

EMES: Eigenschaften mobiler und eingebetteter Systeme

Prozessoren für mobile und eingebettete Systeme II: Die AVR-Architektur

Dipl. Inf. Jan Richling
Wintersemester 2005/2006



Die AVR-Architektur

- Warum AVR (in dieser VL)?
 - Angesiedelt im extremen Low-Cost-Bereich
 - Interessante Controller-Familie mit hoher Skalierbarkeit
 - Systematisch entwickelte Architektur
- Quellen für diese VL:
 - Online verfügbare Dokumente von Atmel: <http://www.atmel.com>
 - Code- und Schaltungsbeispiele mit freundlicher Genehmigung von <http://www.mikrocontroller.net>



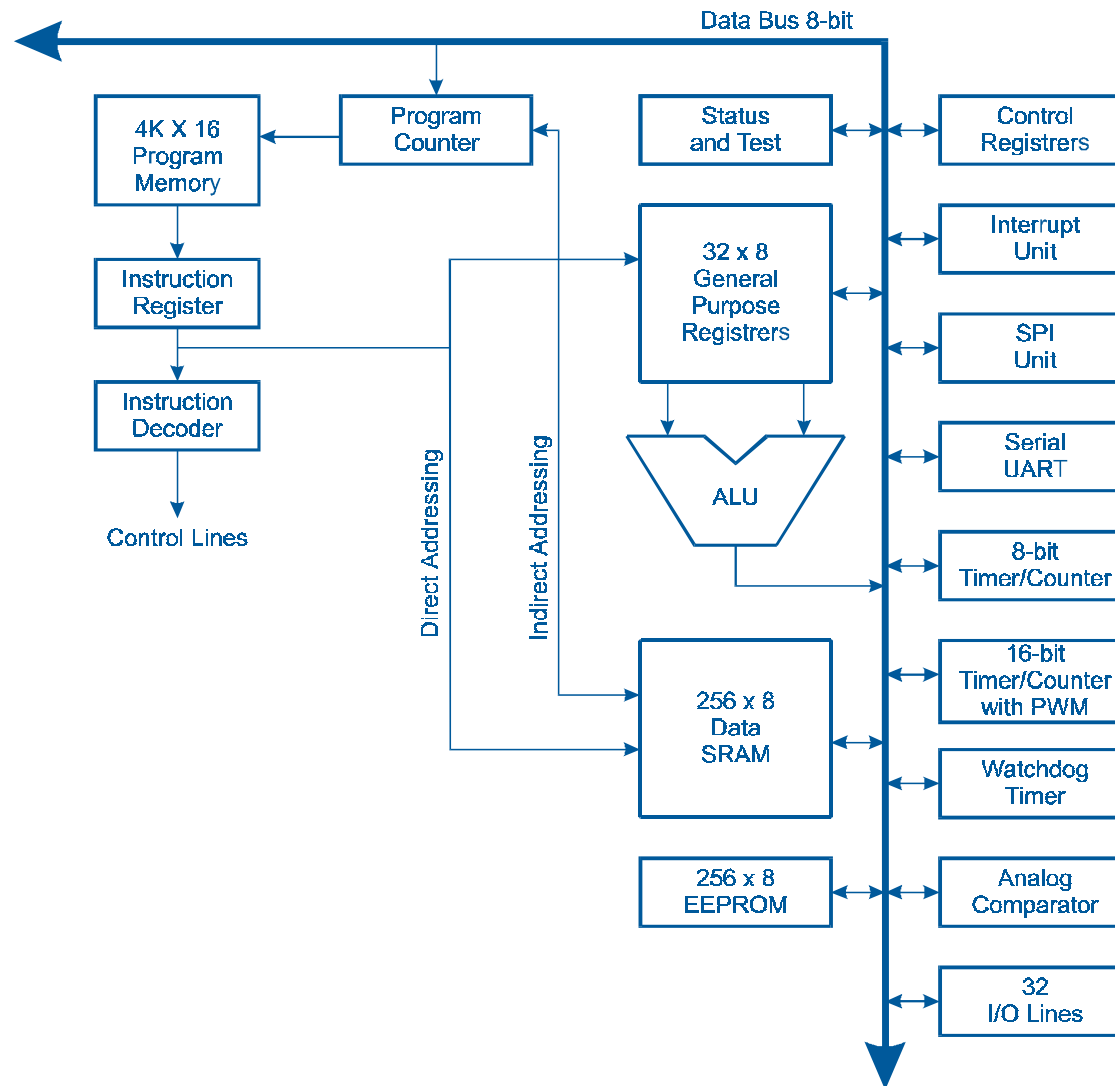
Was ist AVR?

- Eine RISC-Architektur
 - Entwickelt in den 90ern von Alf Egil Bogen und Vegard Wollan am Norwegian Institute of Technology
 - Weiterentwickelt bei einer Tochterfirma von Atmel, die von den beiden Entwicklern gegründet wurde
- Eine Familie von Mikrocontrollern der Firma Atmel
 - Basieren auf dem (weitgehend) gleichen Kern
 - Unterschiedliche Speichergrößen
 - Unterschiedliche I/O-Peripherie
- Herkunft und Bedeutung des Namen AVR sind nicht komplett bekannt
 - Gerüchte:
 - Advanced Virtual RISC
 - Alf Vegard RISC

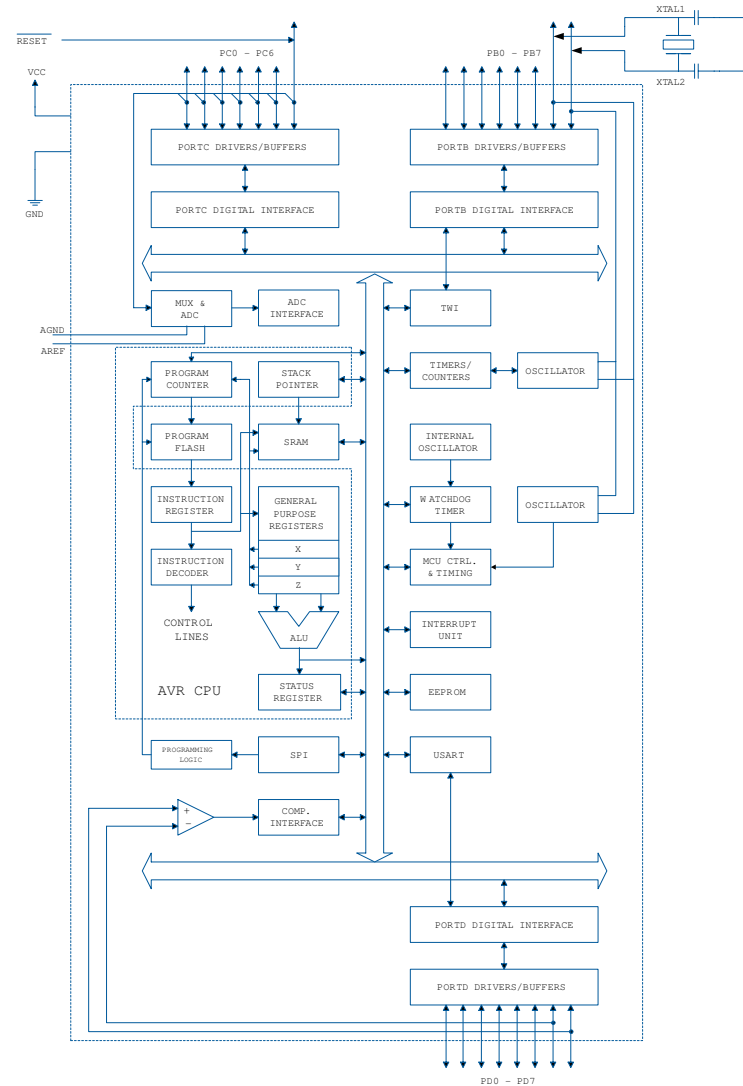
Parameter im Überblick

- Harvard-Architektur
- RISC-Design (mit Einschränkungen)
- Load/Store
- 8 Bit Verarbeitungsbreite
- Wortadressierung (16 Bit)
- 32 Register (in den Speicher abgebildet auf 0x00 bis 0x1F)
- 2-Adress-Maschine
- 130 Instruktionen (ATmega)
- 16 Bit Befehlsformat, wenige Instruktionen brauchen 32 Bit
- Fast alle Instruktionen in einem Zyklus bei bis zu 20 MHz Takt
- Orthogonaler Befehlssatz

Architektur I (Beispiel: AT90S8414)



Architektur II (Beispiel: ATmega 8)





Architektur III

- 32 Register (nächste Folie)
- Programmspeicher: 1 bis 256 KByte Flashspeicher auf dem Chip
Adressierbar: bis 8 MByte durch 22 Bit-Programmzähler
- Datenspeicher:
 - 0 bis 8 KByte SRAM auf dem Chip
 - 0 bis 4 KByte EEPROM auf dem Chip
- ALU:
 - Registerbasierte Operationen in einem Zyklus
 - Keinen direkten Zugriff auf Speicher
 - Multiplikation optional (ATmega)
 - Keine Division
- Keine Pipeline

Architektur IV: Register und I/O

- R0-R31 als General Purpose Register mit einigen Ausnahmen:
 - Benutzung der letzten 6 Register als 16-Bit Speicherpointer:
 - * X-Pointer: R26 und R27
 - * Y-Pointer: R28 und R29
 - * Z-Pointer: R30 und R31
 - Befehle mit immediate Werten (6 Bit breit) können nur R16 bis R31 nutzen
 - add und sub auf 16-Bit-Worten sind nur mit R24-R31 möglich
- Breite Vielfalt an I/O integrierbar:
 - 64 I/O-Ports (jeweils 8 Bit) direkt ansprechbar
 - 192 weitere I/O-Ports über Daten-Adressierung



Befehlssatz I

- Orientiert an C als Hochsprache
- Compilerentwicklung noch vor Ende der Architekturentwicklung begonnen
- Architektur in engem Kontakt zu Compilerbauern optimiert
 - Adressierungsarten
 - 3 Speicherpointer statt 2
 - Direkte Adressierung statt seitenorientierter direkter Adressierung
 - Subtraktionsbefehl statt Addition für Immediate
 - Nichtzerstörende Vergleichsoperation für lange Operanden
- “Skip” Operationen — bedingte Ausführung ähnlich ARM für einige Befehle
- Viele Befehle zur Manipulation von Bits

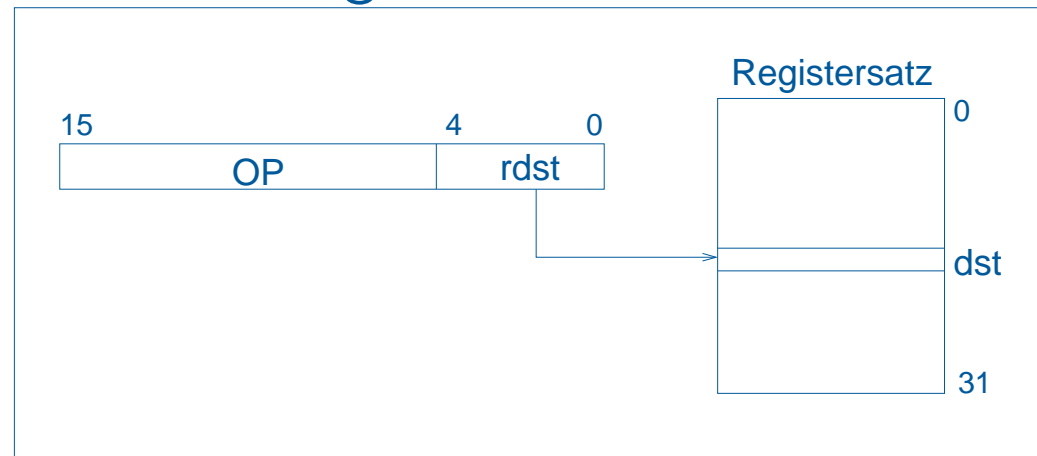


Befehlssatz II

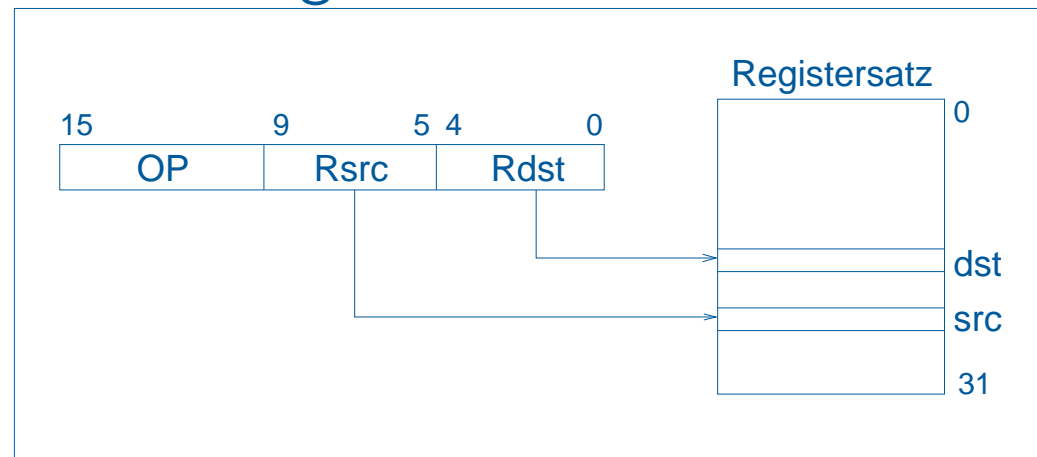
- Befehle zur Flußsteuerung
- Logische Befehle
- Befehle zur Bit-Manipulation
- Arithmetische Befehle
- Load/Store-Befehle
- Verschiedenes

Befehlsformate und Adressierungsarten I

- Register direkt mit einem Register



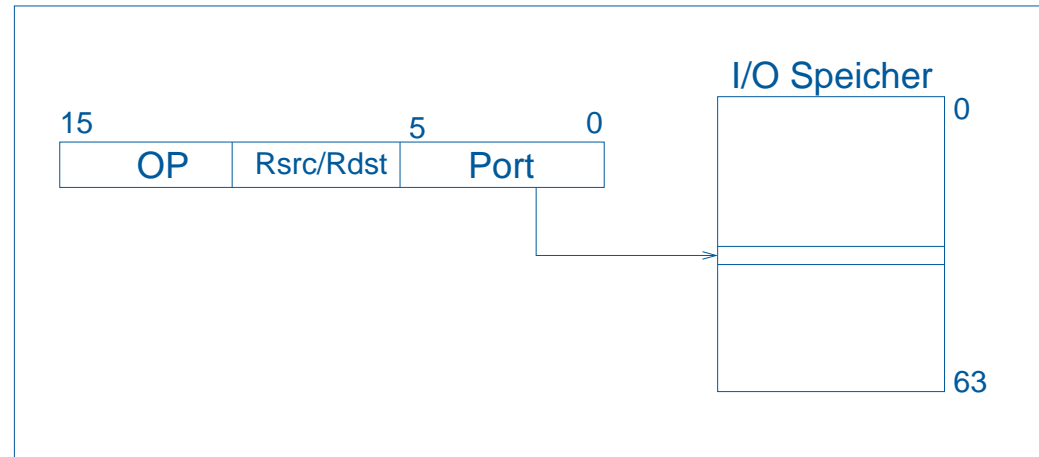
- Register direkt mit zwei Registern SRC und DST



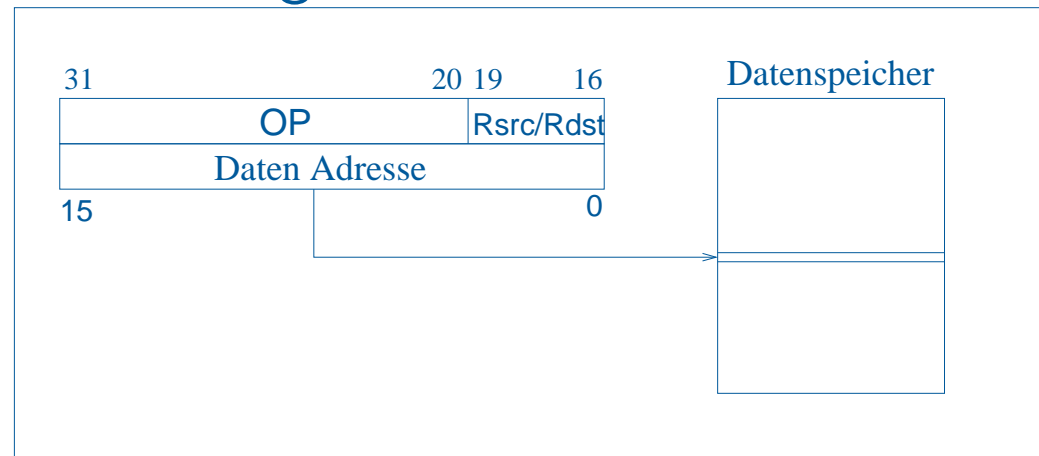
Befehlsformate und Adressierungsarten

II

- Direkte I/O-Adressierung



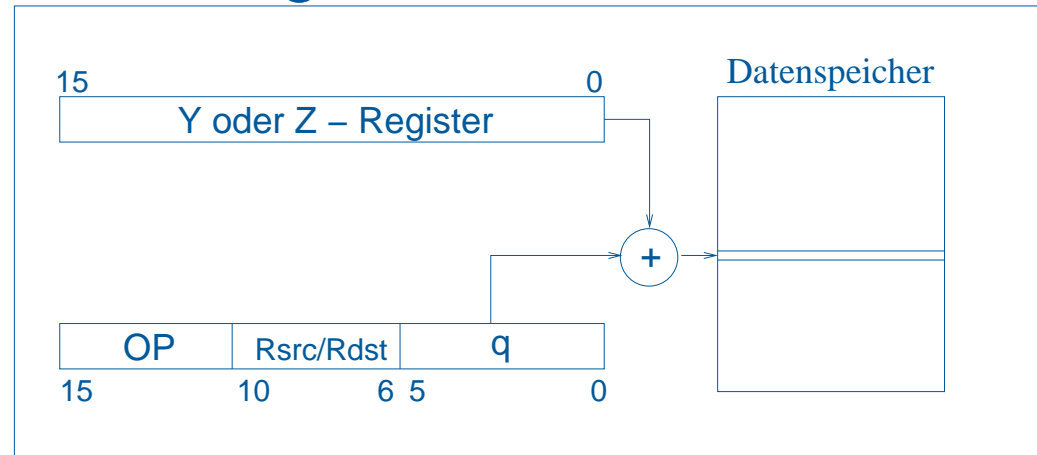
- Direkte Daten-Adressierung



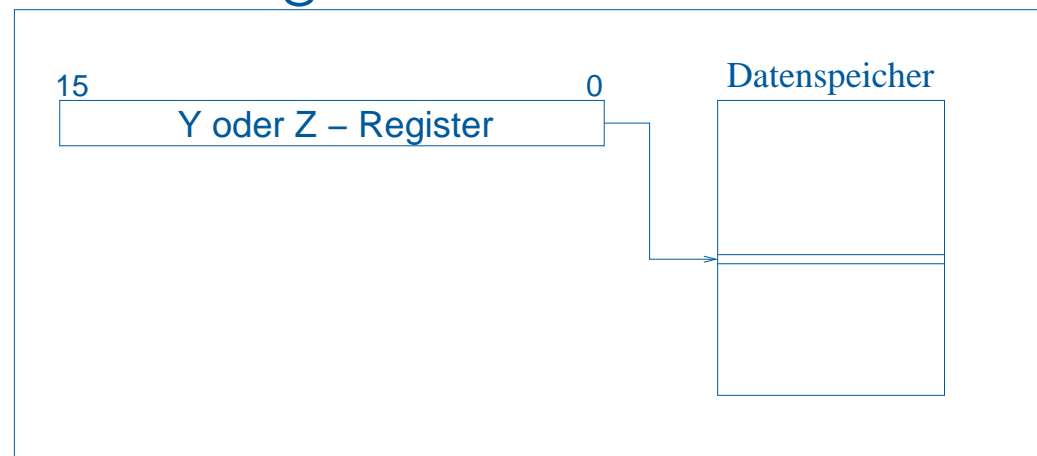
Befehlsformate und Adressierungsarten



- Indirekte Datenadressierung mit Offset



- Indirekte Datenadressierung



Befehlsformate und Adressierungsarten

IV

- Indirekte Datenadressierung mit Prä-Dekrement
- Indirekte Datenadressierung mit Post-Inkrement
- Konstante Adressierung auf dem Programmspeicher
- Programmspeicher mit Post-Inkrement
- Direkte Programmspeicher-Adressierung bei JMP und CALL (22 Bit)
- Indirekte Programmspeicher-Adressierung bei IJMP und ICALL (16 Bit)
- Relative Programmspeicher-Adressierung bei RJMP und RCALL (12 Bit)

- Architektur erlaubt 256 I/O Ports, davon 64 direkt ansprechbar
- Modellabhängige Nutzung der Ports für z.B.:
 - Digitale I/O-Ports (jeder Pin entspricht einem Bit)
Umschaltbar in der Datenrichtung
 - AD-Wandler mit 8 oder 10 Bit Auflösung
 - RS232-Schnittstellen
 - TWI (two wire interface)
 - PWM-Ausgänge (Pulsweitenmodulation)
 - Realtime-Zähler
 - Watchdog
 - Analog Comparator
 - Anbindung externen Speichers
 - ISP (in system programming)

Atmel Corporation: Mehrere Baureihen

- AT90****
 - Erste Serie, nicht mehr für neue Designs empfohlen
- tinyAVR (ATtiny11 bis ATtiny45)
 - Für einfachste Anforderungen mit geringen Kosten
 - Extrem kleine Bauform (ab 8 Pins)
- megaAVR (ATmega8 bis ATmega256)
 - Kosteneffektive MCUs mit sehr vielen I/O-Pins
 - Viel Speicher
 - Flexible I/O-Möglichkeiten
 - JTAG zur Programmierung

Beispiel: ATmega8 I

- 8 KB Flash, 512 Byte EEPROM, 1 KB SRAM
- 2 8-Bit-Zähler/Zeitgeber
- 1 16-Bit-Zähler/Zeitgeber
- Echtzeitähler mit eigenem Oszillator
- 3 PWM-Kanäle
- 6 oder 8 AD-Wandler mit 10 Bit Auflösung
- Byteorientiertes TWI

Beispiel: ATmega8 II

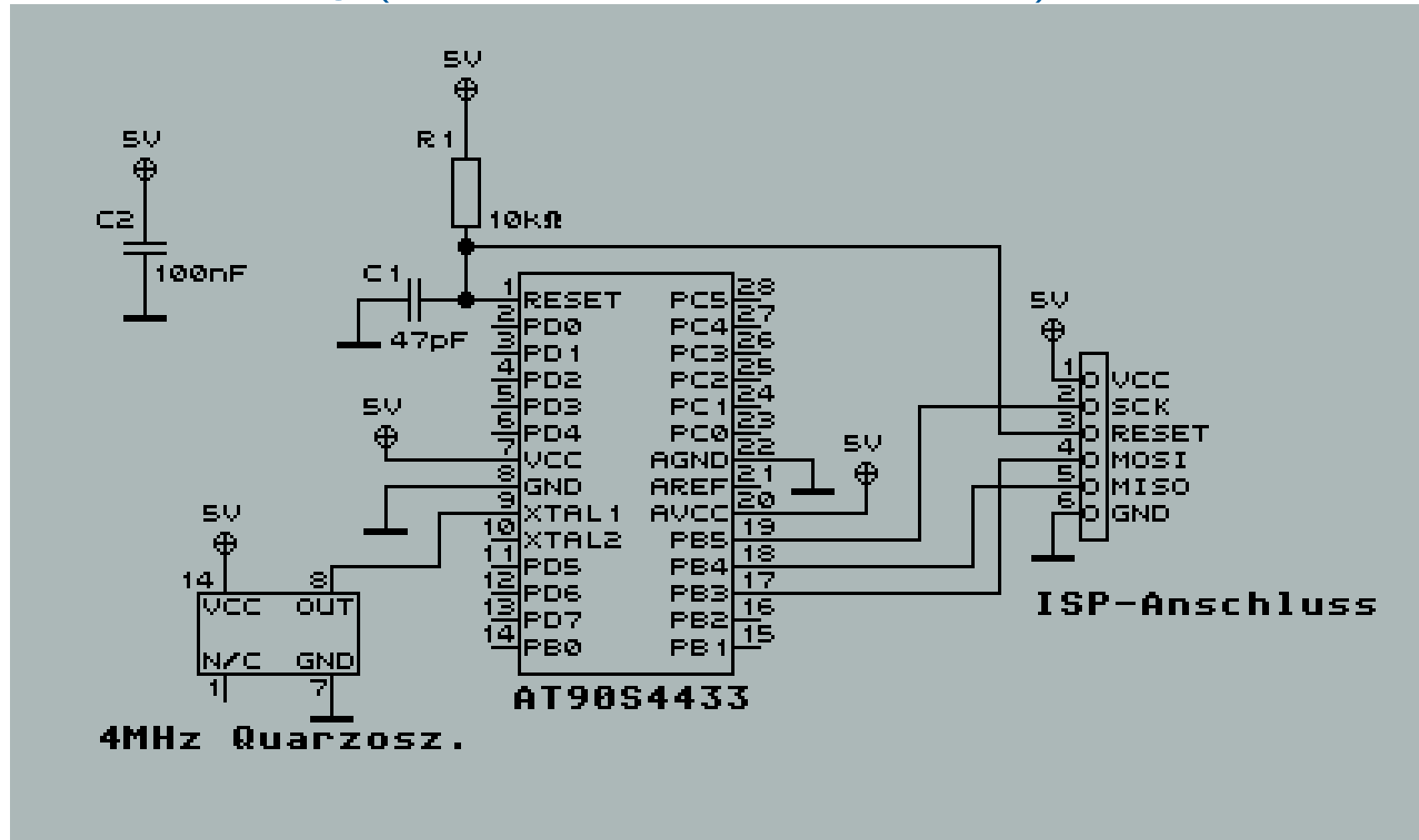
- Programmierbarer Watchdog mit eigenem Oszillator
- Analog Comparator
- 5 Energiespar-Modi
- 23 programmierbare I/O-Leitungen
- 28- oder 32-poliges Gehäuse
- bis zu 8 MHz bei 2.7-5.5V (ATmega8L) oder 16 MHz bei 4.5-5.5V (ATmega8)
- Stromverbrauch bei 4 MHz und 25 Grad C:
 - Aktiv: 3.6 mA
 - Idle: 1.0 mA
 - Power-down: 0.0005 mA

Anwendung: Hardware I

- Gut geeignet für nichtindustrielle Bastelprojekte:
 - DIL-Gehäuse (bis ATmega32)
 - Minimale Außenbeschaltung
 - Einfach zu bauende Programmierschnittstellen
 - Frei verfügbare Entwicklungstools
- Benötigt:
 - Stromversorgung
 - Reset-Beschaltung
 - Optional Takt, falls kein interner Oszillator oder falls höhere Anforderungen an zeitliche Auflösung

Anwendung: Hardware II

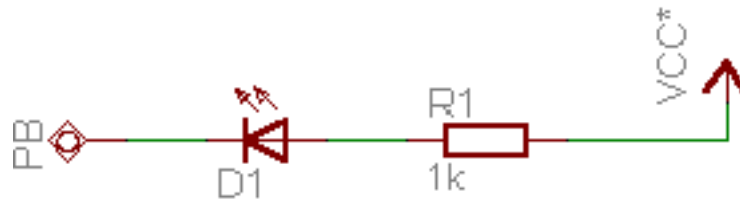
Generelle Schaltung (am Beispiel des AT90S4433):



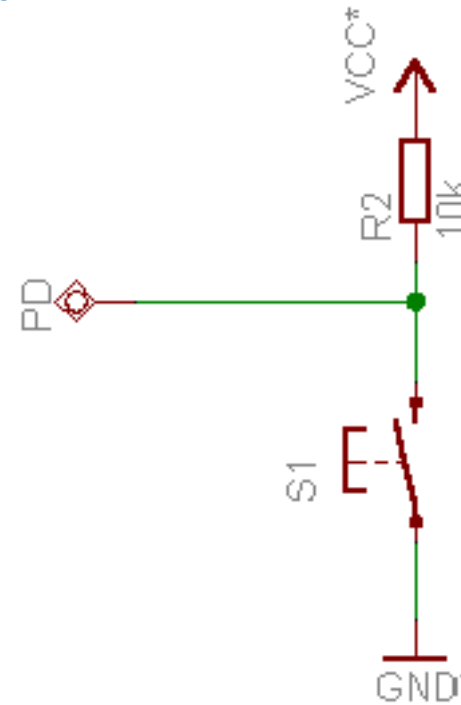
Anwendung: Hardware III

Anbindung von einfachen I/O-Komponenten:

LED:



Taster:



Anwendung: Software

- Programmierung des chipinternen Flash über ISP (in system programming) mittels eines einfachen Parallel-Port-Adapters
- Frei verfügbare Software von Atmel zur Assembler-Programmierung: AVR-Studio
- Kommerzielle Compiler für Basic, C und weitere
- Opensource-Tools für Assembler und C frei verfügbar
 - avr-gcc für Unix und Win32/Cygwin
 - binutils (inklusive Assembler)
 - Programmierertools für den Download auf die MCU

Code-Beispiel I/O mit Assembler

```
.include "4433def.inc"           ;bzw. m8def.inc

ldi r16, 0xFF
out DDRB, r16                    ;Port B durch Ausgabe von 0xFF ins
                                ;Richtungsregister DDRB als Ausgang

ldi r16, 0x00
out DDRD, r16                    ;Port D durch Ausgabe von 0x00 ins
                                ;Richtungsregister DDRD als Eingang

loop:
in r16, PIND                     ;an Port D anliegende Werte (Taster)
                                ;nach r16 einlesen

out PORTB, r16                   ;Inhalt von r16 an Port B ausgeben

rjmp loop                        ;Sprung zu "loop:" -> Endlosschleife
```

Code-Beispiel: I/O mit C (avr-gcc)

```
#include <avr/io.h>
```

```
int main(void)
```

```
{
```

```
    DDRB = 0xff; // Port B als Ausgang
```

```
    DDRD = 0;    // Port D als Eingang
```

```
    for (;;) {
```

```
        PORTB = PIND;
```

```
    }
```

```
}
```



Eigene Projekte

- Kostengünstige Hardware bei hoher Leistung und einfacher Aufbau erlauben auch komplexe Anwendungen mit vertretbarem Aufwand (“Bastelprojekte”)
 - Propeller-Clock und Verwandte
 - Embedded Webserver
 - CAN-Schaltungen
 - Haus-Automatisierung
 - und viele weitere (siehe Google)
- Einstiegspunkt für tiefergehendes Wissen (beispielsweise):
www.mikrocontroller.net