transparenter *Accumulator Cache*
  → Beschleunigung aufeinanderfolgender MAC-Befehle, die das selbe Akkumulator-Register nutzen um einen Takt, da ein Zugriff auf das Register-File entfällt
- serieller 32 Bit Hardware-Dividierer (33/34 Zyklen)

Load/Store Subpipe



DA:

- Adressen generieren
- Zeiger-Inkrement/ Dekrement

D:

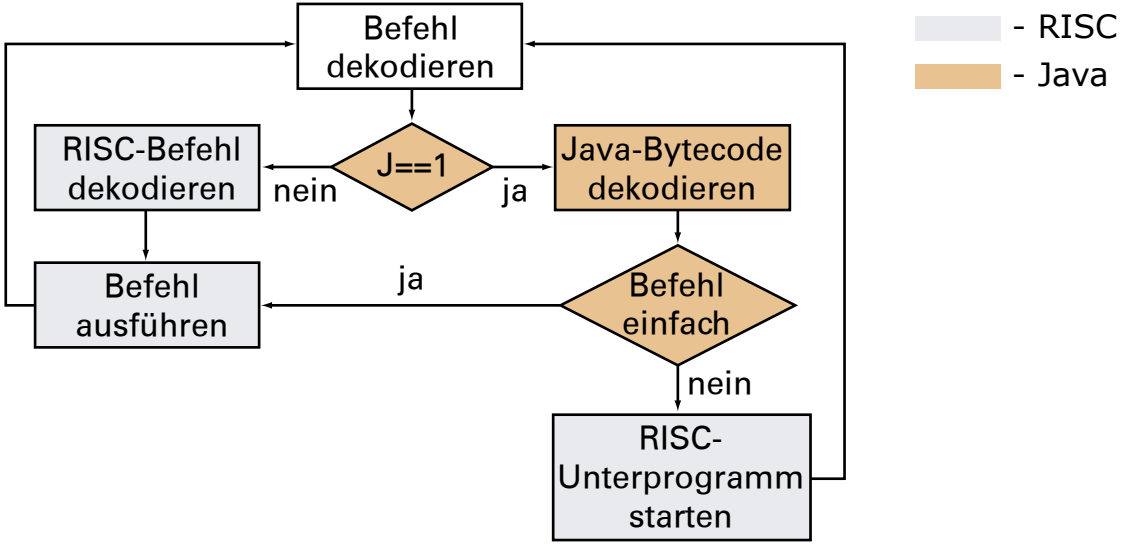
- Kommunikation mit Daten-Cache

WB:

- Big/Little Endian Konvertierung
- Bitfeld-Manipulation

Java Extension Module(1)

- **Java-Modus: Austausch des Befehlsdecoders, Umadressierung der Register**



Java Extension Module(2)

```
void main(){
    risc_func1(...);
    ajvm(arguments);
        void ajvm(){
            init();
            classloader();
            retj;
                iconst_1
                istore_0
                iconst_2
                getfield
                    mfsr R12, JECR
                    cp R12, 0x08
                    ...
                    retj
                ...
                return
            cleanup();
        }
   risc_func2(...);
}
```

konventionelle Hochsprache (C/C++)

Java Virtual Machine

Java Extension Module

RISC-Trap
→ nächste Folie

Java Extension Module(3)

Abarbeitung nicht unterstützter Java-Bytecodes:

Hardware:

1. Erkennung betr. Bytecodes durch Befehlsdecoder
2. auslösenden Bytecode + Operanden in Systemregister JECR (= *Java Exception Cause Register*) kopieren
3. autom. Verzweigung zur RISC-Routine an Adresse JTBA + bytecode-spezifischer Offset (Systemregister JTBA = *Java Trap Base Address*)

Software (JVM):

4. Emulation des Bytecodes durch RISC-Unterprogramm
5. Rückkehr zum Java-Modus durch RISC-Instruktion `retj`

Zusammenfassung Kern

- 16 Universalregister einschließlich PC und SP (wie bei ARM)
- 7-Stufen-Pipeline, 3 parallele Ausführungseinheiten
- Branch Prediction & Branch Folding
- Befehlsdecoder unterstützt Java-Bytecodes

Gliederung

1. Speichersubsystem

2. Prozessorkern

3. Befehlssatz

4. Interrupts & Exceptions

5. Peripherie

ARM-Befehlssatz

- klassischer ARM-RISC-Befehlssatz, 32 Bit Befehlslänge
  → hohe Leistung
- reduzierter Befehlssatz „Thumb“, 16 Bit Befehlslänge
  → Kompakt

- implementierte Instruktionen abh. von Instruction Set Version

  z.B. v4 (Arm7tdmi): Basisbefehle + Multiplikation & MAC
  z.B. v6 (Arm1136JF): zusätzlich SIMD + DSP + FP

(neuere Entwicklung (Thumb2) nicht betrachtet)

AVR32-Befehlssatz

- 214 definierte RISC-Instruktionen, 58 Formate, optimiert für C
- Befehlslänge: 16 Bit, einige 32 Bit
- Befehlsklassen:
  1. „klassische“ Arithmetik-/Logik-Befehle, Load/Store etc.
  2. Bit(-feld-)manipulation
  3. DSP-Instruktionen (MAC, Saturierung)
  4. SIMD-Instruktionen (PAVG, PADDSUB, ...)
  5. Koprozessor-Steuerung
  6. Systemsteuerung (Cache, TLB etc.)

  geplant: 7. Floating Point

ARM: bedingte Ausführung

- 4 Bit Condition Code je Befehl im 32-Bit-Befehlssatz

Opcode [31:28]	Mnemonic extension	Meaning	Condition flag state
0000	EQ	Equal	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set/unsigned higher or same	C set
0011	CC/LO	Carry clear/unsigned lower	C clear
0100	MI	Minus/negative	N set
0101	PL	Plus/positive or zero	N clear
0110	VS	Overflow	V set
0111	VC	No overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z set
1010	GE	Signed greater than or equal	N set and V set, or N clear and V clear (N == V)
1011	LT	Signed less than	N set and V clear, or N clear and V set (N != V)
1100	GT	Signed greater than	Z clear, and either N set and V set, or N clear and V clear (Z == 0,N == V)
1101	LE	Signed less than or equal	Z set, or N set and V clear, or N clear and V set (Z == 1 or N != V)
1110	AL	Always (unconditional)	-
1111	(NV)	See *Condition code 0b1111* on page A3-5	-

AVR32: bedingte Ausführung

- 4 Bit Condition Code in ausgewählten 32-Bit-Befehlen
- 3 Bit Condition Code in ausgewählten 16-Bit-Befehlen

Coding in cond3	Coding in cond4	Condition mnemonic	Evaluated expression	Numerical format	Meaning
B'000	B'0000	eq	Z		Equal
B'001	B'0001	ne	¬Z		Not equal
B'010	B'0010	cc / hs	¬C	Unsigned	Higher or same
B'011	B'0011	cs / lo	C	Unsigned	Lower
B'100	B'0100	ge	N == V	Signed	Greater than or equal
B'101	B'0101	lt	N ⊕ V	Signed	Less than
B'110	B'0110	mi	N	Signed	Minus / negative
B'111	B'0111	pl	¬N	Signed	Plus / positive
N/A	B'1000	ls	C ∨ Z	Unsigned	Lower or same
N/A	B'1001	gt	¬Z ∧ (N==V)	Signed	Greater than
N/A	B'1010	le	Z ∨ (N ⊕ V)	Signed	Less than or equal
N/A	B'1011	hi	¬C ∧ ¬Z	Unsigned	Higher
N/A	B'1100	vs	V		Overflow
N/A	B'1101	vc	¬V		No overflow
N/A	B'1110	qs	Q	Fractional	Saturation
N/A	B'1111	al	True		Always

DSP-Befehle

- **Multiply Accumulate**

$$y_i = \sum_{i=0}^{n} a_i \cdot x_i \quad \longrightarrow \quad \texttt{MAC Rd, Rx, Ry} \quad \longrightarrow \quad R_d = R_x \cdot R_y + R_d$$

- **Saturierung: Begrenzung statt Überlauf**

```
      ADD Rd, Rx, Ry ⟶ 0x7FFFFFFF + 0x00000001 = 0x80000000
SATADD.W Rd, Rx, Ry ⟶ 0x7FFFFFFF + 0x00000001 = 0x7FFFFFFF
```

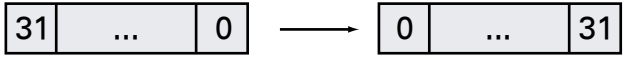
- **8 Bit und 16 Bit Datentypen**

```
ADDHH.W Rd, Rx:t, Ry:b ⟶ Rd[31:0] = Rx[31:16] + Ry[15:0]
  LDINS.B Rd:h, Rp[12] ⟶ Rd[24:16] = *(Rp + 12)
  MULWH.D Rd, Rx, Ry:b ⟶ (Rd+1:Rd) = (Rx * Ry[15:0]) & 0x0000
```

Bitmanupulations-Befehle

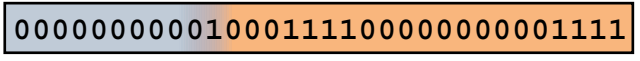
- **Bit-Reverse (FFT etc.)**

```
BREV Rd
```



- **Count Leading Zeros (Huffman-Coding etc.)**

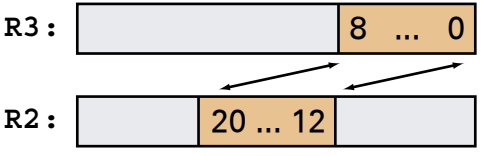
```
CLZ Rd, Rs
```




```
Rd = 10
```

- **Bitfeldmanipulation (Zugriff auf Peripherie etc.)**

```
struct{
    long BYPASS: 1;
    long reserved1: 11;
    long CLKVAL: 9;
    long LINECNT: 11;
}AP7000_LCDCON1;
     ...
AP7000_LCDCON1.CLKVAL += i;
```




```
...
BFEXT.U R3, R2, 12, 9
ADD R3, R4 ; R4 contains i
BFINS R2, R3, 12, 9
...
```

SIMD-Befehle

- Registeraufteilung in 2 × 16 Bit oder 4 × 8 Bit
- „homogene“ Operationen:

```
PADD[S], PSUB[S], PABS, PAVG, PMAX, PMIN, PLSL, PLSR
```

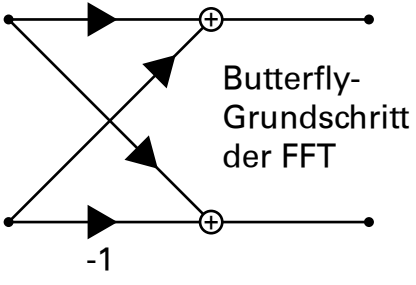
- „heterogene“ Operationen:

```
PADDSUB[S|H], PSUBADD[S|H], PSAD, ...
```

```
PADDSUBH Rd, Rx:b, Ry:t

Rd[31:16] = (Rx[15:0]  + Ry[31:16]) / 2
Rd[15:0]  = (Rx[31:16] - Ry[15:0] ) / 2
```

→ 1 Takt!



Zusammenfassg. Befehlssatz

- RISC-Befehlssatz mit DSP und SIMD-Erweiterungen
- optimiert für C

- ARM: 32-Bit-Befehlssatz oder 16-Bit-Befehlssatz (Thumb)
  → Optimierung Größe/Geschw. auf Funktionsebene
- AVR32: Befehlslänge 16 und 32 Bit
  → Optimierung Größe/Geschw. auf Instruktionsebene

- Condition-Code-Feld im Befehl erlaubt bedingte Ausführung

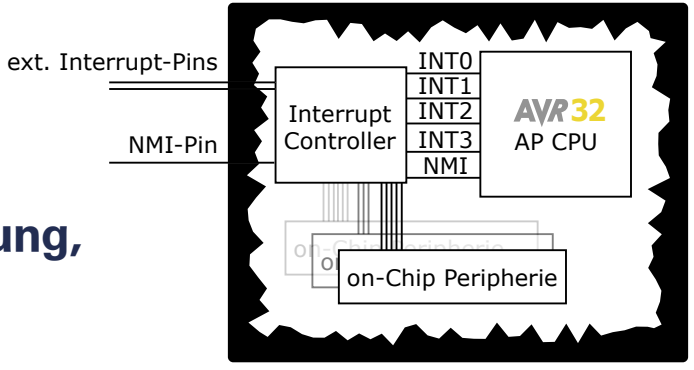
Gliederung

1. Speichersubsystem
2. Prozessorkern
3. Befehlssatz
4. **Interrupts & Exceptions**
5. Peripherie

Begriffsklärung

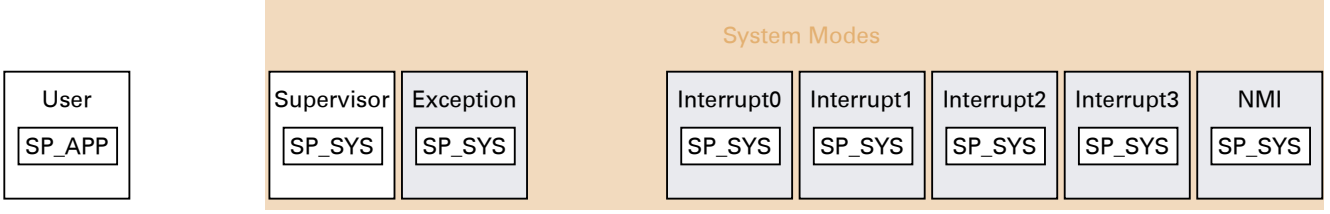
- **Exceptions: vom Prozessorkern selbst generierte Ausnahme**
  Quellen: → TLB Miss
  → access violation
  → invalid Opcode
  → ... (bis zu 28 Quellen bei AVR32)

- **Interrupts:**
  von Komponente außerhalb des Kerns generierte Unterbrechungsanforderung, z.B. Timer-Überlauf...

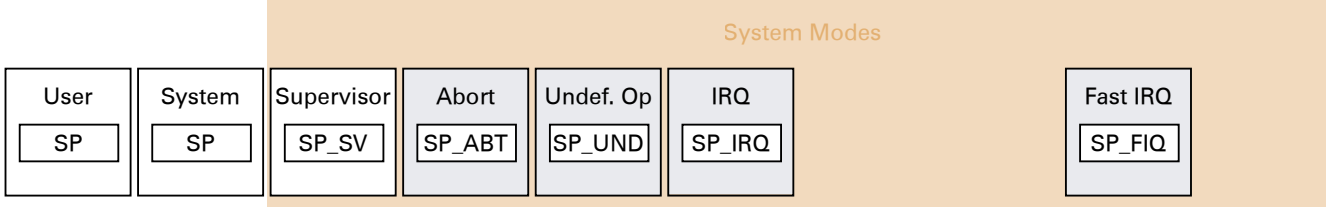


Prozessormodi

- AVR32-Prozessormodi



- ARM-Prozessormodi



Exception-Behandlung

- Generierung z.B. von TLB, ID-Pipeline-Stufe etc.
- Abarbeitung in Reihenfolge des Auftretens, feste Prioritäten bei gleichzeitigem Auftreten
- Aktionen, wenn EM (Exception Mask) = 0:

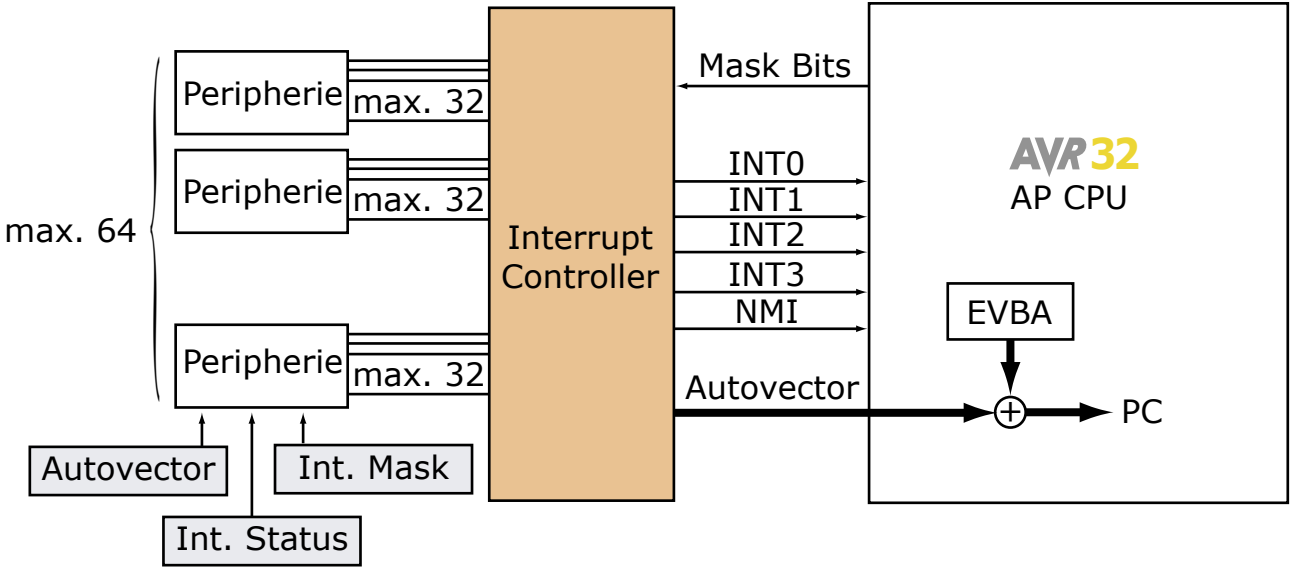
1. Prozessor wechselt in den Exception-Mode
2. Aktivieren der Shadow-Register für SR und PC
3. EM Bit in SR setzen
4. Handler-Adresse anspringen (Systemregister EVBA + Exception-spezifischer Offset)

Interrupt-Behandlung(1)

- Interrupt-Controller außerhalb des Kerns verwaltet Anfragen von Peripherie-Komponenten und steuert Int0..3/NMI-Signale an den Kern (siehe nächste Folie)
- Priorität aufsteigend: Int0, Int1, Int2, Int3, NMI
- Aktionen, wenn GM (Global Interrupt Mask) = 0:
  1. Prozessor wechselt in den Int0..3 bzw. NMI-Modus
  2. Aktivieren der Shadow-Register für SR, PC und bei Int3 zusätzlich R8..R12 sowie LR
  3. Mask Bits alle für Int-Level ≤ akt. Int-Level setzen
  4. Handler-Adresse anspringen (Systemregister EVBA + Offset von Interrupt Controller)

Interrupt-Behandlung(2)

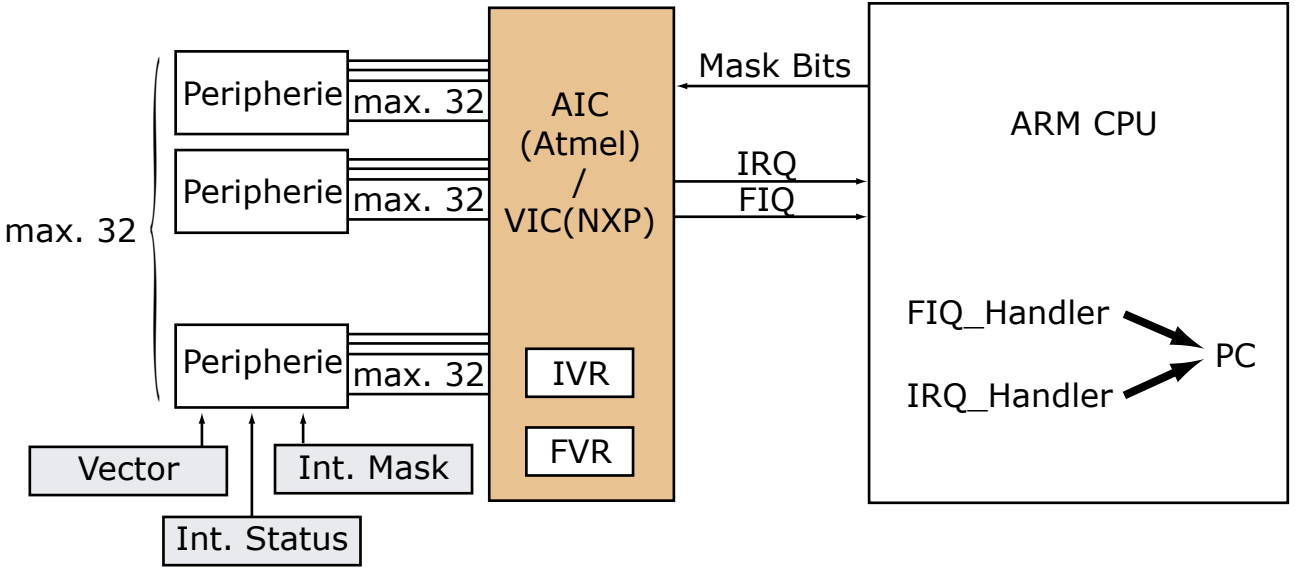
- Zusammenwirken von AVR32-Kern und Interrupt Controller



- 64 × 32 = 2048 Peripherie-Interruptleitungen möglich

Interrupt-Behandlung(3)

- **Behandlung bei ARM-Kernen vor Cortex-X:**



- **32 × 32 = 1024 Peripherie-Interruptleitungen möglich**

Zusammenfassung I+E

- zwei Stacks: User Stack und System Stack
- Ausnahmebehandlung immer mit System Stack

- kerninterne Ausnahmebehandlung im *Exception Mode,* externe Unterbrechungsanforderungen in Int0..3 und NMI
- Shadow-Register für R8..R14 im Modus Int3

- Interrupt Controller verwaltet Peripherie-Unterbrechungsaufforderungen

Gliederung

1. Speichersubsystem

2. Prozessorkern

3. Befehlssatz

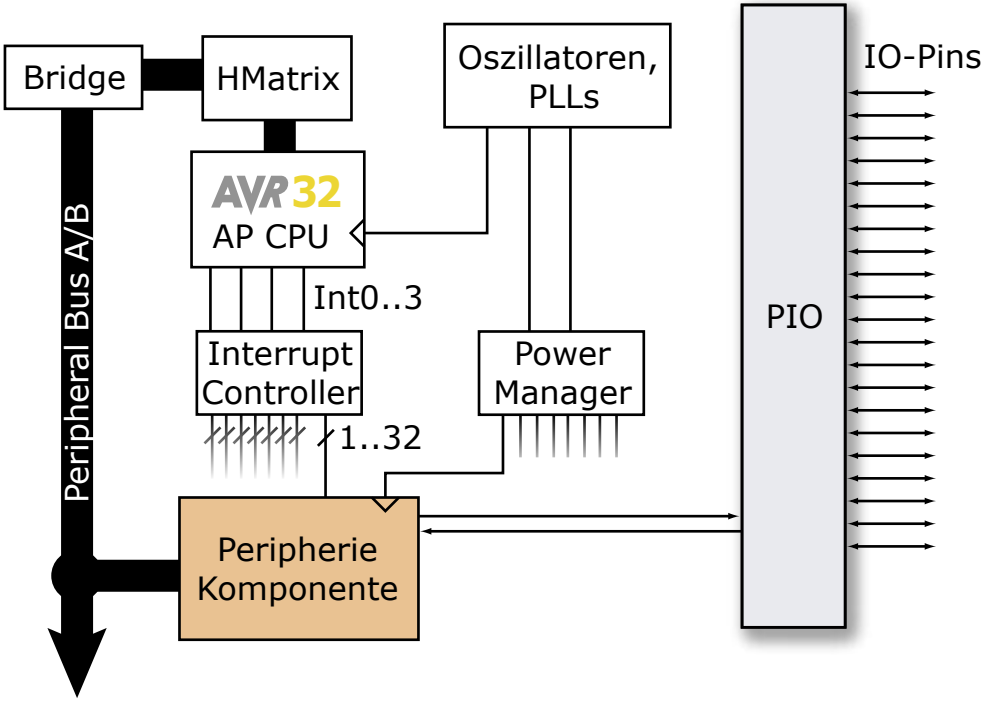
4. Interrupts & Exceptions

5. Peripherie

Peripherie-Komponenten

- Standard-Peripherie:
  - → Timer
  - → GPIO
  - → Serielle Schnittstellen
- Systemperipherie
  - → Busmatrix (HMatrix)
  - → Memory Controller
  - → Interrupt-Controller
  - → Power-Manager
- Multimedia-Peripherie
  - → LCD-Controller
  - → CMOS-Sensor-Interface
  - → 16-bit Stereo-Audio-DAC
  - → SD/MMC-Card-Interface
- Netzwerk
  - → USB-Device-Port (Hi)
  - → 2x Ethernet

Peripherie-Integration



Peripherie-Konfiguration(1)

- Memory Mapped IO-Register an geschützten Adressen

```
typedef struct avr32_usart_brgr_t {
    unsigned int: 13;
    unsigned int fp: 3;
    unsigned int cd: 16;
} avr32_usart_brgr_t;

typedef struct avr32_usart_t {
    ...
    union {
        unsigned long brgr
        avr32_usart_brgr_t BRGR
    };
    ...
} avr32_usart_t;

#define AVR32_USART0_ADDRESS  0xFFE00C00
#define AVR32_USART0  (*((volatile avr32_usart_t*)AVR32_USART0_ADDRESS))
```

Peripherie-Konfiguration(2)

- AVR32-Peripherie ähnelt stark Atmel-ARM-Peripherie

23.9 Synchronous Serial Controller (SSC) User Interface

Table 23-4. Register Mapping

Offset	Register	Register Name	Access	Reset
0x0	Control Register	CR	Write	–
0x4	Clock Mode Register	CMR	Read/Write	0x0
0x8	Reserved	–	–	–
0xC	Reserved	–	–	–
0x10	Receive Clock Mode Register	RCMR	Read/Write	0x0
0x14	Receive Frame Mode Register	RFMR	Read/Write	0x0
0x18	Transmit Clock Mode Register	TCMR	Read/Write	0x0
0x1C	Transmit Frame Mode Register	TFMR	Read/Write	0x0
0x20	Receive Holding Register	RHR	Read	0x0
0x24	Transmit Holding Register	THR	Write	–
0x28	Reserved	–	–	–
0x2C	Reserved	–	–	–
0x30	Receive Sync. Holding Register	RSHR	Read	0x0
0x34	Transmit Sync. Holding Register	TSHR	Read/Write	0x0
0x38	Receive Compare 0 Register	RC0R	Read/Write	0x0
0x3C	Receive Compare 1 Register	RC1R	Read/Write	0x0
0x40	Status Register	SR	Read	0x000000CC
0x44	Interrupt Enable Register	IER	Write	–
0x48	Interrupt Disable Register	IDR	Write	–
0x4C	Interrupt Mask Register	IMR	Read	0x0
0x50-0xFC	Reserved	–	–	–
0x100- 0x124	Reserved for Peripheral Data Controller (PDC)	–	–	–

31.8 Synchronous Serial Controller (SSC) User Interface

Register	Register Name	Access	Reset
	SSC_CR	Write	–
r	SSC_CMR	Read/Write	0x0
	–	–	–
	–	–	–
e Register	SSC_RCMR	Read/Write	0x0
e Register	SSC_RFMR	Read/Write	0x0
e Register	SSC_TCMR	Read/Write	0x0
e Register	SSC_TFMR	Read/Write	0x0
gister	SSC_RHR	Read	0x0
gister	SSC_THR	Write	–
	–	–	–
	–	–	–
ng Register	SSC_RSHR	Read	0x0
ng Register	SSC_TSHR	Read/Write	0x0
Register	SSC_RC0R	Read/Write	0x0
Register	SSC_RC1R	Read/Write	0x0
	SSC_SR	Read	0x000000CC
gister	SSC_IER	Write	–
gister	SSC_IDR	Write	–
ster	SSC_IMR	Read	0x0
	–	–	–
eral Data Controller (PDC)	–	–	–

Zusammenfassg. Peripherie

- viele Peripherie-Komponenten ähnlich bis identisch bei (Atmel-)ARM und AVR32
  → erleichtert Umstieg!
- Konfiguration über Memory-Mapped-IO-Register
- Strom sparen: jede Komponente kann über *Power Manager* deaktiviert werden
- Multiplexing von IO-Pins mit Peripherie-Funktionen durch *Parallel IO Controller (PIO)*

Bewertung(1)

- hochmoderne Prozessorarchitektur für eingebettete Multimedia-Systeme
- Allround-MCU dank umfangreicher Peripherie

- neu sind nicht die Features, sondern ihre Anwendung in dieser Leistungsklasse

- einzige Gemeinsamkeit von AVR8 und AVR32: der Name!
- auffallende Ähnlichkeit zu ARM-Kernen, jedoch viele Verbesserungen
  → Atmels lizenzfreie Konkurrenz zum ARM

Bewertung(2)

- Befehlssatz optimiert für moderne rechenintensive Multimedia-Anwendungen
- Leistungsfähigkeit
- Codedichte
- C-Optimierung
- umfangreiche ARM-ähnliche Atmel-Peripherie
- Atmel-Support (Toolkette, IDE, BSP, Appnotes...)
- geringer Stromverbrauch

- Bindung an einen Hersteller
- junge Architektur → potentielles Risiko von Silicon Bugs

AVR8 erreichte in nur 10 J. 30% Marktanteil → AVR32... ?!?

Vergleich(1)

		AVR32B	ARM7	ARM9	ARM11
SPEICHER	Max. Taktfrequenz	~180MHz	55..70MHz	<200MHz	>500MHz
	Architektur	Harvard	v. Neumann	Harvard	Harvard
	1st Level Cache	Split-Cache	-	Split-Cache	Split-Cache
	2nd Level Cache	-	-	-	unified
	MMU	ja	nein	ja	ja
	TLB nachladen	Software	-	Software	Hardware
	Coprozessorunterstützung	bis zu 8	bis zu 16	bis zu 16	bis zu 16
	Multiprozessor-unterstützung	nein	nein	nein	ja
KERN	Register	16x32 bit	16x32 bit	16x32 bit	16x32 bit
	Pipelinestufen	7	3	5	8
	paral. Execution units	3	1	1	3
	HW-Multiplier	16x32 bit	8x32 bit	8x32 bit	?x32 bit
	Branch prediction	ja	nein	ja	ja
	Branch folding	ja	nein	nein	nein (?)
	Return Stack	4 Elemente	-	-	3 Elemente

Vergleich(2)

ALLE	Befehlslänge	16+32 bit	32 bit ARM 16 bit Thumb	32 bit ARM 16 bit Thumb	32 bit ARM 16 bit Thumb
	Bedingte Ausführung	16+32 bit	nur 32 bit	nur 32 bit	nur 32 bit
	Relative Codedichte [1]	1	1.6 (ARM) 1.1 (Thumb)	1.6 (ARM) 1.1 (Thumb)	1.6 (ARM) 1.1 (Thumb)
BEFEHLE	Rel. Performance [1]	1	k.a.	0.6	0.75
	DSP-Befehle	ja	nur MAC	nur MAC	ja
	SIMD-Befehle	ja	nein	nein	ja
	Bitmanipulation	ja	nein	wenige	wenige
	Multiprozessor-Synchron.	nein	nein	nein	ja
	Floating-Point	nein (geplant)	nein	nein	Koprozessor
	Java-HW-Beschleunigung	ja, JEM	nein	nein	ja, Jazelle
INT	Interrupt-Leitungen	4+NMI	2	2	2
	Autovektor	ja	nein	nein	möglich
	Register-Shadowing	ja (INT3)	ja (FIQ)	ja (FIQ)	ja (FIQ)

[1] Quelle: http://www.atmel.com/products/avr32/ap7/ap7_3.asp?family_id=682

Quellen

- AVR32 Architecture Reference Manual (32000B-AVR32-11/07)
- AVR32AP Technical Reference Manual (32001A-AVR32-06/06)
- AT32AP7000 Datasheet (32003K-AVR32-10/07)

- ARM Architecture Reference Manual (ARM DDI 0100E)
- ARM7tdmi Technical Reference Manual (ARM DDI 0029G)
- AT91SAM7X... Datasheet (6120F-ATARM-03-Oct-06)

- AT90CAN128 Datasheet (Rev. 7679D-CAN-02/07)

- http://www.electronicsweekly.com/Articles/2007/01/31/40641/arm-dominated-mcu-market-is-ready-for-change.htm
- http://www.atmel.com/dyn/corporate/view_detail.asp?FileName=Ships500M.htm