

Die AVR Mikrocontrollerfamilie

Johannes Bauer

AKES - Ausgewählte Kapitel eingebetteter Systeme

10. Mai 2006

Was sind AVR's?

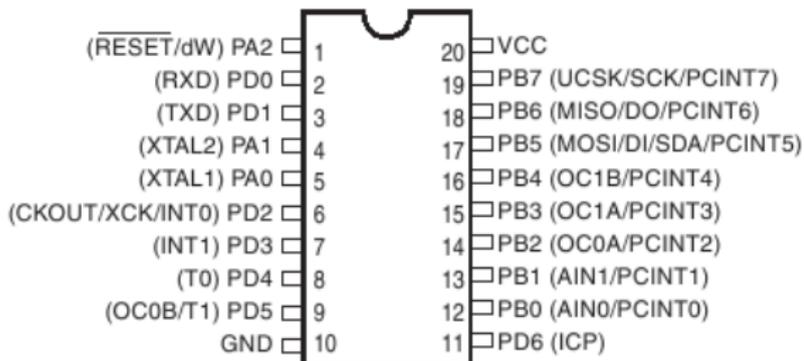
2/35

- ▶ Anwendungsfelder
- ▶ Hardwareeigenschaften
- ▶ Programmierung

Mikrocontroller

3/35

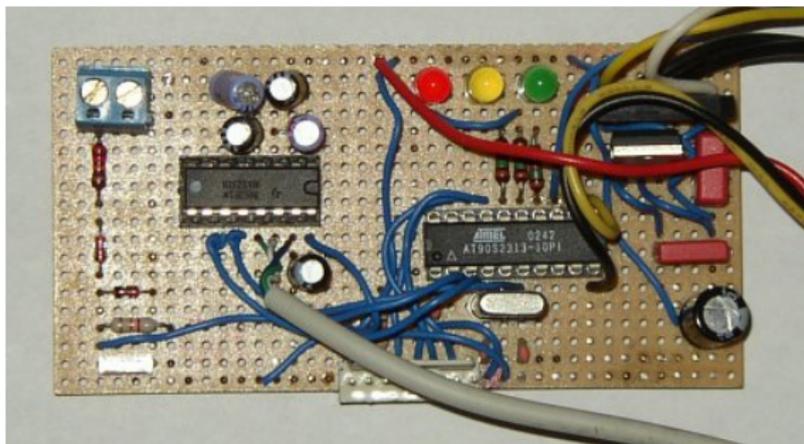
- ▶ Kleiner Chip
- ▶ Viel integrierte Hardware



Mikrocontroller

4/35

- ▶ Nur noch wenig zusätzliche ICs notwendig
- ▶ Reduktion externer Bauelemente



Anwendungsbeispiel

5/35

- ▶ **Warum einen AVR?**
- ▶ Einfach beschaffbar
- ▶ Leicht programmierbar
- ▶ Gut zu debuggen
- ▶ Sehr flexibel (mehrere Größen)

Die AVR-Familie

6/35

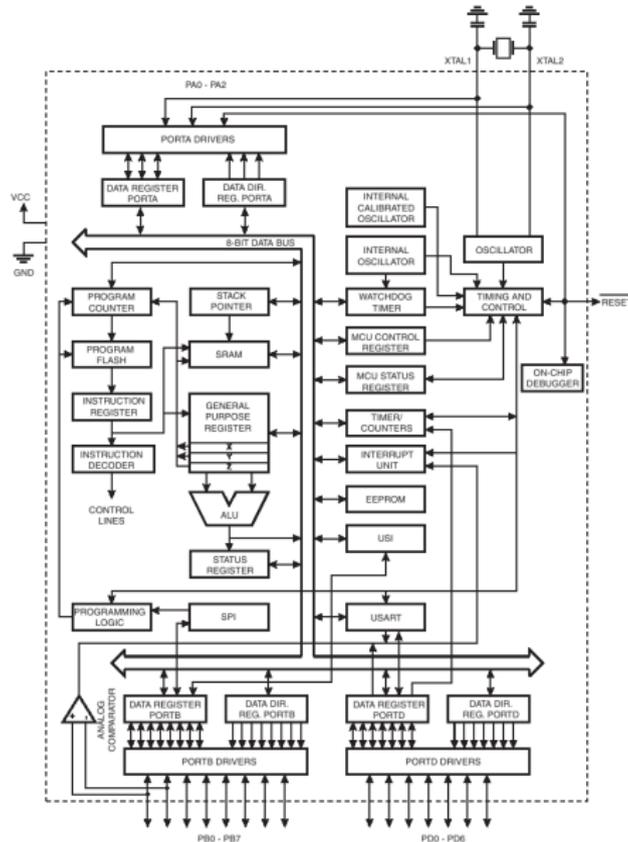
- ▶ AT90S...
 - ▶ Älteste AVR 8-Bit Generation
 - ▶ Abgekündigt
 - ▶ Noch ca. 7 aktive Typen
- ▶ ATtiny...
 - ▶ Kleine ICs (8-Pinner)
 - ▶ Stromsparend
 - ▶ 8 aktive Typen
- ▶ ATmega...
 - ▶ Teilweise sehr groß (64 Pins beim ATmega128)
 - ▶ Viel on-board Hardware
 - ▶ 15 aktive Typen

Die AVR-Familie

7/35

- ▶ Bis auf wenige Ausnahmen Opcode-kompatibel
- ▶ SPI-Programmierung immer gleich
- ▶ Möglichkeit, multiple Footprints handzuhaben

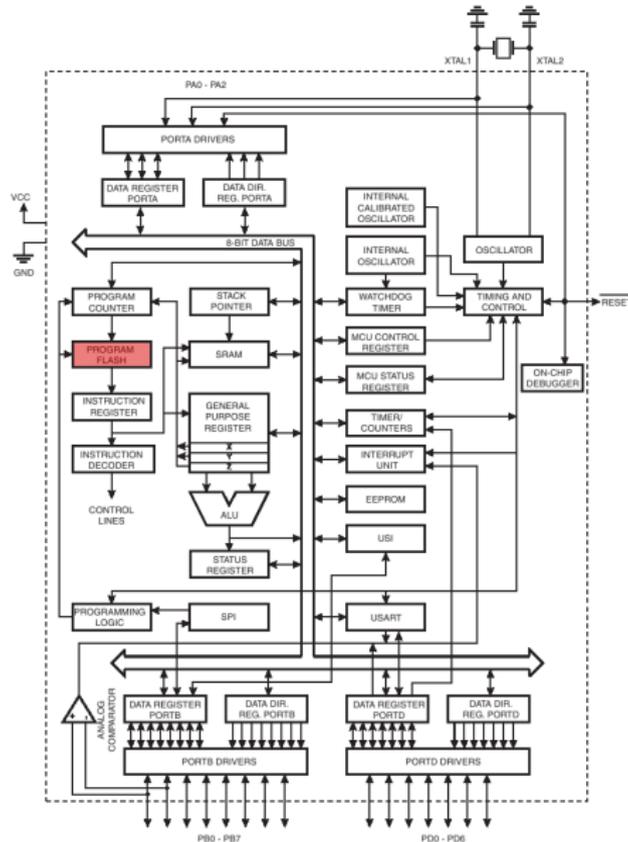
Name	Flash	RAM	EEPROM	I/O	Preis
S1200	1kB	-	64 Bytes	15	1,45
S2313	2kB	128 Bytes	128 Bytes	15	1,85
tiny11	1kB	-	-	6	1,15
tiny15	1kB	-	64 Bytes	6	1,50
tiny26	2kB	128 Bytes	128 Bytes	16	2,00
mega8	8kB	1kB	512 Bytes	23	2,75
mega16	16kB	1kB	512 Bytes	32	4,15
mega32	32kB	2kB	1kB	32	6,20
mega128	128kB	4kB	4kB	53	10,50



Flash-Speicher

10/35

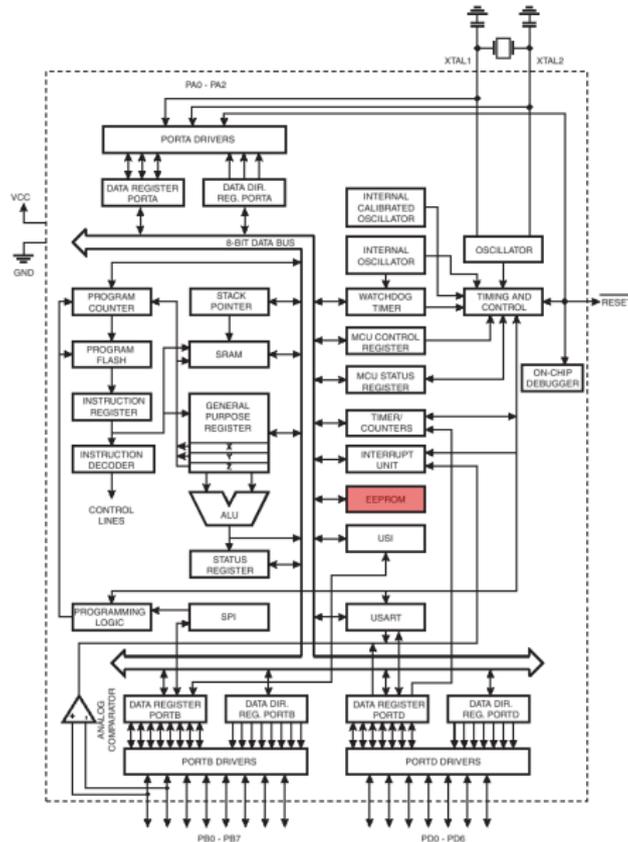
- ▶ Programmspeicher
- ▶ On-Chip
- ▶ Bis zu 10.000 wiederprogrammierbar
- ▶ Über SPI seriell programmierbar (SCK, MISO, MOSI)
- ▶ Typische Größe 1kB-128kB



Das EEPROM

12/35

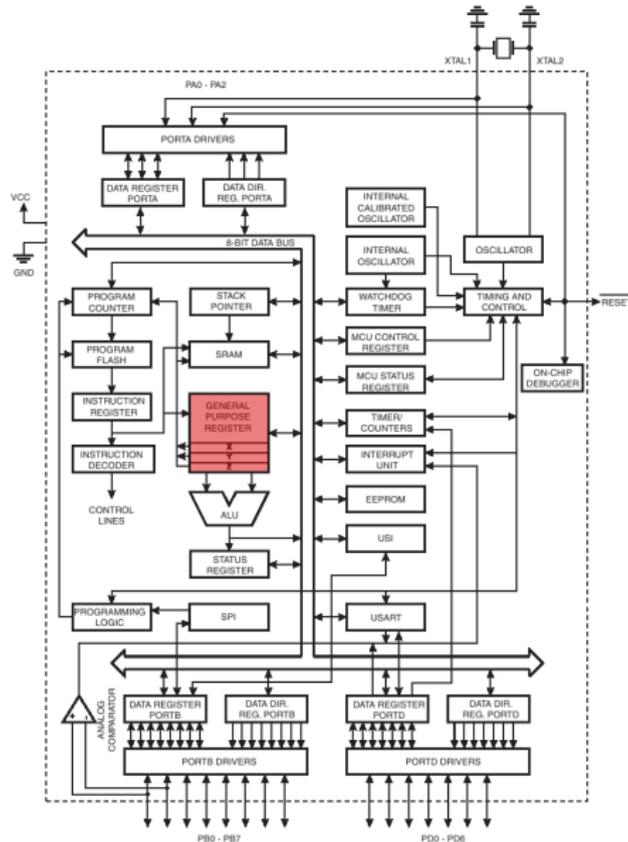
- ▶ Datenspeicher
- ▶ On-Chip
- ▶ Bis zu 100.000 wiederprogrammierbar
- ▶ Durch das Programm selbst les- und schreibbar
- ▶ Speicher für Konfigurationsdaten
- ▶ Typische Größe 128-4kB



Standardhardware

14/35

- ▶ 32 General Purpose Registers
- ▶ On-Chip Flash
- ▶ (Fast) jeder AVR hat SRAM und ein EEPROM
- ▶ Mindestens einen Timer

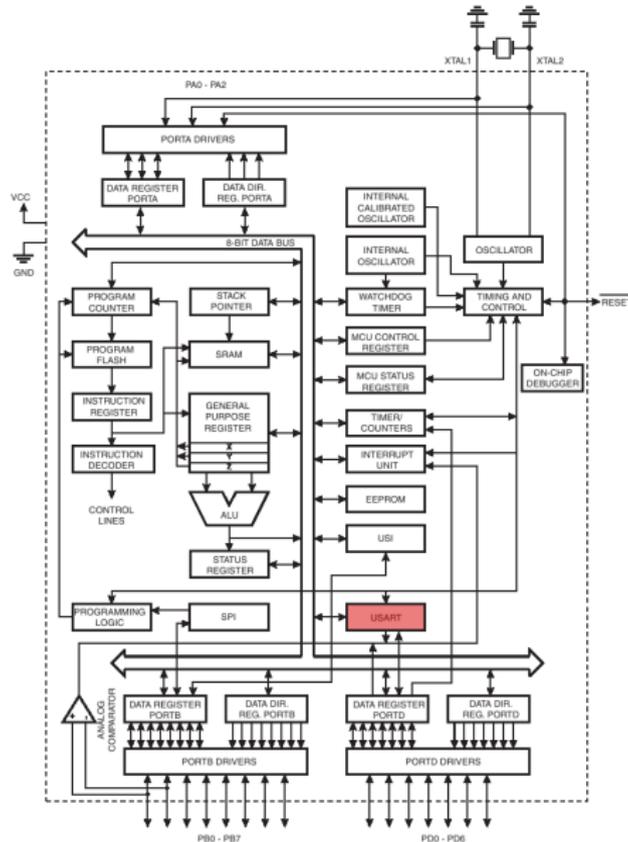


	7	0	Addr.
			0x00
			0x01
			0x02
			...
			0x0D
			0x0E
			0x0F
General Purpose Working Registers			0x10
			0x11
			...
			0x1A
			0x1B
			0x1C
			0x1D
			0x1E
			0x1F

Spezialhardware: USART

17/35

- ▶ Kommunikation mit anderen AVR's oder PC's
- ▶ Bei RS232 verwendet (Pegelwandlung erforderlich)
- ▶ Benötigt zwei Portpins: RX, TX
- ▶ Hat so gut wie jeder AVR (bis auf die tiny1n-Typen)



Spezialhardware: PWM

19/35

- ▶ Pulsbreitenmoduliertes Signal an einem Ausgabepin (variabler Duty-cycle)
- ▶ Effiziente Steuerung von bspw. Motoren oder Lampen
- ▶ Mit Tiefpaß als DAC verwendbar
- ▶ Schon kleine AVR's (ATtiny26) haben 2 PWM-Kanäle
- ▶ Bis zu 8 Kanäle (z.B. ATmega128)

Spezialhardware: ADC

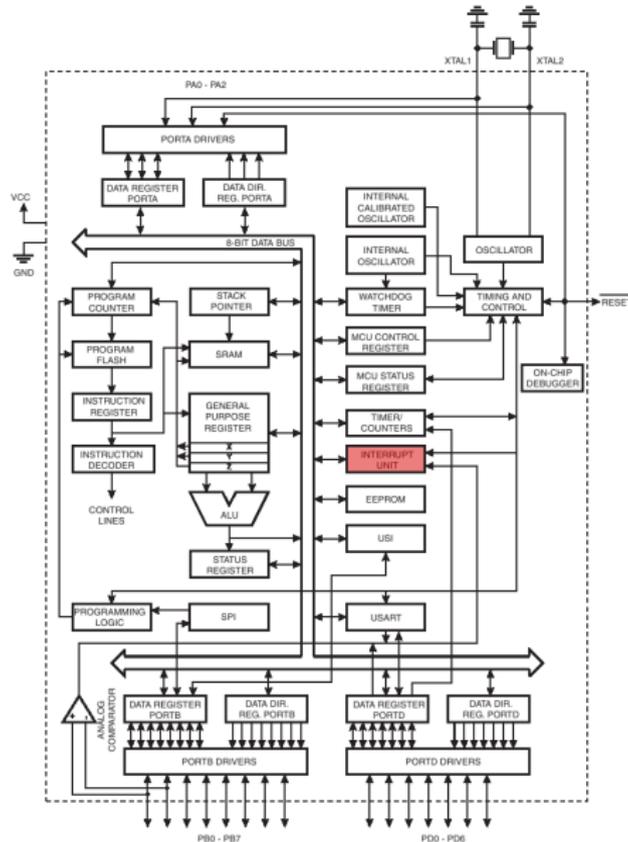
20/35

- ▶ 10 Bit ADC, 15kSPS
- ▶ Schon ein kleiner ATtiny15 hat 4 Kanäle
- ▶ Bei größeren AVR's (ATmega128) kann der Gain eingestellt werden
- ▶ ATmega128: 8 Kanäle (gemultiplext)
- ▶ Spart insgesamt ein bis zwei ICs (Verstärker, Wandler)

Interrupts

21/35

- ▶ Asynchrone Programmunterbrechung
- ▶ Einsprungpunkt im Programm: „Vektor“
- ▶ Verschiedene Quellen für Interrupts, z.B.:
 - ▶ Zeitgeber
 - ▶ Datenempfang (z.B. über das USART)
 - ▶ Externe Quelle

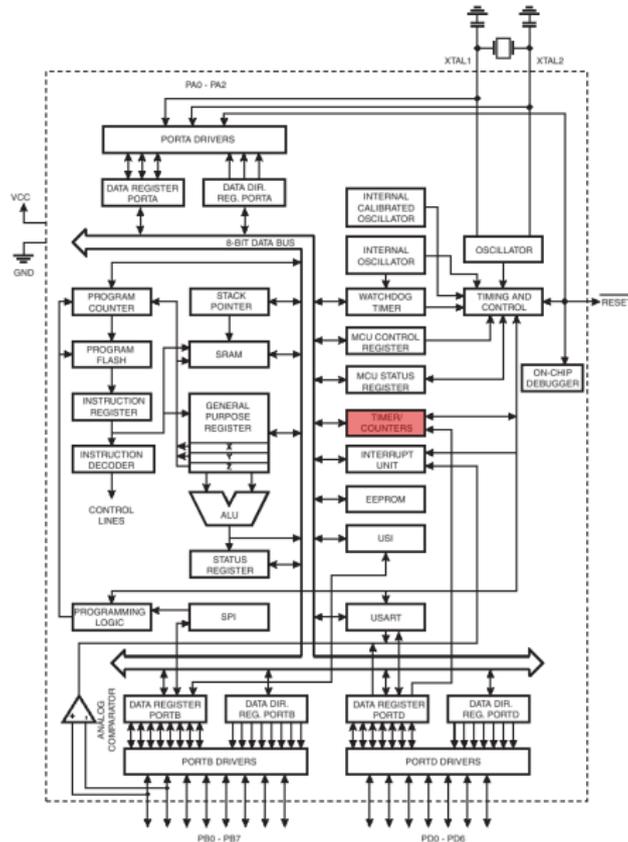


Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset
2	0x0001	INT0	External Interrupt Request 0
3	0x0002	INT1	External Interrupt Request 1
4	0x0003	TIMER1 CAPT	Timer/Counter1 Capture Event
5	0x0004	TIMER1 COMPA	Timer/Counter1 Compare Match A
6	0x0005	TIMER1 OVF	Timer/Counter1 Overflow
7	0x0006	TIMER0 OVF	Timer/Counter0 Overflow
8	0x0007	USART0, RX	USART0, Rx Complete
9	0x0008	USART0, UDRE	USART0 Data Register Empty
10	0x0009	USART0, TX	USART0, Tx Complete
11	0x000A	ANALOG COMP	Analog Comparator
12	0x000B	PCINT	Pin Change Interrupt
13	0x000C	TIMER1 COMPB	Timer/Counter1 Compare Match B
14	0x000D	TIMER0 COMPA	Timer/Counter0 Compare Match A
15	0x000E	TIMER0 COMPB	Timer/Counter0 Compare Match B
16	0x000F	USI START	USI Start Condition
17	0x0010	USI OVERFLOW	USI Overflow
18	0x0011	EE READY	EEPROM Ready
19	0x0012	WDT OVERFLOW	Watchdog Timer Overflow

Interrupts: Timer

24/35

- ▶ Sehr feingranulare Zeiteinstellung möglich
- ▶ Präemptives Multitasking durch Timerinterrupts
- ▶ „Time-Triggered“ Tasksplanung möglich



Interrupts: Ereignisse

26/35

- ▶ „Event-Triggered“ Interrupts
- ▶ Beispiel: „Daten empfangen“ oder „Bereit zum Senden“
- ▶ Externe Signale (Flanken) können Programm unterbrechen
- ▶ Vorteil: Kein Polling, Erkennung der Flanke in Hardware (Latch)
- ▶ Schalten der MCU in den Sleep-Mode, „Wake on Interrupt“

Die offene AVR-Toolchain

27/35

- ▶ **Assemblierung:** `avr-as`
- ▶ **Compilierung:** `avr-gcc`
- ▶ **Flashing:** `avrdude`

Der Assembler

28/35

- ▶ Makroassembler ähnlich dem normalen gcc-as
- ▶ Übersetzt Assembler-Input in Objektdatei-Output
- ▶ Danach Linking mit `avr-ld` notwendig
- ▶ Extraktion der Binärdaten mit `avr-objdump`

Der Compiler

29/35

- ▶ C oder C++-Compiler
- ▶ Schön standardisierte Header-Files (nach einigen Änderungen)
- ▶ Kompilierung mit `avr-gcc` erzeugt direkt das Outputfile
- ▶ Handling multipler Objekte wie vom `gcc` gewohnt

Das Flashprogramm

30/35

- ▶ Intuitive Kommandozeilenbedienung
- ▶ Gut aus Makefiles heraus zu verwenden
- ▶ Unterstützt auch problematische AVR's
- ▶ Unterstützt alle gängigen Flashgeräte (inkl. selbstgebaute)

BTnodes

31/35

- ▶ Basierend auf ATmega128, 8MHz
- ▶ Zusätzlich BT-Hardware auf dem Board
- ▶ Viele Interfaces der MCU extern herausgeführt (I2C, UART, ADC, ...)
- ▶ BT-Stack als OpenSource in C verfügbar



BTnodes

33/35

- ▶ Entwicklung der ETH Zürich
- ▶ Vorteile:
 - ▶ Keine Hardwareentwicklung
 - ▶ Funktioniert Out-of-the-box
- ▶ Developer-Kit (inkl. Programmiergerät, Software, 2 BTnodes) für 520 EUR zu haben
- ▶ Anwendungsgebiete: Biologie, verteilte Systeme

Fazit

34/35

- ▶ Kleine 8-Bit RISC MCUs
- ▶ Sehr flexibel, viel Hardware auf dem Chip
- ▶ Wenige externe Komponenten benötigt
- ▶ Ausgesprochen gute Wahl, um mit MCUs herumzuprobieren
- ▶ Ein riesen Spielspaß!

Gibt es noch...

35/35

Fragen?