

Atmel AVR RISC μ C unter GNU/Linux programmieren

Jan Grosser
email@jan-grosser.de

TroLUG, Troisdorf, 05.12.2013

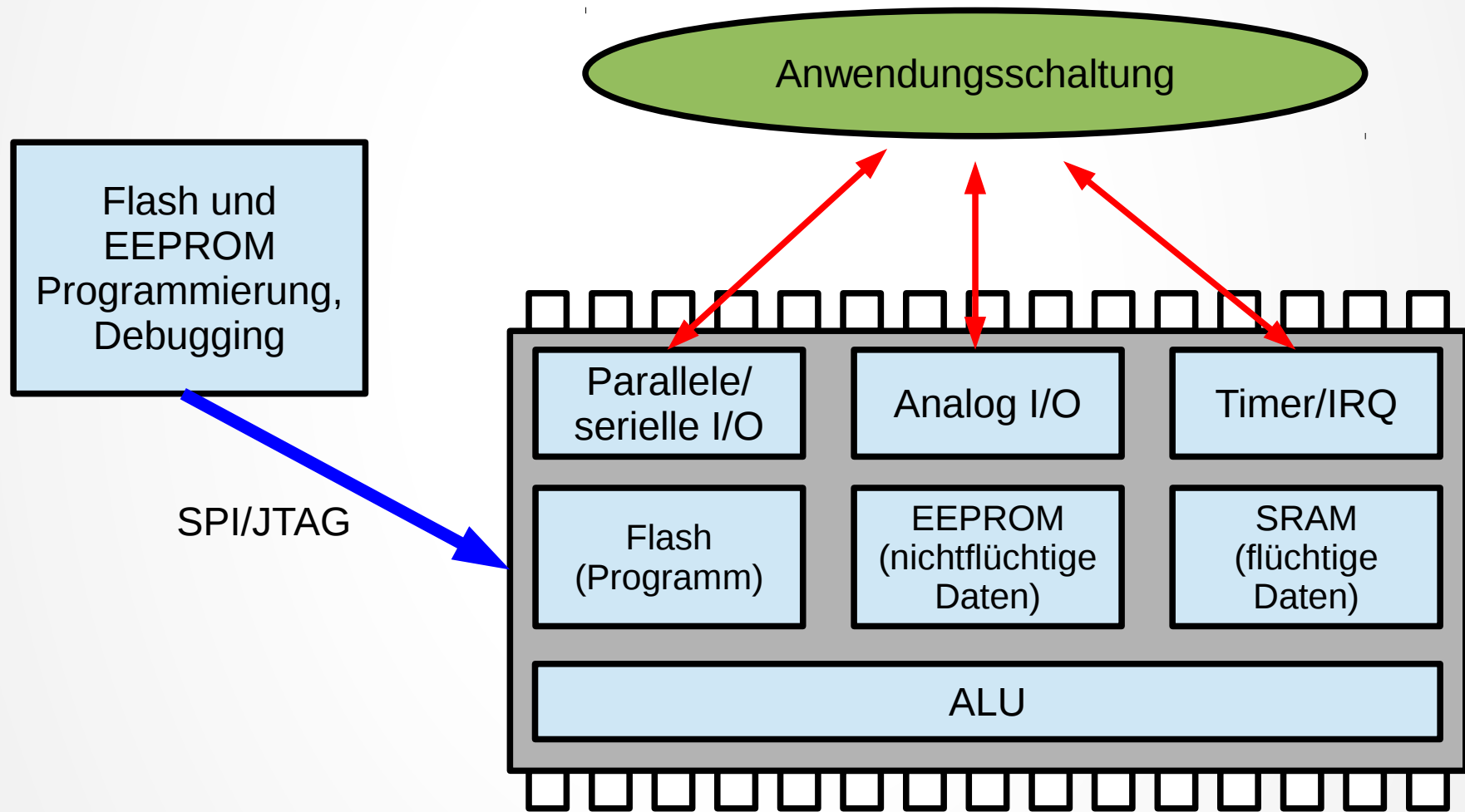
Inhalt

- Begriffe
 - Mikrocontroller
 - Assembler
- Was wird benötigt?
 - Hardware
 - Software
- Beispiel-Workflow (simple-io.asm)
- Probleme/Lösungen
- Lesenswert

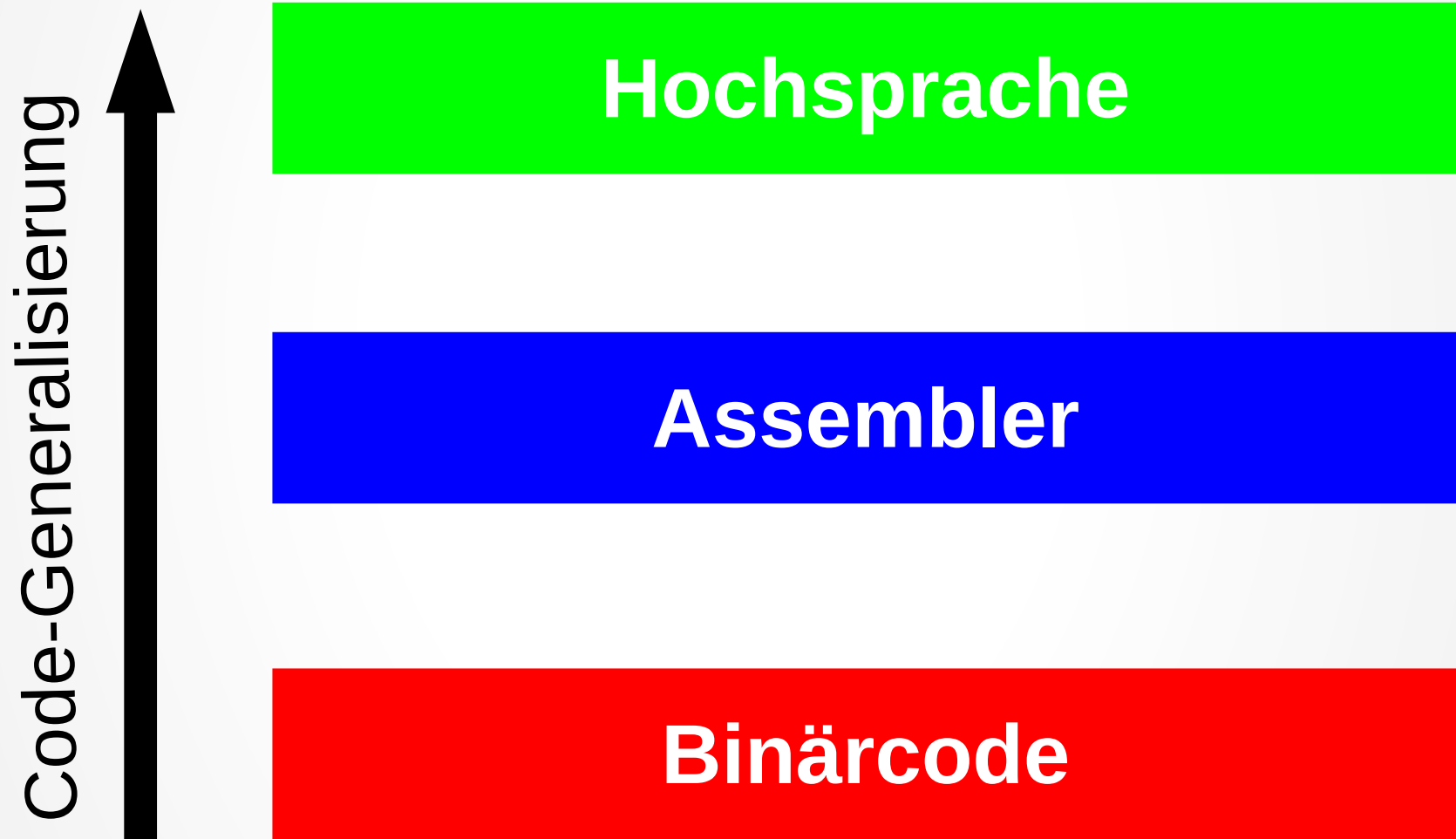
Mikrocontroller

- μ C sind leistungsfähige Mini-Computer
- System-on-Chip:
Prozessor + Speicher + Peripherie
- Armbanduhren, Waschmaschinen, Chipkarten, Autos, ...
- Assembler, C, BASIC, Pascal, Forth, ...
- Ab < 1€ (z.B. ATtiny2313A-PU)
- Bekannte Familien:
 - Atmel AVR/AVR32 (8, 32 Bit)
 - PIC Mikrocontroller (8, 16, 32 Bit)
 - Texas Instruments MSP430 (16 Bit)
 - u.v.m.

Mikrocontroller



Assembler



Assembler

- Menschenlesbare Darstellung von Maschinensprache (Mnemos statt Opcode)
- Gegenteil: Generalisierte Hochsprache wie bspw. C
- Vorteile
 - Direkte Programmierung des Zielsystems
 - Nutzbarkeit aller Features
 - **Effizienter** als Hochsprachen
 - Verständnis der Funktionsweise des Zielsystems
- Nachteile
 - RISC μ C haben nur primitive Funktionen
 - Komplexe Funktionen nur aufwändig realisierbar
 - Schlechte Portierbarkeit

Was wird benötigt?

- Hardware
 - Evaluations-Board
 - Programmer
 - Steckbrett, Elektronik-Kleinkram
- Software
 - Compiler/Assembler
 - Programmier-Software für μC

Hardware: Evaluationsboard

- **ATMEL Evaluations-Board** für < 30€ bei www.pollin.de
- PoC/Entwicklung
- Grundl. Beschaltung (z.B. Power-Versorgung)
- Peripherie (Schalter, LEDs, RS232, ...)
- Sockel für μC oder festverlöteter μC
- Eigene Schaltung läßt sich verbinden
- Arduino Boards gehen z.B. auch!

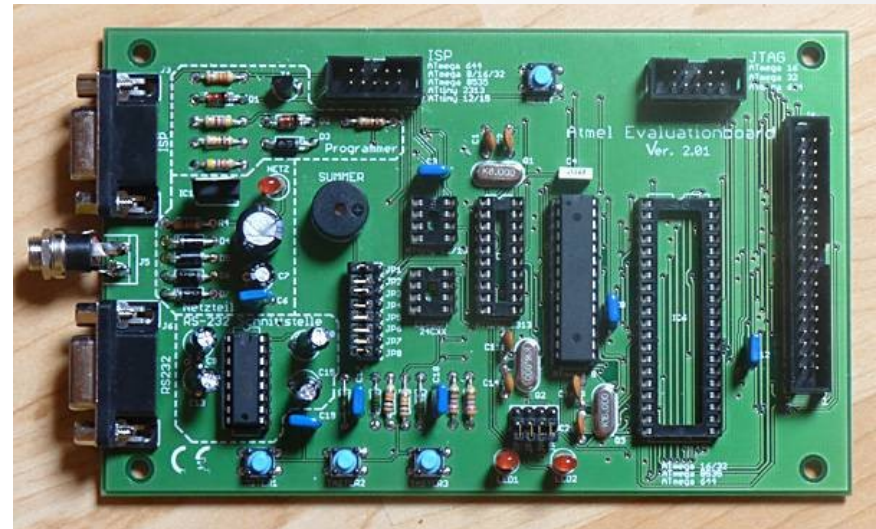
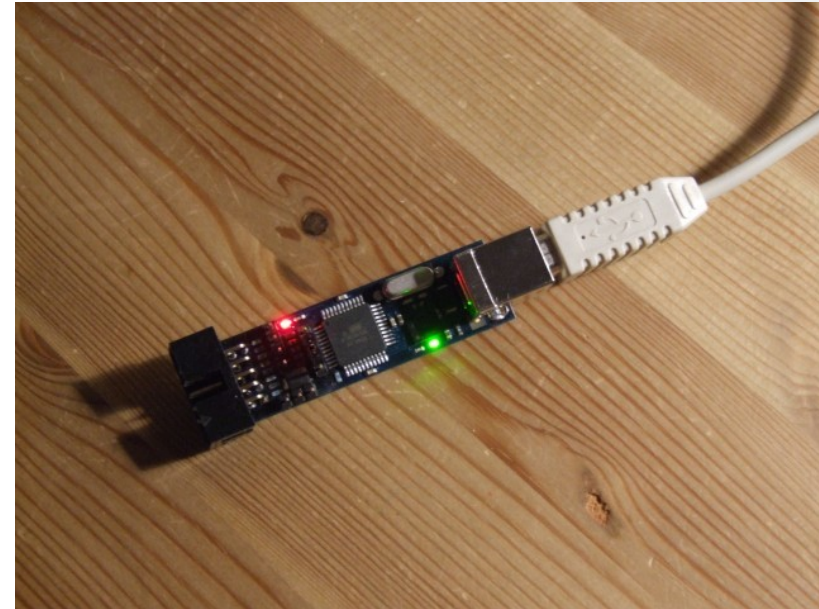


Bild: L. Höll (www.mydarc.de)

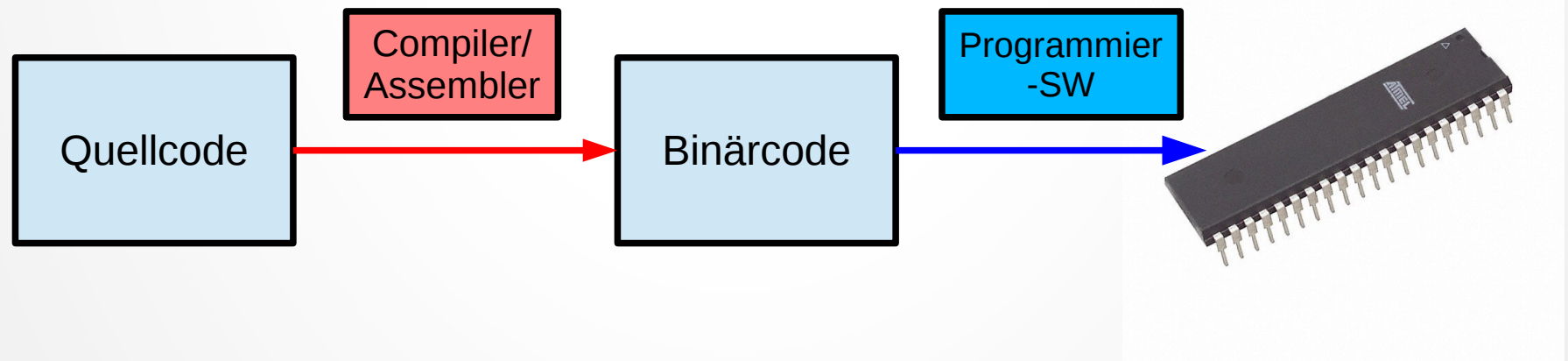
Hardware: Programmer

- Embedded Projects **USBprog** für < 30€
- Kompatibel AVRISP mk2
- Verbindung PC <--> μ C
- Schreiben und Lesen des Flash und EEPROM
- Frei programmierbar, d.h. vielfältig einsetzbar
 - In-System-Programmer/Serial Programming IF
 - JTAG Programmer/Debugger
 - UART-USB-Wandler
 - ...



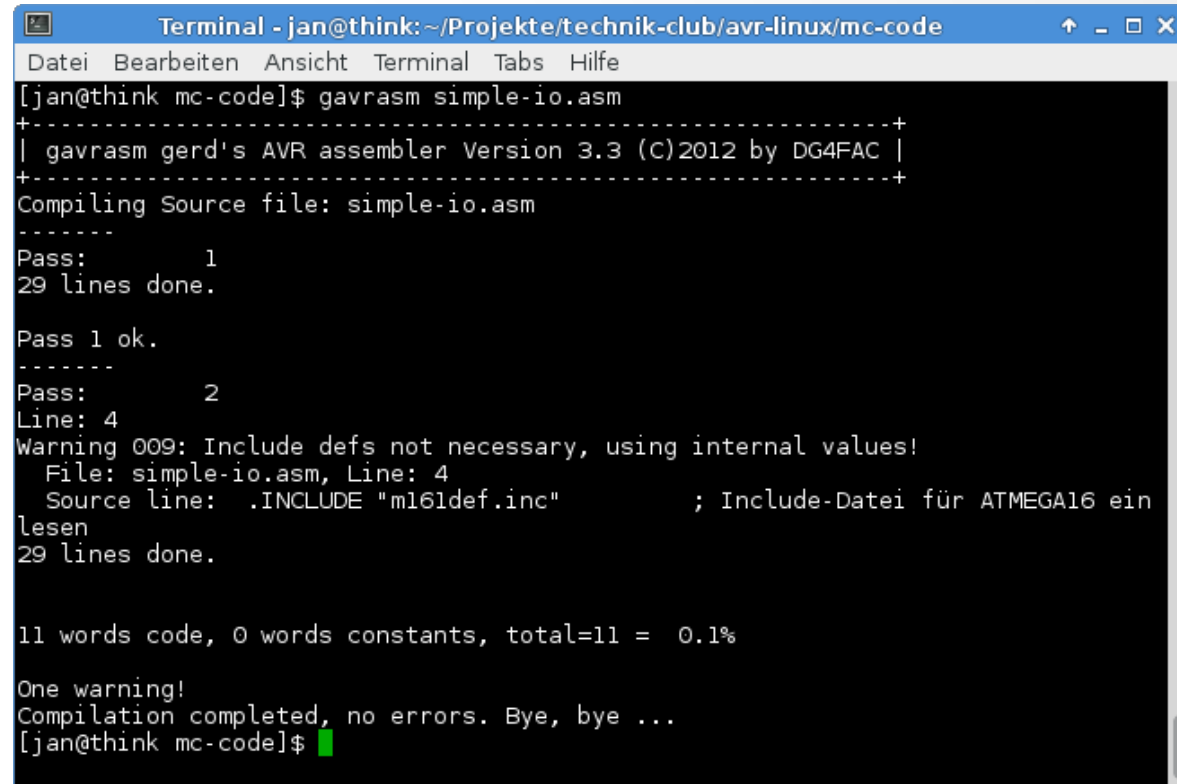
Software: Compiler

- Atmel AVR RISC μ C in mehreren Sprachen programmierbar:
 - Assembler
 - C
 - BASIC
 - ...



Software: Compiler

- **gavrasm**
„Gerds AVR
Assembler“
- Open Source
- Unterstützt viele
ATMEL AVR μ C
- Erzeugt
Maschinencode
- Kommandozeilen-Programm



```
Terminal - jan@think: ~/Projekte/technik-club/avr-linux/mc-code
Datei Bearbeiten Ansicht Terminal Tabs Hilfe
[jan@think mc-code]$ gavrasm simple-io.asm
+-----+
| gavrasm gerd's AVR assembler Version 3.3 (C)2012 by DG4FAC |
+-----+
Compiling Source file: simple-io.asm
-----
Pass:      1
29 lines done.

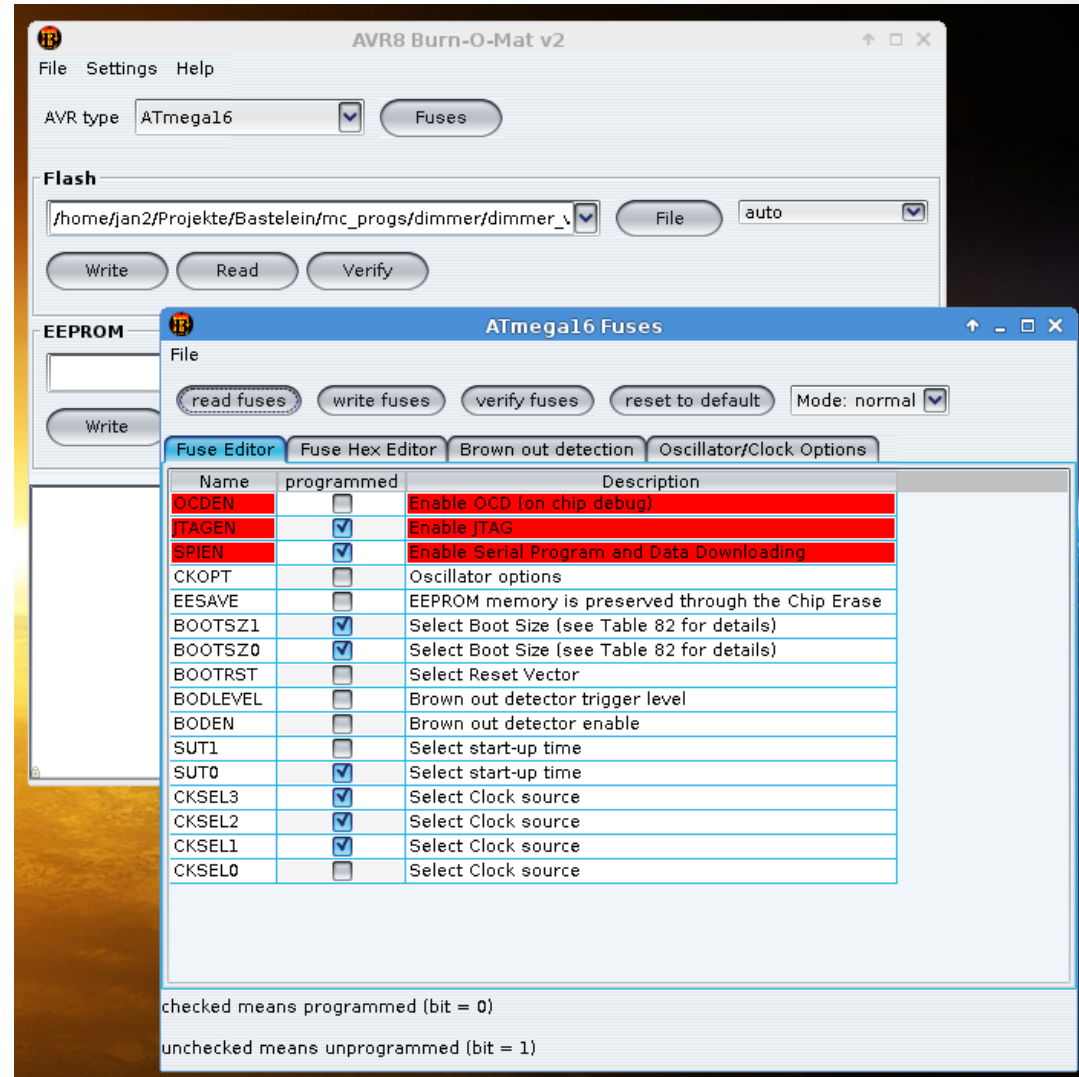
Pass 1 ok.
-----
Pass:      2
Line: 4
Warning 009: Include defs not necessary, using internal values!
  File: simple-io.asm, Line: 4
    Source line:  .INCLUDE "m161def.inc"           ; Include-Datei für ATMEGA16 ein
lesen
29 lines done.

11 words code, 0 words constants, total=11 =  0.1%

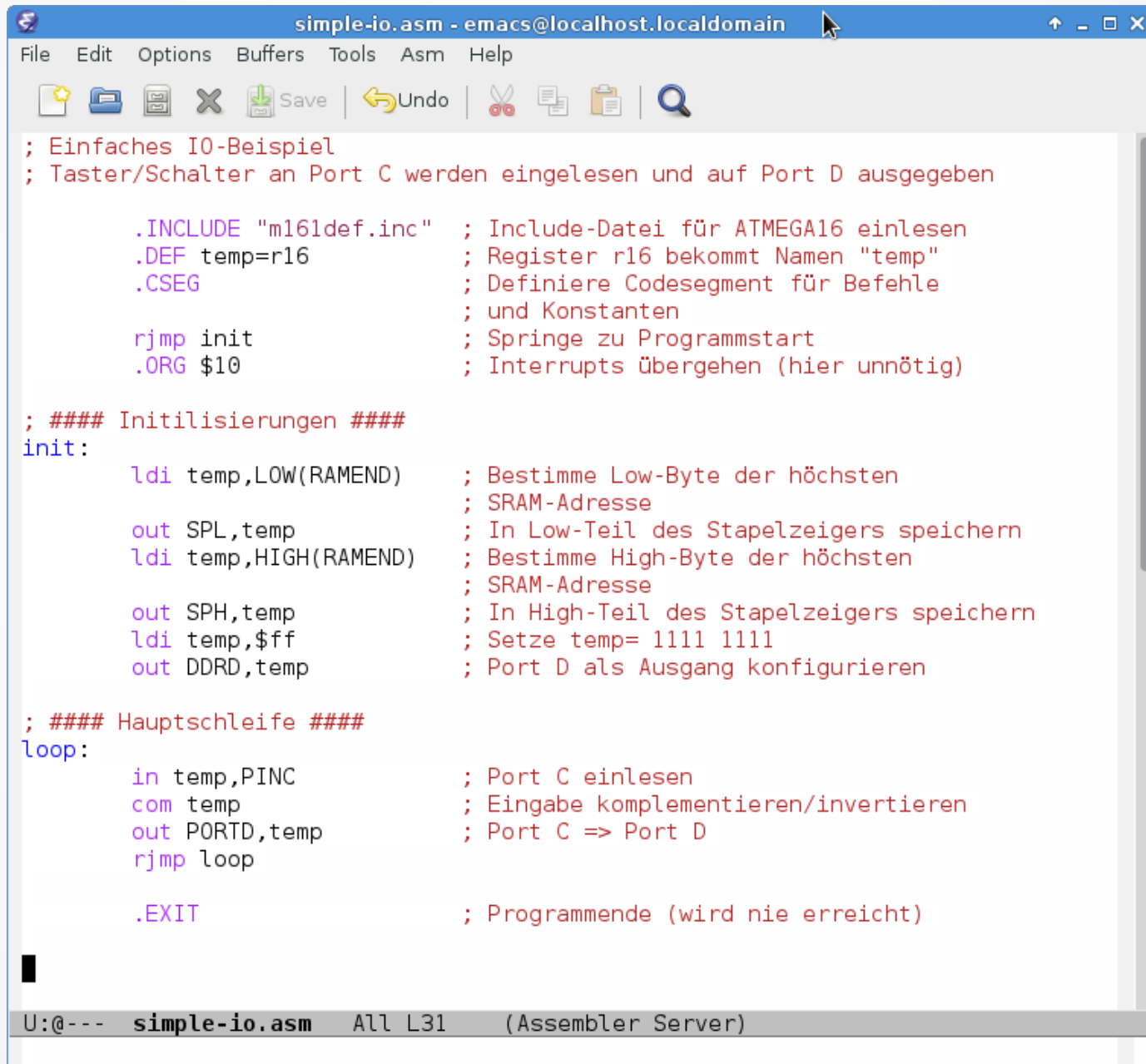
One warning!
Compilation completed, no errors. Bye, bye ...
[jan@think mc-code]$
```

Software: Programmier-SW

- **Avrdude**: Open Source Programmer
- **AVR8 Burn-O-Mat** GUI für avrdude
- Hilft insbes. beim Setzen der Fuses



Beispiel-Workflow: Quellcode



```
simple-io.asm - emacs@localhost.localdomain
File Edit Options Buffers Tools Asm Help

; Einfaches I/O-Beispiel
; Taster/Schalter an Port C werden eingelesen und auf Port D ausgegeben

    .INCLUDE "m16ldef.inc" ; Include-Datei für ATMEGA16 einlesen
    .DEF temp=r16          ; Register r16 bekommt Namen "temp"
    .CSEG                 ; Definiere Codesegment für Befehle
                           ; und Konstanten
    rjmp init              ; Springe zu Programmstart
    .ORG $10               ; Interrupts übergehen (hier unnötig)

; ##### Initilisierungen #####
init:
    ldi temp,LOW(RAMEND)   ; Bestimme Low-Byte der höchsten
                           ; SRAM-Adresse
    out SPL,temp           ; In Low-Teil des Stapelzeigers speichern
    ldi temp,HIGH(RAMEND) ; Bestimme High-Byte der höchsten
                           ; SRAM-Adresse
    out SPH,temp           ; In High-Teil des Stapelzeigers speichern
    ldi temp,$ff           ; Setze temp= 1111 1111
    out DDRD,temp          ; Port D als Ausgang konfigurieren

; ##### Hauptschleife #####
loop:
    in temp,PINC           ; Port C einlesen
    com temp               ; Eingabe komplementieren/invertieren
    out PORTD,temp         ; Port C => Port D
    rjmp loop

    .EXIT                  ; Programmende (wird nie erreicht)

U:@--- simple-io.asm All L31 (Assembler Server)
```

Beispiel-Workflow: Assemblieren

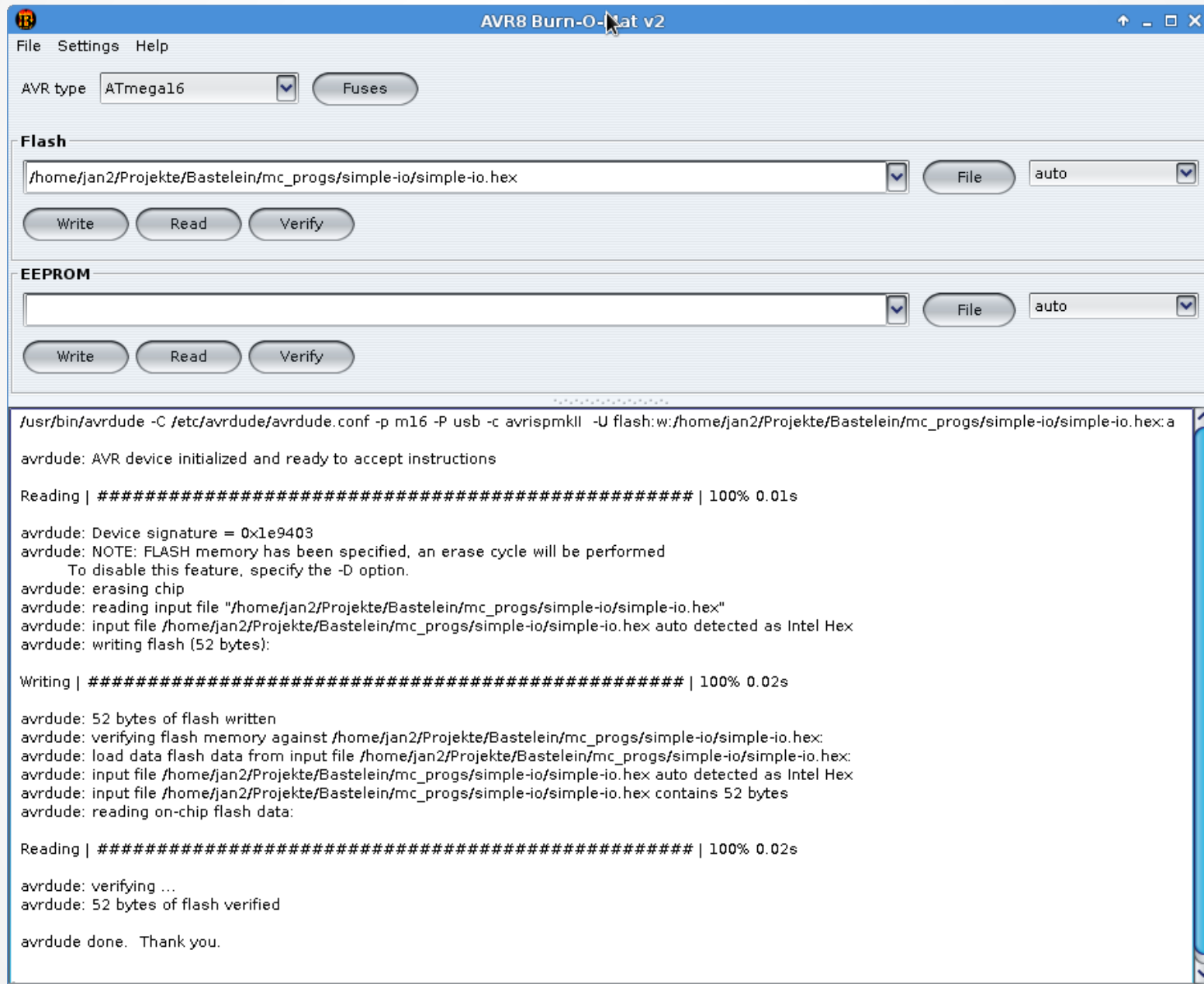
```
Terminal - fish /home/jan/Projekte/Bastelein/mc_progs/simple-io
Datei Bearbeiten Ansicht Terminal Tabs Hilfe
jan@think ~/P/B/m/simple-io> gavrasm simple-io.asm
+-----+
| gavrasm gerd's AVR assembler Version 3.3 (C)2012 by DG4FAC |
+-----+
Compiling Source file: simple-io.asm
-----
Pass:          1
29 lines done.

Pass 1 ok.
-----
Pass:          2
Line: 4
Warning 009: Include defs not necessary, using internal values!
  File: simple-io.asm, Line: 4
  Source line:  .INCLUDE "m161def.inc"          ; Include-Datei für ATMEGA16 ein
lesen
29 lines done.

11 words code, 0 words constants, total=11 =  0.1%

One warning!
Compilation completed, no errors. Bye, bye ...
jan@think ~/P/B/m/simple-io> ls simple*
simple-io.asm simple-io.hex simple-io.lst
jan@think ~/P/B/m/simple-io>
```

Beispiel-Workflow: Flashen



Probleme/Lösungen: Kein Zugriff auf Programmer

- Auf Linux-Systemen mit „Hotplugging-Manager“ **udev**:
 - `$ sudo su` #Root-Rechte erhalten
 - `$ touch /etc/udev/rules.d/98-local.rules` # Rules-Datei anlegen
 - `$ cat 'ACTION=="add", SUBSYSTEM=="usb", ATTRS{idVendor}=="03eb", ATTRS{idProduct}=="2104", MODE="0666" >> /etc/udev/rules.d/98-local.rules`
- Jetzt sollte der Programmer wie folgt mit avrdude ansprechbar sein. Programmer mit PC und Evaluations-Board verbinden und dann folgenden avrdude Befehl ausführen:
 - `$ avrdude -c avrispmkII -P usb -p m16 -v` # Test connection

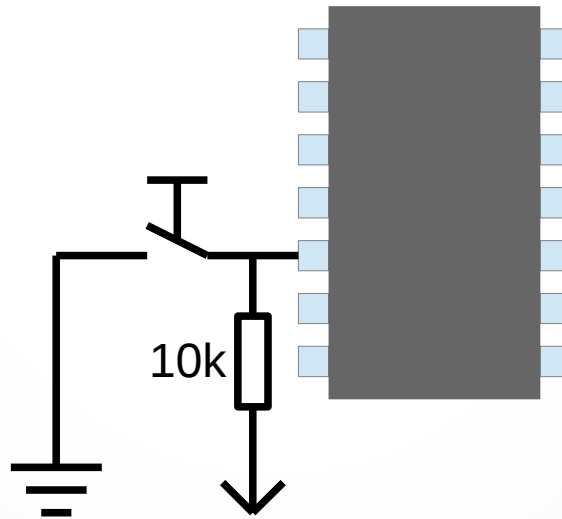
Probleme/Lösungen: Fuses

- Falsche Wahl der Fuses
 - CKSEL0 ... 3 falsch gesetzt => μ C taktet nicht
 - Deaktivierung von SPI => keine Programmierung mehr möglich
 - JTAG aktiv => Entspr. Pins nicht als I/O nutzbar

Fuse	Beschreibung
OCDEN	On-Chip-Debug: Kein kompletter Power-Down
JTAGEN	JTAG Debugging IF (de)aktivieren
SPIEN	Serial Programming IF (de)aktivieren
CKSEL0 ... 3	Clock Source: <ul style="list-style-type: none">• Externer Quarz• Interner oder externer RC Oszillator• Externes Clock Signal

Probleme/Lösungen: I/O

- Input-Pins sollten nicht „offen“ sein
 - Nutzung interner Pull-Up-Widerstand (PUD-Bit in SFIOR Register)
 - Nutzung externer Pull-Up/Down-Widerstand



Probleme/Lösungen: I/O

- Taster an Ports „entprellen“

- SW-Entprellung
- HW-Entprellung (z.B. MC 14490)

```
; Warte auf fallende Flanke an PIND0 (Taste drücken)
loop:  sbic PIND,PIND0          ; Übersprunge nächsten
                                           ; Befehl wenn PIND0 low
                                           ; Herausspringen
        rjmp tast2

; Führe Funktion aus, die mit Tastendruck getriggert wird
inc counter          ; Inkrementiere Counter

rcall pause          ; Verzögerung

; Warte nun auf steigende Flanke (Taste loslassen)
tast1:  sbis PIND,PIND0          ; Übersprunge nächsten
                                           ; Befehl wenn PIND0 high
                                           ; Wiederhole Schleife
        rjmp tast1

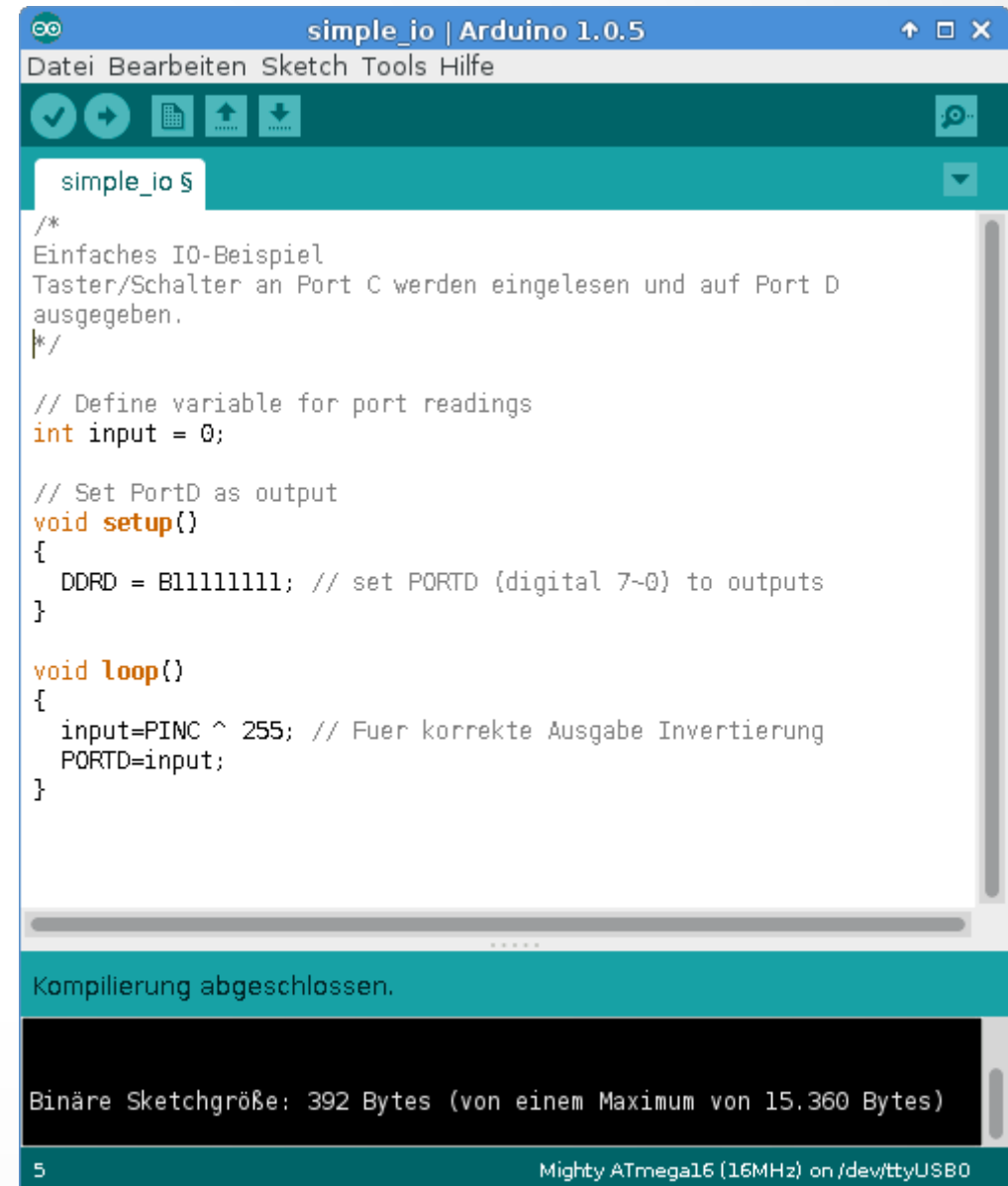
tast2:  rjmp loop                ; Programm-Schleife

; =====

; Verzögerung durch Hochzählen eines 16Bit-Registers
pause:  push r24                 ; Rette Register
        push r25                 ; Rette Register
        clr r24                  ; Low Byte 16Bit Zählreg.
        clr r25                  ; High Byte
pause1: sbiw r24,1               ; dekrementiere Zählreg.
        brne pause1             ; Gehe pause1 solange !=0
        pop r25                  ; Register zurücksetzen
        pop r24                  ; Register zurücksetzen
        ret                     ; Rücksprung
```

Vergleich: avr-gcc vs gavrasm

- Assembler: 52 Bytes
- Arduino/avr-gcc: 392 Bytes



The screenshot shows the Arduino IDE interface. The main window displays a sketch named 'simple_io' with the following code:

```
/*
Einfaches IO-Beispiel
Taster/Schalter an Port C werden eingelesen und auf Port D
ausgegeben.
*/

// Define variable for port readings
int input = 0;

// Set PortD as output
void setup()
{
  DDRC = B11111111; // set PORTC (digital 7~0) to outputs
}

void loop()
{
  input=PINC ^ 255; // Fuer korrekte Ausgabe Invertierung
  PORTD=input;
}
```

Below the code editor, the status bar indicates 'Kompilierung abgeschlossen.' (Compilation completed). The bottom status bar shows 'Binäre Sketchgröße: 392 Bytes (von einem Maximum von 15.360 Bytes)' (Binary sketch size: 392 Bytes (of a maximum of 15.360 Bytes)). The bottom right corner shows the target board 'Mighty ATmega16 (16MHz) on /dev/ttyUSB0'.

Lesenswert

- Bücher
 - G. Schmitt, Mikrocomputertechnik mit Controllern der Atmel AVR-RISC-Familie, Oldenbourg (39,80€)
- WWW
 - <http://www.mikrocontroller.net/articles/AVR-Tutorial>
 - http://www.avr-asm-tutorial.net/avr_de/beginner/
 - <http://www.avrbeginners.net/>