



Informatikwerkstatt, Foliensatz 1

Einführung bis Bitverarbeitung

G. Kemnitz

Institut für Informatik, Technische Universität Clausthal
27. Oktober 2014



Ziel der Veranstaltung

Die Veranstaltung »Informatikwerkstatt« soll

- ein Gefühl vermitteln, wie die praktische Arbeit eines Informatikers aussieht
- und helfen, später den praktischen Bezug zum Lehrstoff z.B. über Datenstrukturen, Automaten, verteilte Systemen, Rechnernetzen zu finden.



Inhalt des Foliensatzes

Einführung Mikrorechner

- 1.1 Hallo LED
- 1.2 Bitverarbeitung
- 1.3 Weitere Ports
- 1.4 Warteschleifen

Zielstellung

Aufbau eines mikrorechnergesteuerten Fahrzeuges:

- Phase 1: Vermittlung von Grundkenntnissen
 - Kennenlernen einen Mikrorechners
 - seiner Programmierumgebung in C und
 - wie man Hardware (Schalter, Leuchtdioden, Motoren, ...) darüber ansteuert.
- Phase 2: Beispielprogramme
 - Datenaustausch mit dem PC
 - Drehzahlsteuerung von Motoren
 - Einlesen von Sensordaten
 - Datenverarbeitung auf dem PC mit Matlab
- Phase 3: Spezifikation, Entwicklung, Montage und Test der eigenen Lösung. Ergebnispräsentation.



Zu erbringende Leistungen

Teilnahme an 80% der Veranstaltungen (12 Veranstaltungen).

Teilnahmebestätigung gibt es:

- Für Anwesenheit bei der Einführungsveranstaltung und den Praxisvorträgen.
- Bestätigung des Übungsleiters bei den Laborübungen, das aktiv an der Lösung von Aufgaben gearbeitet wurde.
- Die Präsentation einer Lösung auf der Abschlusveranstaltung.

Je nach Vorkenntnis kann bei jedem Aufgabenkomplex zwischen unterschiedlich schweren Aufgaben gewählt werden.

In Abstimmung mit dem Übungsleiter kann auch Hardware ausleihen und zu Hause gearbeitet werden.

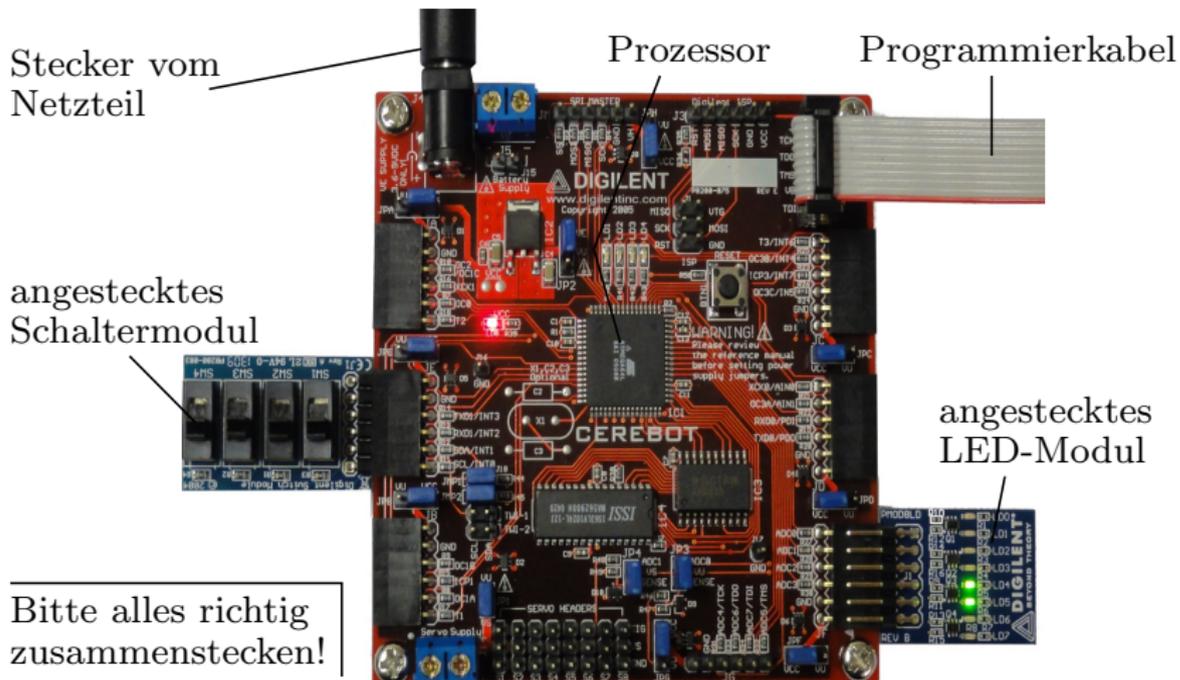


Einführung Mikrorechner



1. Einführung Mikrorechner

Die Mikrorechnerbaugruppe »Cerebot«





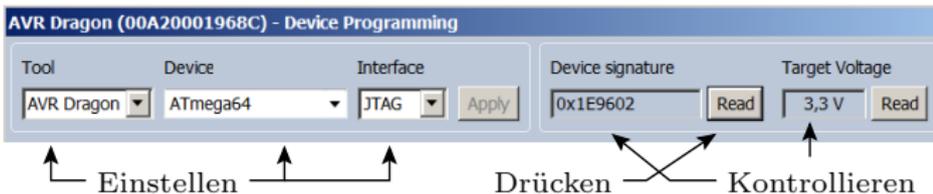
Verbindung mit dem PC

- Rechner unter Windos starten
- Web-Browser öffnen. Foliensatz zum Mitlesen öffnen:

`techwww.in.tu-clausthal.de/site/Lehre/Informatikwerkstatt/`

- Atmel Studio starten 
- Zur Kontrolle, dass der Prozessor richtig angeschlossen und vom System erkannt wird, in Atmel Studio:

Tools > Device Programming





1. Einführung Mikrorechner

- Unter »Device Information« findet man auch das Datenblatt (Datasheet) des Mikrorechners

Interface settings

Tool information

Device information

Memories

Fuses

Lock bits

Production file

Detected Device

Device names	ATmega64, ATmega64A
Device signature	0x1E9602
JTAG id	0x1960203F
Revision	B

Datasheet Information

		ATmega64
CPU	Verarbeitungsbreite	AVR8
Flash size	(65536 Befehle)	64 Kbytes
EEPROM size	(2048 Bytes Daten)	2 Kbytes
SRAM size	(64512 Bytes Daten)	4 Kbytes
VCC range	Versorgungsspannung	2,7 - 5,5 V
Maximum speed		N/A

External links

 [Device Information](#)

 [Datasheets](#)





1. Einführung Mikrorechner

■ Kontrolle der Sicherungsbits (Fuses)

Die Sicherungsbits aktivieren Grundfunktionen, z.B. Programmierschnittstellen, Kopierschutz, ... Bei Einstellungsfehlern lässt sich der Mikrorechner nicht programmieren, die Programme funktionieren nicht, ...

	Fuse Name	Value	
Interface settings	✓ CompMode	<input type="checkbox"/>	
Tool information	✓ WDTON	<input type="checkbox"/>	—Watchdog aus lassen
Device information	✓ OCDEN	<input checked="" type="checkbox"/>	—ON-Chip Debug freigegeben
Memories	✓ JTAGEN	<input checked="" type="checkbox"/>	—JTAG-Programmierung freigeben
Fuses	✓ SPIEN	<input checked="" type="checkbox"/>	
Lock bits	✓ EESAVE	<input type="checkbox"/>	
	✓ BOOTSZ	4096W_7000	
	✓ BOOTRST	<input type="checkbox"/>	
Production file	✓ CKOPT	<input checked="" type="checkbox"/>	
	✓ BODLEVEL	2V7	
	✓ BODEN	<input type="checkbox"/>	
	✓ SUT_CKSEL	INTRCOSC_8MHZ_6CK_64MS	

Die Bedeutung der anderen Bits kann man mit der Suchfunktion in der pdf-Datei herausfinden. Vorerst nicht wichtig.



Hallo LED



Das erste Programm

Port D (Schalter) Eingang
Port F (LEDs) Ausgang
Wiederhole immer
lese Port E in einen Variable
schreibe Wert auf Port F

```
#include <avr/io.h> 1
uint8_t tmp; // 8-Bit-Variable
int main(){
    DDRD = 0; // DDR... Bit(i)=0: Eingabebit
    DDRF = 0xff; // DDR... Bit(i)=1: Ausgabebit
    while(1) { // wiederhole immer
        tmp = PIND; // Lesen von Port D
        // hier kann der Wert von tmp bearbeitet werden
        PORTF = tmp; //
    }
}
```

¹Definiert Registeradressen und Datentypen wie uint8_t

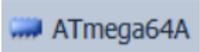


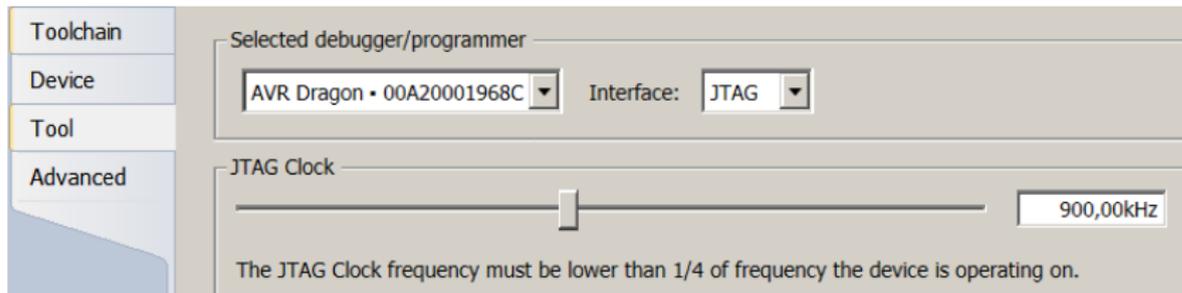
Arbeitsschritte

- Projekt einrichten: Programmiersprache, Art des Projekts, Arbeitsverzeichnis, Prozessortyp, Programmiergerät und Optionen für die Programmerstellung festlegen.
- Programm eingeben, übersetzen und Übersetzungsfehler beseitigen.
- Programm im Debugger starten. Schrittweise Abarbeitung und Kontrolle der Zwischenergebnisse.
- Programm normal starten und testen der Funktion in Echtzeit.



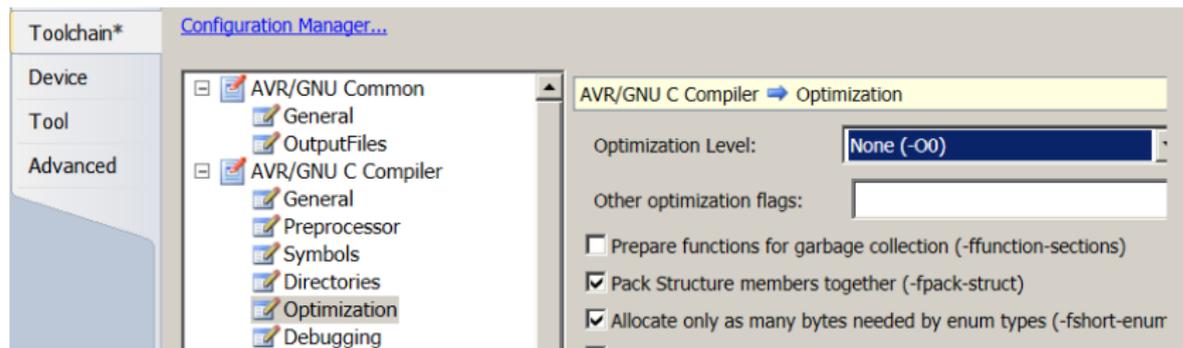
Projekt einrichten

- Neues Projekt anlegen:
File > New > Project > GCC C Executable
Project Name: HalloLED. Location: H:\Werkstatt.
Prozessortyp: Atmega64.
- Doppelklick auf  ATmega64A.
- Unter Tool wie im nachfolgenden Bild Dragon und JTAG auswählen:





- Unter Toolchain die Optimierung für den Übersetzer ausschalten² (O1 durch O0 ersetzen)



- Zeilennummern einschalten:

Tools > Options > Text Editor > All languages
> General: Display on line numbers ✓

²Optimiert sonst für das Ein-/Ausgabeverhalten entbehrliche Schritte weg, z.B. das Zwischenspeichern von Variablenwerten im Datenspeicher und Warteschleifen. Optimierte Programme lassen sich nur eingeschränkt im Schrittbetrieb auf Quellcodeniveau testen.



Programm eingeben

- Automatisch erzeugten Programmrahmen vervollständigen und speichern:

```
 9  #include <avr/io.h>
10
11  uint8_t tmp;
12  int main(void)
13  {
14      DDRD = 0b00000000; // alle Bits 0 (Eingänge)
15      DDRF = 0b11111111; // alle Bits 1 (Ausgänge)
16      while(1)
17      {
18          //TODO:: Please write your application code
19          tmp = PIND;
20          PORTF = tmp;
21      }
22 }
```

Im Schrittbetrieb abarbeiten

- Debugger starten:

Debug > Start Debugging and Break (Alt+F5)

```
10
11 uint8_t tmp;
12 int main(void)
13 {
14     DDRD = 0b00000000; // alle Bits 0 (Eingänge)
15     DDRF = 0b11111111; // alle Bits 1 (Ausgänge)
16     while(1) {
17         tmp = PIND;
18         PORTF = tmp;
19     }
20 }
```

– nächste Anweisung; – Unterbrechungspunkt; – Schritt abarbeiten und halten; – Fortsetzen

Vor der Ausführung Beobachtungsfenster öffnen

- Fenster zur Beobachtung der Ein- und Ausgabewerte:

Debug > Windows > IO View

Name	Address	Value	Bits
I/O PORTC			
I/O PORTD			
I/O PORTE			
I/O PORTF			
I/O PORTG			
I/O PIND	0x30	0x07	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/>
I/O DDRD	0x31	0x00	<input type="checkbox"/>
I/O PORTD	0x32	0x00	<input type="checkbox"/>

- Fenster zur Beobachtung der (globalen) Variablen:

Debug > Windows > Watch > Watch1

Name	Value	Type
tmp	0x00	uint8_t{data}@0x0100

Programm in Echtzeit abarbeiten

- Debug > Stop Debugging (Ctrl+Shift+F5) .
- Debug > Start without Debugging .
- Änderung der Schalterwerte und Kontrolle der LED-Ausgabe.



Bitverarbeitung



Verarbeitung von Bits

- Mikrorechner verarbeiten vielfach einzelne Bits:
Schalteneingaben, LED-Ausgaben, Motor ein/aus, ...
- Die Bits werden in unserem Prozessor immer zu Bytes (8 Bit) zusammengefasst.

Verarbeitungsmöglichkeiten für Bitvektoren:

```
uint8_t a, b; // zwei 8-Bit-Variablen
```

- kopieren

```
a = b;
```

- nach rechts oder links verschieben

```
a = 0b10110111;
```

```
b = a >> 2; // b: 0b00101101 = 0x2D
```

```
a = b << 3; // a: 0b10111000 = 0xB8
```

(0b... – Binärdarstellung; 0x...–Hexadezimaldarstellung).



■ bitweise Negation

 $a = 0b10110111;$ $a = \sim a; // a: 0b01001000$

■ bitweises UND (Ergebnis 1, wenn beide Operanden 1 sind)

 $a = 0b10011111 \& 0b00111101; // a: 0b00011101$

■ bitweises ODER (Ergebnis 1, wenn mindestens ein Operand 1 ist)

 $a = 0b10011111 | 0b00111101; // a: 0b10111111$

■ bitweises EXOR (Ergebnis 1, wenn genau ein Operand 1 ist)

 $a = 0b10011111 \wedge 0b00111101; // a: 0b10100010$

x_1	x_0	\bar{x}_0	$x_1 \wedge x_0$	$x_1 \vee x_0$	$x_1 \oplus x_0$
0	0	1	0	0	0
0	1	0	0	1	1
1	0		0	1	1
1	1		1	1	0

Programmieraufgabe: $LE4 = SW1 \text{ UND } SW2$

Schalter / LED	SW1	SW2	LD4
Port, Bitnummern	D0	D1	F0

Möglicher Algorithmus:

```
uint8_t a, b;
...
while(1) {
    a = PIND; // Lesen aller Schalterwerte
    // nach b kopieren und alle Bits außer SW1 löschen
    b = a & 0b00000001;
    // in a alle Bits außer SW2 löschen
    // und Ergebnis in Bit 0 verschieben
    a = (a & 0b00000010) >>1;
    // bitweise UND verknüpfen und ausgeben
    PORTF = a & b;
}
```



Ein Programm (geringfügig modifizierter Algorithmus):

```
 9  #include <avr/io.h>
10
11  uint8_t a, b;
12  int main(void)
13  {
14      DDRD = 0b00000000; // alle Bits 0 (Eingänge)
15      DDRF = 0b11111111; // alle Bits 1 (Ausgänge)
16      while(1) {
17          a = PIND & 0b00000001;
18          b = PIND & 0b00000010;
19          b = b>>1;
20          PORTF = a & b;
21      }
22 }
```

■ Wie funktioniert dieses Programm?



Aufgabe 1.1: Beispielprogramm ausprobieren

- Anlegen eines neuen Projektes mit dem Namen Logiktest und ansonsten denselben Einstellungen wie »HalloLED«.
- Eingabe des Beispielprogramms der Folie zuvor.
- Test im Schrittbetrieb.
- Test in Echtzeit.
- Veränderung des Programms. Andere Verarbeitung oder etwas andere Funktion.
- Ausprobieren, was die Veränderungen bewirken.



Aufgabe 1.2: Berechnung der Parität

Schreiben Sie ein Programm, das auf LED(4) die Parität der vier Schalterwerte ausgibt und testen Sie es.

Die Parität ist die bitweise Summe der Eingabevariablen (Schalter) unter Vernachlässigung des Übertrags:

$$\text{LED}(4) = \text{SW}(3) \text{ EXOR } \text{SW}(2) \text{ EXOR } \text{SW}(1) \text{ EXOR } \text{SW}(0)$$

SW(3)	SW(2)	SW(1)	SW(0)	LED(4)
0 (aus)				
0 (aus)	0 (aus)	0 (aus)	1 (ein)	1 (ein)
0 (aus)	0 (aus)	1 (ein)	0 (aus)	1 (ein)
0 (aus)	0 (aus)	1 (ein)	1 (ein)	0 (aus)
...



Aufgabe 1.3: Komplexe Logikaufgabe

- LD4 soll leuchten, wenn mindestens einer der vier Schalter eingeschaltet ist.
- LD5 soll leuchten, wenn mindestens zwei der vier Schalter eingeschaltet sind.
- LD6 soll leuchten, wenn mindestens drei der vier Schalter eingeschaltet sind.
- LD7 soll leuchten, wenn alle vier Schalter eingeschaltet sind.



Weitere Ports



Nutzung weiterer Ports

- Der Mikrorechner hat sieben 8-Bit-Ports (A bis G).
- Die Baugruppe hat sechs identische Stecker (JA bis JG und JM), auf denen jeweils vier Port-Bits herausgeführt sind. Zuordnung siehe CEREBOT-rm.pdf, Table 1³.
- Wenn man mit einem Adapterkabel den Stecker JF mit der oberen statt wie bisher mit der unteren Reihe verbindet, leuchten LD0 bis LD3 statt LD4 bis LD7.
- Man kann auch mit zwei Kabeln die Bits von zwei Ports auf die acht LEDs des Moduls ausgeben.

³Die pdf-Datei ist auf techwww oder mit google auffindbar.



Aufgabe 1.4: Bestimmung Stecker-Port-Zuordnung

Vervollständigen Sie unter Nutzung der Dokumentation folgende Zuordnungstabelle zwischen Steckern und Ports:

Stecker	Pin 1	Pin2	Pin3	Pin 4
JA				
JB				
JC				
JD				
JE	PD 0	PD 1	PD 2	PD 3
JF	PF 0	PF 1	PF 2	PF 3
JG				
JM				

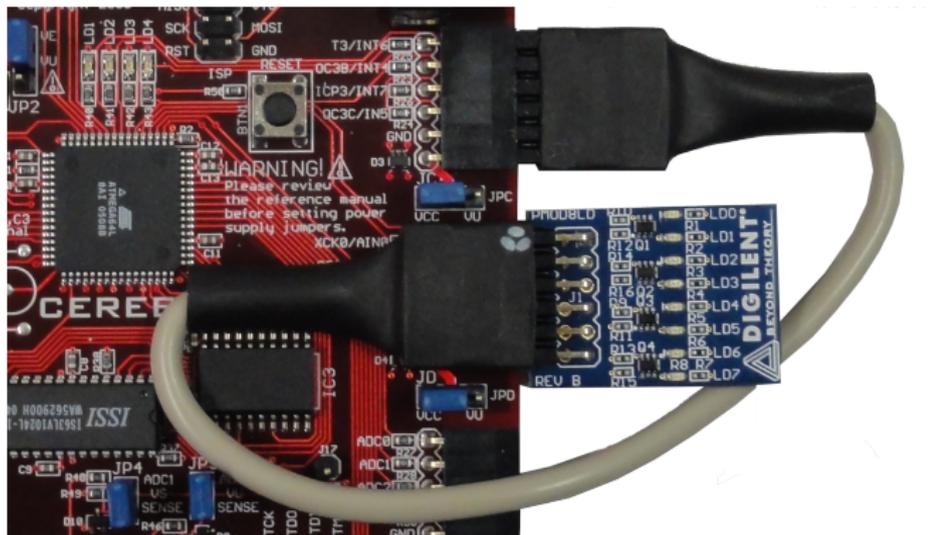


Aufgabe 1.5: LED-Ausgabe auf der Cerebot-Baugruppe

- Schaltermoduls am Stecker JE lassen.
- Die Schalterwerte sollen auf den vier LEDs direkt über dem Mikrorechner ausgegeben werden.
- Umstecken des Schaltermoduls an JA und Änderung des Programms, so dass auch jetzt noch der Eingabewert von Schalter i auf Leuchtdiode LD_i der Cerebot-Baugruppe ausgegeben wird.

Aufgabe 1.6: LED-Ausgabe über JC und JD

- Anstecken von JC an die obere und JD an die untere Anschlussreihe des LED-Moduls.
- Auf LD0 bis LD3 soll SW1 bis SW4 direkt und auf LD4 bis LD7 soll SW1 bis SW4 negiert ausgegeben werden.





Warteschleifen



Warteschleifen

- Die Leuchtdioden sollen sichtbar blinken. Dazu müssen zwischen Ein- und Ausschalten mindestens 100 ms vergehen.
- Der Prozessor arbeitet in 100 ns etwa eine halbe Millionen Maschinenbefehle bzw. Tausende C-Anweisungen ab. Lösung Warteschleife:

```
uint8_t a=0; // Ausgabewert
uint16_t b; // Zähler WB: 0 bis  $2^{16} - 1 \approx 65.000$ 
...
while(1){
    PORTF = a;
    for (b = 0; b<65000; b++); // Warteschleife
    a++;
}
```

- Was tut der Programmausschnitt?
- Warum ist er für einen Test im Schrittbetrieb ungeeignet?



Aufgabe 1.7: Wandernder Leuchtpunkt

Entwickeln Sie ein Programm, bei dem eine Leuchtpunkt auf den 4 LEDs auf der Cerebot-Baugruppe hin und her wandert.



Aufgabe 1.8: Wandernder Leuchtpunkt für Fortgeschrittene

- LED-Modul so anstecken und Programm so ändern, dass der Leuchtpunkt über alle 8 LEDs des LED-Moduls hin und her wandert.
- Einbeziehung des Schaltermoduls zur Steuerung des Lichteffektes, z.B. so, dass bei eingeschaltetem Schalter SW(0) die LED-Ausgabe invertiert und bei eingeschaltetem Schalter SW(1) die Wartezeit zwischen den Ausgaben halbiert wird.