Microcontroller-Grundlagen

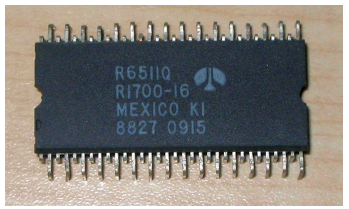
Timo Cramer

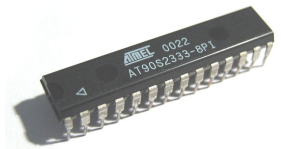
05. November 2015

Definition

Was sind Microcontroller?

- Ein-Chip-Computersysteme
  - Prozessor, RAM, Peripherie, etc. in einem
- meist wenige KB Speicher
- sehr geringer Stromverbrauch
- sehr geringer Preis
- Echtzeitfähigkeit
- direkter Hardware-Zugriff





[8, 7, 2]

Motivation

Warum halte ich einen Vortrag darüber?

- Sie werden *überall* verwendet!
  - Unterhaltungselektronik
  - KFZ
  - Chipkarten
  - ...

Motivation

Warum halte ich einen Vortrag darüber?

- Sie werden *überall* verwendet!
  - Unterhaltungselektronik
  - KFZ
  - Chipkarten
  - ...

Wikipedia – Microcontroller

A typical home in a developed country is likely to have only four general-purpose microprocessors but around three dozen microcontrollers. A typical mid-range automobile has as many as 30 or more microcontrollers.

Motivation

Warum halte ich einen Vortrag darüber?

- Sie werden *überall* verwendet!
  - Unterhaltungselektronik
  - KFZ
  - Chipkarten
  - ...

Wikipedia – Microcontroller

A typical home in a developed country is likely to have only four general-purpose microprocessors but around three dozen microcontrollers. A typical mid-range automobile has as many as 30 or more microcontrollers.

- Trotzdem werden sie selten in Lehrveranstaltungen angesprochen

Architekturen

Es gibt viele konkurrierende Hersteller und Architekturen

- Harvard- vs. Von-Neumann-Architektur
- RISC vs. CISC
- 4-, 8-, 16-, 32-Bit
- Hardware-Multiplikation etc.

Architekturen

Es gibt viele konkurrierende Hersteller und Architekturen

- Harvard- vs. Von-Neumann-Architektur
- RISC vs. CISC
- 4-, 8-, 16-, 32-Bit
- Hardware-Multiplikation etc.

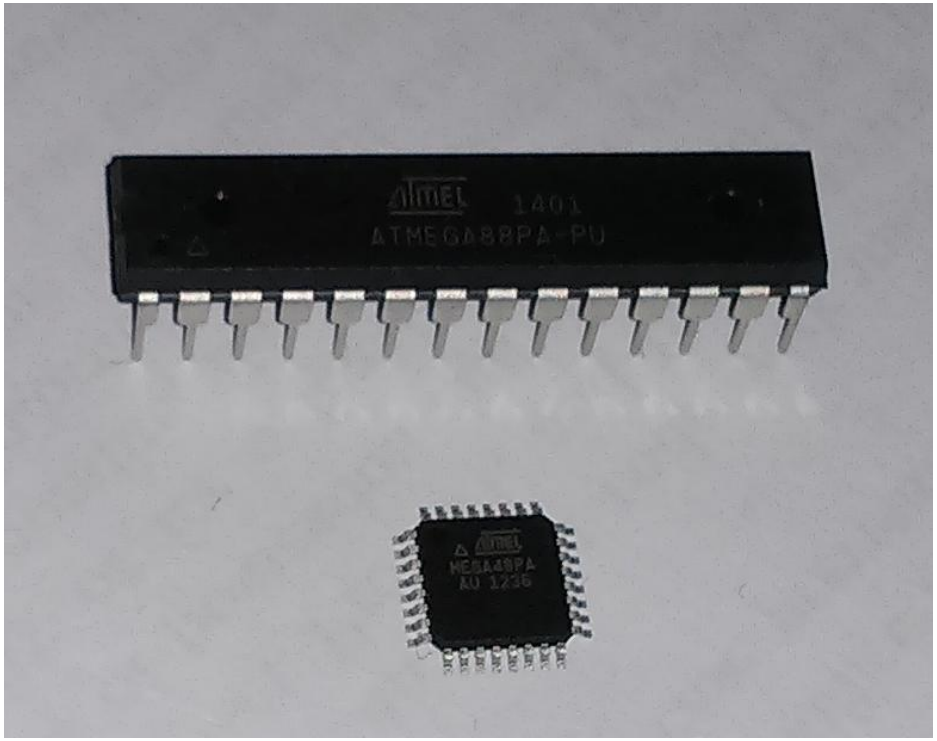
Beispiele:

- Texas Instruments mit MSP430
- Microchip mit PIC
- Atmel mit AVR
- viele, viele mehr

Beispiel: ATmega8A[4]

Speicher & CPU

- 8 KB Flash-Speicher
- 1 KB RAM
- 512 Byte EEPROM
- 8-Bit AVR-Architektur (RISC)

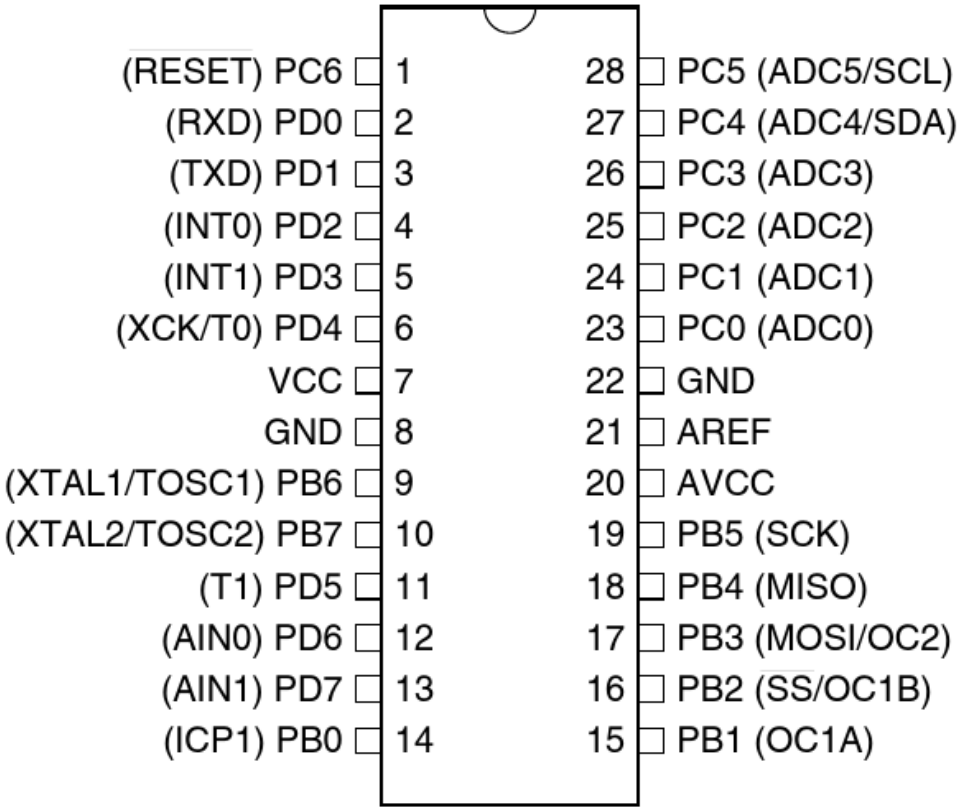


(Abbildung ähnlich)

Beispiel: ATmega8A[4]

Peripherie

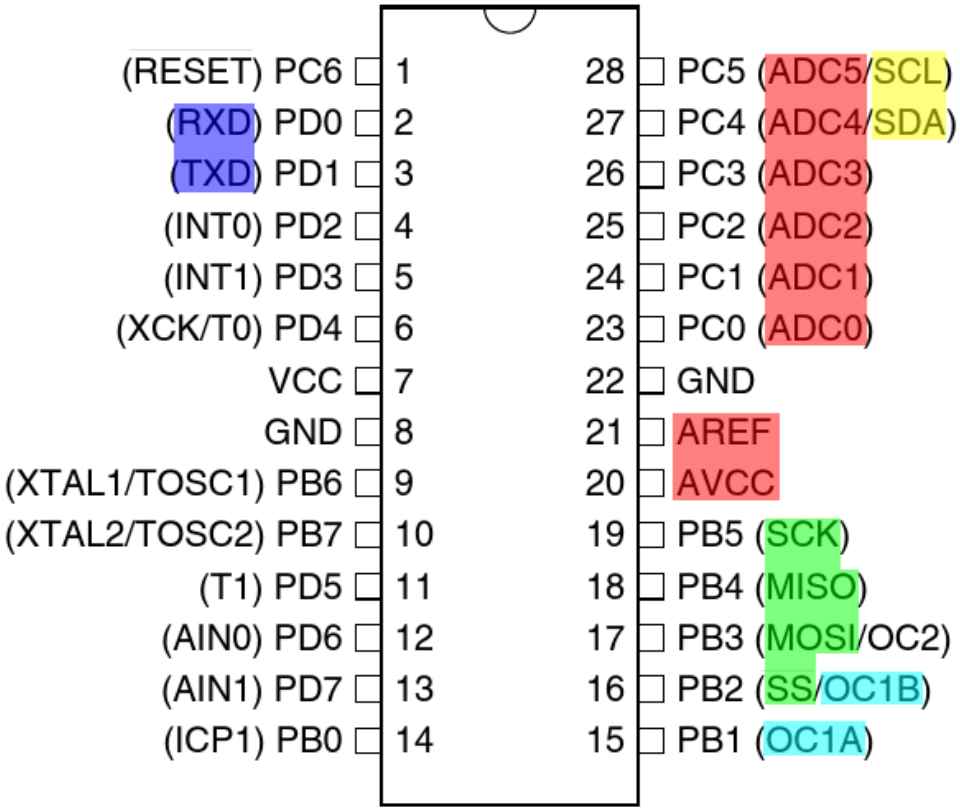
- 28–32 Pins
- GPIO
- I^2C
- SPI
- UART
- A/D-Wandler
- PWM



Beispiel: ATmega8A[4]

Peripherie

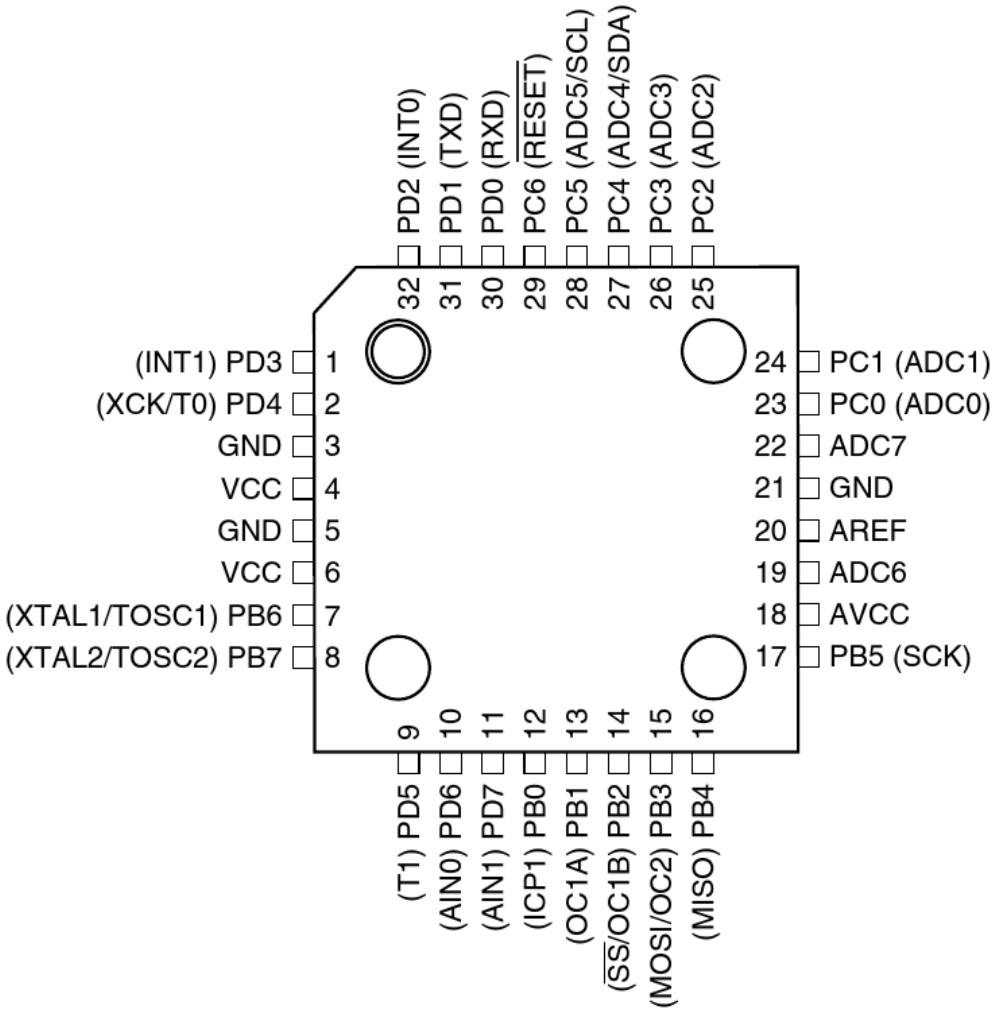
- 28–32 Pins
- GPIO
- I^2C
- SPI
- UART
- A/D-Wandler
- PWM



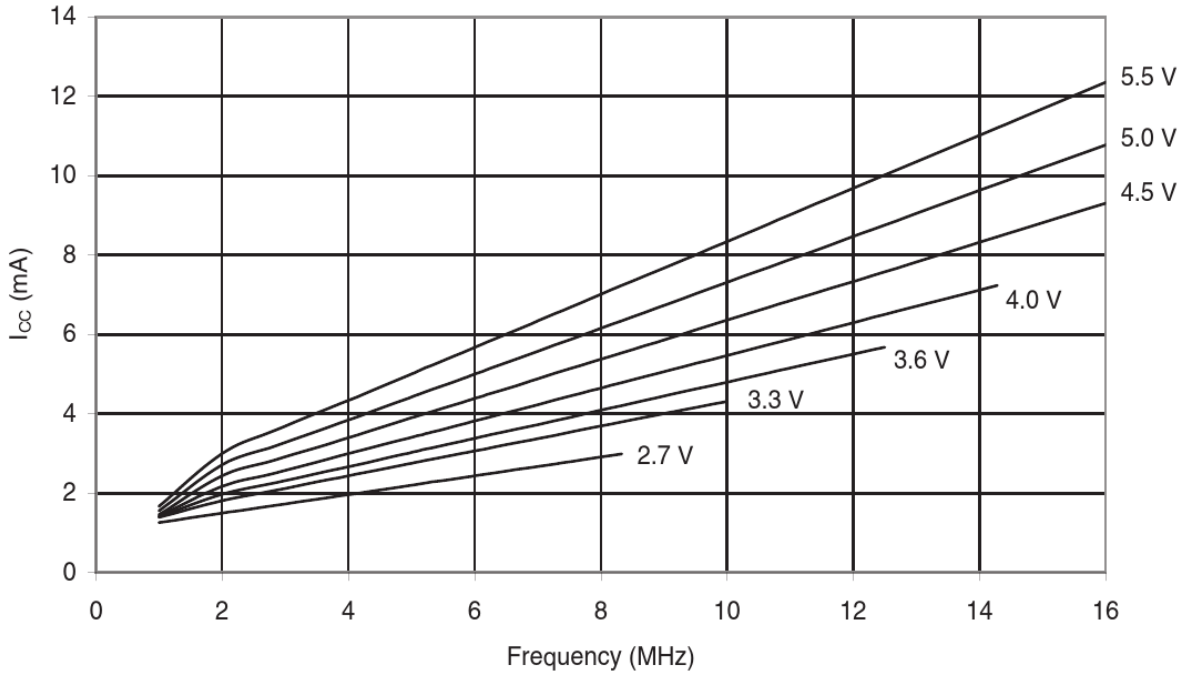
Beispiel: ATmega8A[4]

Peripherie

- 28–32 Pins
- GPIO
- I^2C
- SPI
- UART
- A/D-Wandler
- PWM



Beispiel: ATmega8A[4]



Energie

- 2.7–5 V Betriebsspannung
- bis zu 16 MHz
- 5 Schlafmodi

Beispiel:
3.6 mA Stromverbrauch im active mode bei 4 MHz und 3 V

Programmierung

- Hardware-nahe Sprachen
  - Assembler
  - C
  - C++
- Einschränkungen
  - keine Speicherverwaltung mit `malloc` oder `new`
  - keine Dateioperationen, Streams etc.
  - $\Rightarrow$ keine komplette libc
- Memory-Mapped I/O
- häufig kommerzielle Compiler

Bit-Operationen

- viele Informationen sind in einzelnen Bits gespeichert
- Speicherzugriffe erfolgen in Byte-Granularität
- Setzen/Löschen von Bits mit Hilfe von |, & und ~

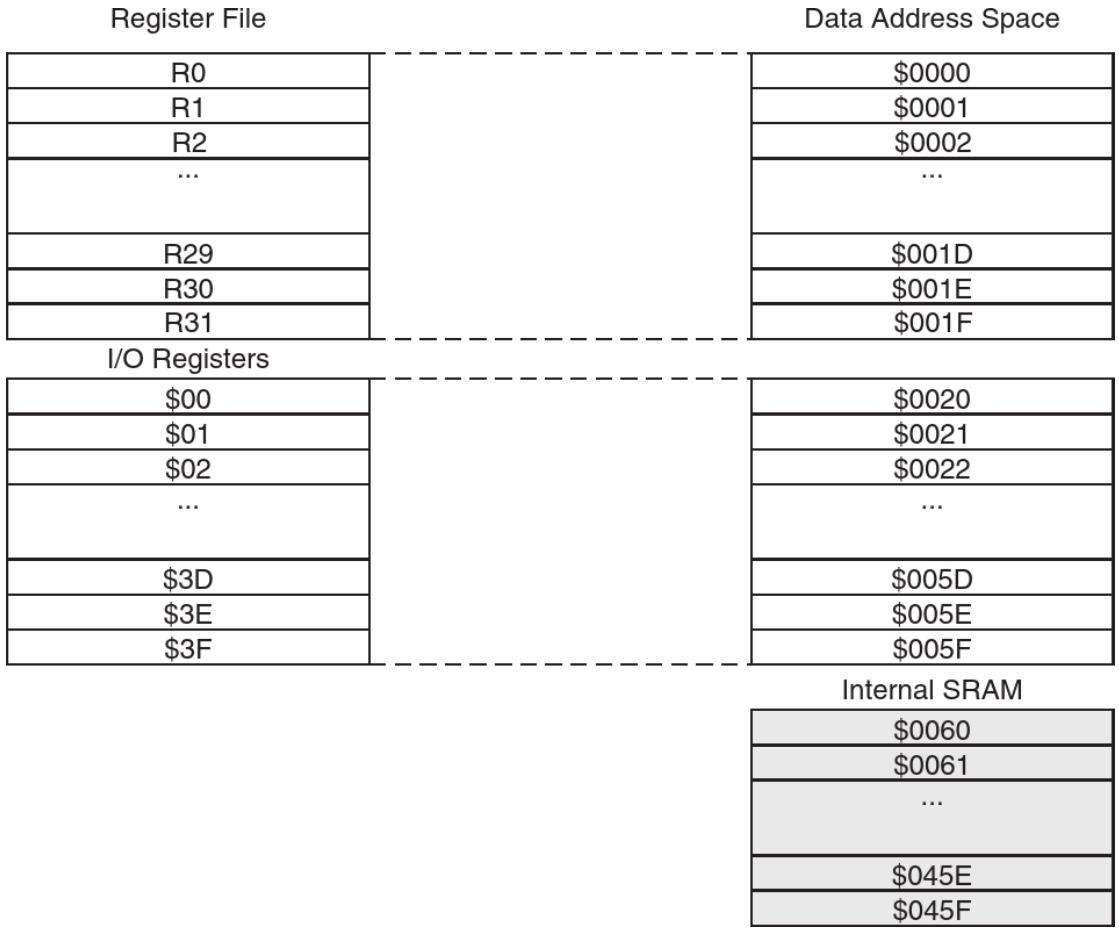
	0x15	00010101
|	0x0c	00001100
=	0x1d	00011101

	0x15	00010101
&	0x0c	00001100
=	0x04	00000100

- Häufig:
  - Setzen von Bit n: `var |= (1 << n)`
  - Löschen von Bit n: `var &= ~(1 << n)`

Memory-Mapped I/O

- Einblendung von Registern im Adressraum des Hauptspeichers



Memory-Mapped I/O & Bit-Operationen

- zur einfacheren Programmierung: `#defines` in `avr/io.h`
- Beispiel: Setzen eines Bits in PORTB

PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

```
PORTB |= (1 << PB3);
// oder
PORTB |= _BV(PB3);
```

Memory-Mapped I/O & Bit-Operationen

- Beispiel: Lesen eines Bits in UCSRA

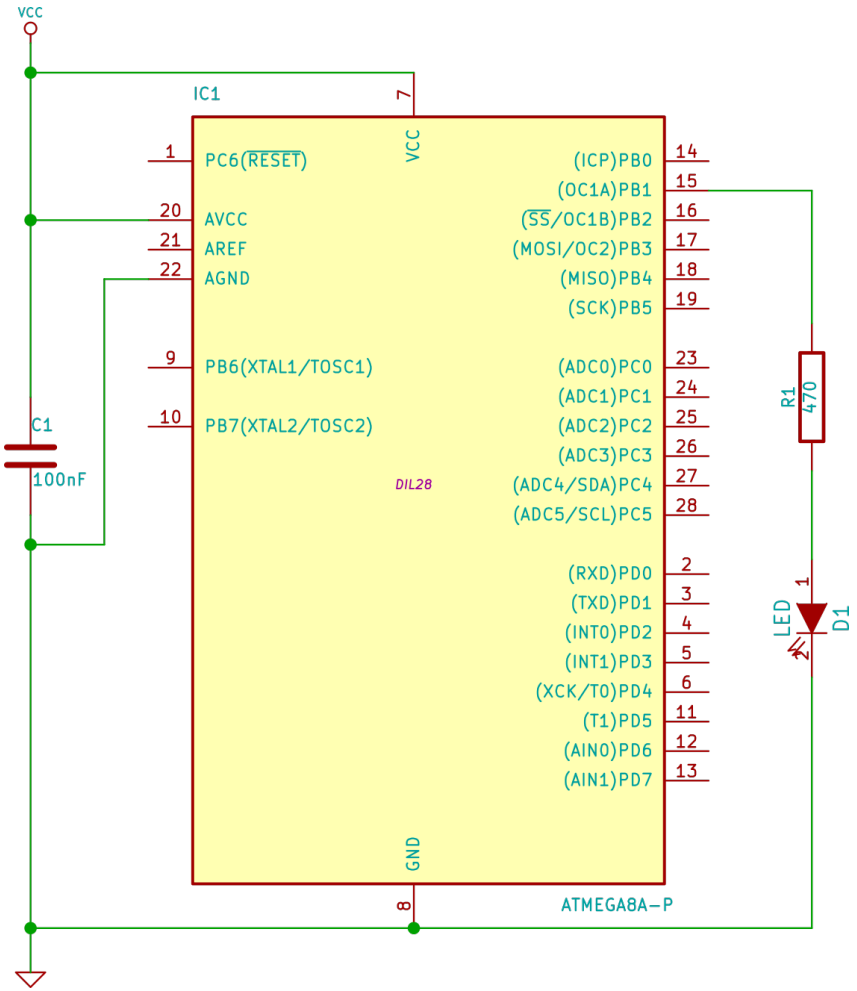
UCSRA – USART Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

```
if(UCSRA & (1 << TXC)) {
    // TXC-Bit ist gesetzt
} else {
    // TXC-Bit ist nicht gesetzt
}
```

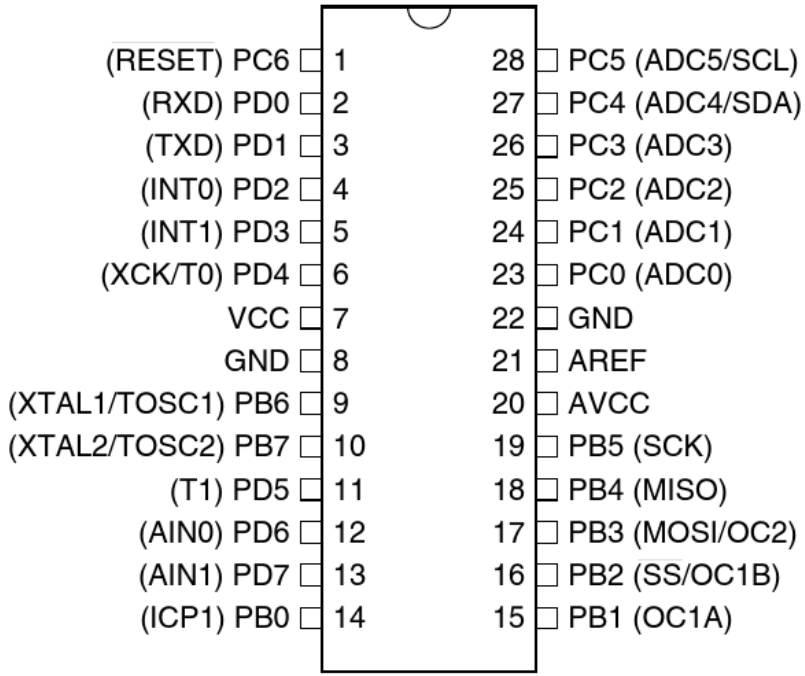
„Hello World“

- Textausgabe ist schon eher schwierig
- einfaches Lebenszeichen: blinkende LED
- einfache Schaltung



„Hello World“ – Programm

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

int main(void) {
    // Pin auf Output stellen
    DDRB |= (1 << PB1);

    while(1) {
        // Pin B1 auf High
        PORTB |= (1 << PB1);
        _delay_ms(1000);
        // Pin B1 auf Low
        PORTB &= ~(1 << PB1);
        _delay_ms(1000);
    }
    return 0;
}
```



DDRx Input/Output

PORTx High/Low

PINx Status

Kompilieren

```
avr-gcc -mmcu=atmega8a -DF_CPU=1000000 -Os \
                       -o helloworld.elf helloworld.c
avr-strip helloworld.elf
```

- Chip-Angabe
- Taktfrequenz (für `_delay_ms`)
- möglichst kleines Programm (Optimierung)

Flashen

Wie bekommen wir das Programm jetzt auf den Chip?

- dedizierte Hardware zum Beschreiben des Flash-Speichers
- In-System-Programmer (ISP)
- viele verschiedene Schnittstellen, häufig
  - USB
  - Serielle Schnittstelle
  - Parallelport
- bei uns: USBASP  [6]

Flashen

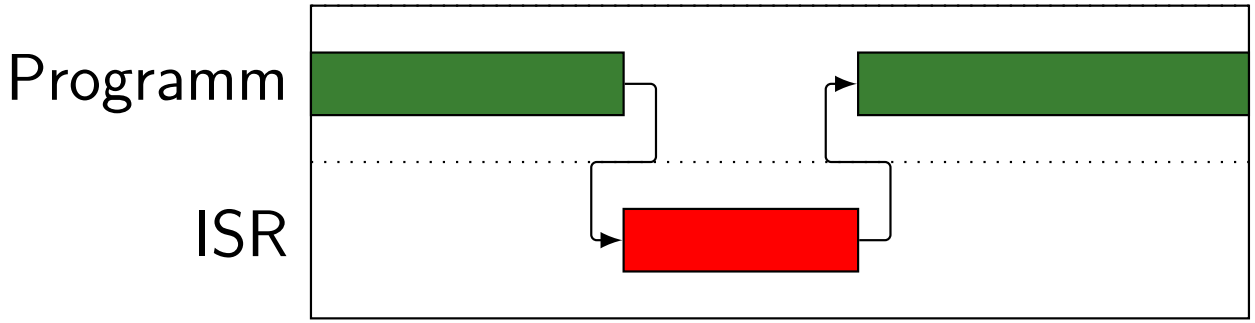
Wie bekommen wir das Programm jetzt auf den Chip?

```
avr-objcopy -O ihex -R .eeprom helloworld.elf helloworld.hex
avrdude -p m8 -c usbasp -U flash:w:helloworld.hex:i \
                        -U lfuse:w:0xe1:m -U hfuse:w:0xd9:m
```

- Intel Hex-Format
  - Auswahl der Sections
- Fuse-Bits
  - Taktquelle
  - ggf. Taktfrequenz
  - Überschreibungsmöglichkeiten einschränken
- Tool-Unterstützung, z. B. `fusecalc`

Interrupts

- Hardware-Unterbrechung
- danach: Rückkehr ins eigentliche Programm



- Behandlungsroutinen selbst definierbar
  - ISR (Interrupt-Service-Routine)
  - zeitkritische Ausführung

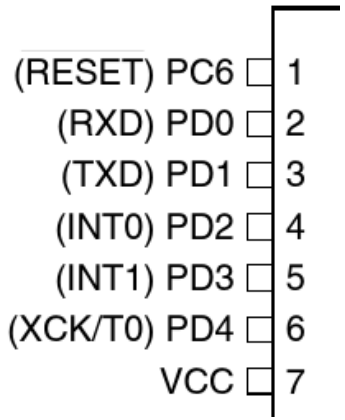
Interrupts – Quellen

- fest definierte Quellen
  - Kommunikation
  - Timer
  - Ereignisse
- Erlauben/Blockieren mit `sei`/`cli`
- Maskierung

Vector No.	Program Address(2)	Source	Interrupt Definition
1	0x000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	TIMER2 COMP	Timer/Counter2 Compare Match
5	0x004	TIMER2 OVF	Timer/Counter2 Overflow
6	0x005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	0x006	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	0x007	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	0x008	TIMER1 OVF	Timer/Counter1 Overflow
10	0x009	TIMER0 OVF	Timer/Counter0 Overflow
11	0x00A	SPI, STC	Serial Transfer Complete
12	0x00B	USART, RXC	USART, Rx Complete
13	0x00C	USART, UDRE	USART Data Register Empty
14	0x00D	USART, TXC	USART, Tx Complete
15	0x00E	ADC	ADC Conversion Complete
16	0x00F	EE_RDY	EEPROM Ready
17	0x010	ANA_COMP	Analog Comparator
18	0x011	TWI	Two-wire Serial Interface
19	0x012	SPM_RDY	Store Program Memory Ready

Interrupts – Programmierung

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

volatile uint8_t counter = 0;

ISR(INT0_vect) {
    ++counter;
}

int main(void) {
    DDRD &= ~(1 << PD2);
    PORTD |= (1 << PD2);
    GICR |= (1 << INT0);
    sei();
    while(1) {
        if(counter >= 5) {
            /* ... */
        }
    }
}
```



- Interrupt wenn PD2 auf Low
  - Pull-Up per PORTD
- wichtig: `volatile`
  - Variable kann sich „im Hintergrund“ ändern
  - Einfluss auf Optimierung

Interrupts – Synchronisierung

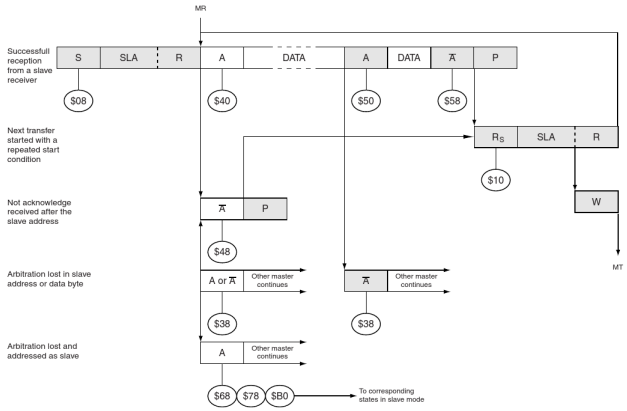
```
#include <avr/io.h>
#include <avr/interrupt.h>

volatile uint16_t counter = 0;

ISR(INT0_vect) {
    ++counter;
}

int main(void) {
    uint16_t tmp;
    /* ... */
    cli();
    tmp = counter;
    sei();
    /* ... */
}
```

- Anweisungen sind nicht zwangsläufig atomar
  - „8-Bit Microcontroller“
- Ausschalten von Interrupts
  - mit Risiko verbunden
- Interrupts während einer ISR sind abgeschaltet
  - . . . aber einschaltbar

Schnittstellen & Kommunikation

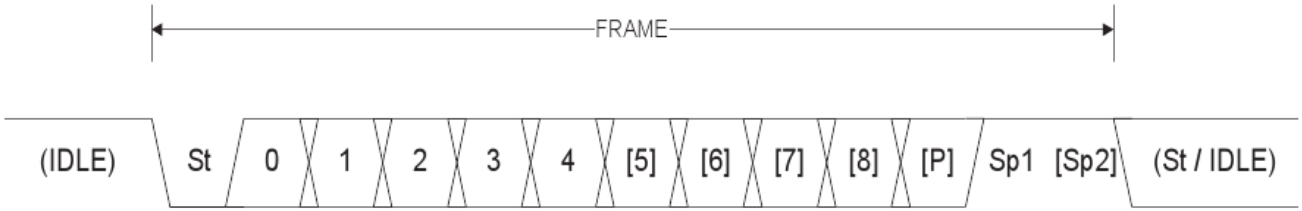
Sehr wichtiges Thema bei Microcontrollern!

- Memory-Mapped I/O
  - Spezialregister für Kommunikation
- bei komplizierteren Bussen: Zustandsautomaten



- wenn nicht „in Hardware“ vorhanden: GPIO-Bit-Banging
  - normalerweise nicht effizient
  - Beispiel: V-USB

Beispiel: UART



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxD or TxD). An IDLE line must be high.

- überall zu finden (z. B.  [5])
- asynchron, Festlegung auf Baudrate
- häufig: 8N1, 9600 Baud

Beispiel: UART – Setup

```
#include <avr/io.h>
#define BAUD 9600
#include <util/setbaud.h>

void uart_init(void) {
    UBRRH = UBRRH_VALUE;
    UBRRL = UBRRL_VALUE;

    if(USE_2X) {
        UCSRA |= _BV(U2X);
    }

    // erlaube Empfangen und Senden
    UCSRB |= _BV(RXEN) | _BV(TXEN);
    UCSRC |= _BV(UCSZ0) | _BV(UCSZ1); // 8N1
}
```

- setbaud.h definiert
  - UBRRH_VALUE
  - UBRRL_VALUE
  - USE_2X
- Einstellungen in den Registern
  - UBRRH, UBRRL (Baudrate)
  - UCSRA, UCSRB, UCSRC (Anzahl Bits etc.)

Beispiel: UART – Senden & Empfangen

```
uint8_t uart_recv(void) {
    while(!(UCSRA & _BV(RXC))) {
        /* wait */
    }
    uint8_t received = UDR;
    return received;
}

void uart_send(uint8_t to_send) {
    while(!(UCSRA & _BV(UDRE))) {
        /* wait */
    }
    UDR = to_send;
}
```

UDR UART Data Register

RXC Receive Complete

UDRE UART Data Register Empty

Schnittstellen & Kommunikation – Ausblick

```
ISR(USART_RXC_vect) {
    uint8_t received = UDR;
    fifo_insert(received);
}

uint8_t uart_recv(void) {
    while(fifo_empty()) {
        /* wait */
    }

    return fifo_dequeue();
}
```

- Kommunikation mit Hilfe von Interrupts
  - Effizienz
  - Asynchronität
  - einfacheres Verfolgen des Zustandsautomaten

Energiesparen – Motivation

Unser ATmega8A ist immer noch recht hungrig!

- Beispiel: Knopfzelle (3 V, 150 mAh)
- Zur Erinnerung: bei 4 MHz Verbrauch von 3.6 mA
- Laufzeit:

$$\frac{150\,\text{mAh}}{3.6\,\text{mA}} = 41.7\,\text{h}$$

Energiesparen – Motivation

Unser ATmega8A ist immer noch recht hungrig!

- Beispiel: Knopfzelle (3 V, 150 mAh)
- Zur Erinnerung: bei 4 MHz Verbrauch von 3.6 mA
- Laufzeit:

$$\frac{150\,\text{mAh}}{3.6\,\text{mA}} = 41.7\,\text{h}$$

Vielleicht muss unser Controller nicht die ganze Zeit laufen...

- periodische Aufgaben
- Handlung auf Knopfdruck

Energiesparen – Motivation

Annahme: Unser Controller muss nur 1 ms pro Sekunde laufen

- Power-down Mode ($\sim 1\,\mu A$) für den Rest der Zeit
- Laufzeit:

$$\frac{150\,\text{mAh}}{3.6\,\text{mA} \cdot \frac{1}{1000} + 1\,\mu\text{A} \cdot \frac{999}{1000}} = 32\,616\,\text{h}$$

$$= 3.7\,\text{Jahre}$$

Energiesparen – Schlafmodi

Die Schlafmodi schalten Features eines Controllers ab, z. B.

- CPU
- Oszillator

Modus	Stromaufnahme	Aufwachzeit
Active	1.2 mA	–
Idle	0.3 mA	6 Takte
ADC Noise Reduction	0.3 mA	6 Takte
Power-down	0.3 µA	lang
Power-save	10 µA	lang
Standby	35 µA	6 Takte

[1]

Energiesparen – Aufwachen

Wie wachen wir wieder auf?

Energiesparen – Aufwachen

Wie wachen wir wieder auf?

- Durch Interrupts

	Active Clock Domains					Oscillators		Wake-up Sources					
Sleep Mode	**clk_{CPU}**	**clk_{FLASH}**	**clk_{IO}**	**clk_{ADC}**	**clk_{ASY}**	**Main Clock Source Enabled**	**Timer Osc. Enabled**	**INT1 INT0**	**TWI Address Match**	**Timer 2**	**SPM/ EEPROM Ready**	**ADC**	**Other I/O**
Idle			X	X	X	X	X(2)	X	X	X	X	X	X
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X	X	X	
Power Down								X(3)	X				
Power Save					X(2)		X(2)	X(3)	X	X(2)			
Standby(1)						X		X(3)	X				

Notes: 1. External Crystal or resonator selected as clock source
2. If AS2 bit in ASSR is set
3. Only level interrupt INT1 and INT0

[3]

Energiesparen – Ausblick

Häufig ist der Controller nicht der energiehungrigste Teil

- hungrige Peripherie, z. B. Sensoren, Funk
- neue Herausforderungen
  - An-/Abschalten
  - Neuinitialisierung vs. Laufenlassen
  - Daten sammeln und gesammelt versenden

Sonstiges

Natürlich können wir hier nicht alles besprechen...
Es fehlt z. B.

- Watchdog-Timer
- Taktquellen
- Timer/Counter
- EEPROM
- Bootloader & Selbst-Programmierung

Die Microcontroller-AG

- Gründung im Sommersemester 2015
- Ergänzung zur Elektro-AG
- Interessenschwerpunkt: digitale Schaltungen und Low-Level-Programmierung
- Finanzielle Unterstützung des FSR

Ihr seid eingeladen, mitzumachen!

- Vorläufiger Termin: Do ab 14 Uhr im Fachschaftsflur

Ausstattung

Eine Grundausstattung ist vorhanden:

- ATmega48PA, ATtiny2313A, ATmega16A
- LEDs, Widerstände, Quarze
- 433 MHz-Funkmodule
- ein Arduino Uno, ein Raspberry Pi
- USB-Programmer, USB-UART-Adapter

Ausstattung

Wir entschieden uns für AVR

- geringer Preis (1–2 € pro Chip)
- freie GCC-Toolchain
- Abstraktion durch avr-libc
- einfache Programmierwerkzeuge
- Arduino

In Zukunft kommt vielleicht MSP430 hinzu.

Voraussetzungen

- Programmierung

Alles weitere glauben wir, euch beibringen zu können!

Projektideen

Wir wollen gern *eure* Projekte verwirklicht sehen, aber wir dachten z. B. an...

- LEDs blinken lassen (easy)
- Videospiel-Controller auslesen (medium easy)
- Experimentieren mit 433 MHz-Funk (medium)
- USB-Geräte, z. B. Tastatur, Gamepad (medium hard)
- ein Betriebssystem schreiben (hard)

- Unterstützung der Kiosk-AG
  - Karten leer?
  - Tür zu lange offen?

Interesse?

Wiki

Terminumfrage

Bitbucket

Quellen I

Mikrocontroller.net: Sleep Mode.

https://www.mikrocontroller.net/articles/Sleep_Mode.

PIC18F8720.

https://commons.wikimedia.org/wiki/File:PIC18F8720.jpg#/media/File:PIC18F8720.jpg.

Atmel.

ATmega8 Datasheet.

http://www.atmel.com/Images/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf, Februar 2013.

Quellen II

Atmel.

ATmega8A Datasheet.

http://www.atmel.com/Images/Atmel-8159-8-bit-AVR-microcontroller-ATmega8A_datasheet.pdf, Februar 2013.

Duncan Lithgow.

Serial port.

https://commons.wikimedia.org/wiki/File:Serial_port.jpg#/media/File:Serial_port.jpg.

Protostack.

USBASP Bild.

http://www.protostack.com/accessories/usbasp-avr-programmer.

Quellen III

Stehfun.

ATMEL-AT90S2333.

https://commons.wikimedia.org/wiki/File:ATMEL-AT90S2333.jpg#/media/File:ATMEL-AT90S2333.jpg, 1 May 2006.

TheBug.

R6511.

https://commons.wikimedia.org/wiki/File:R6511.jpg#/media/File:R6511.jpg.