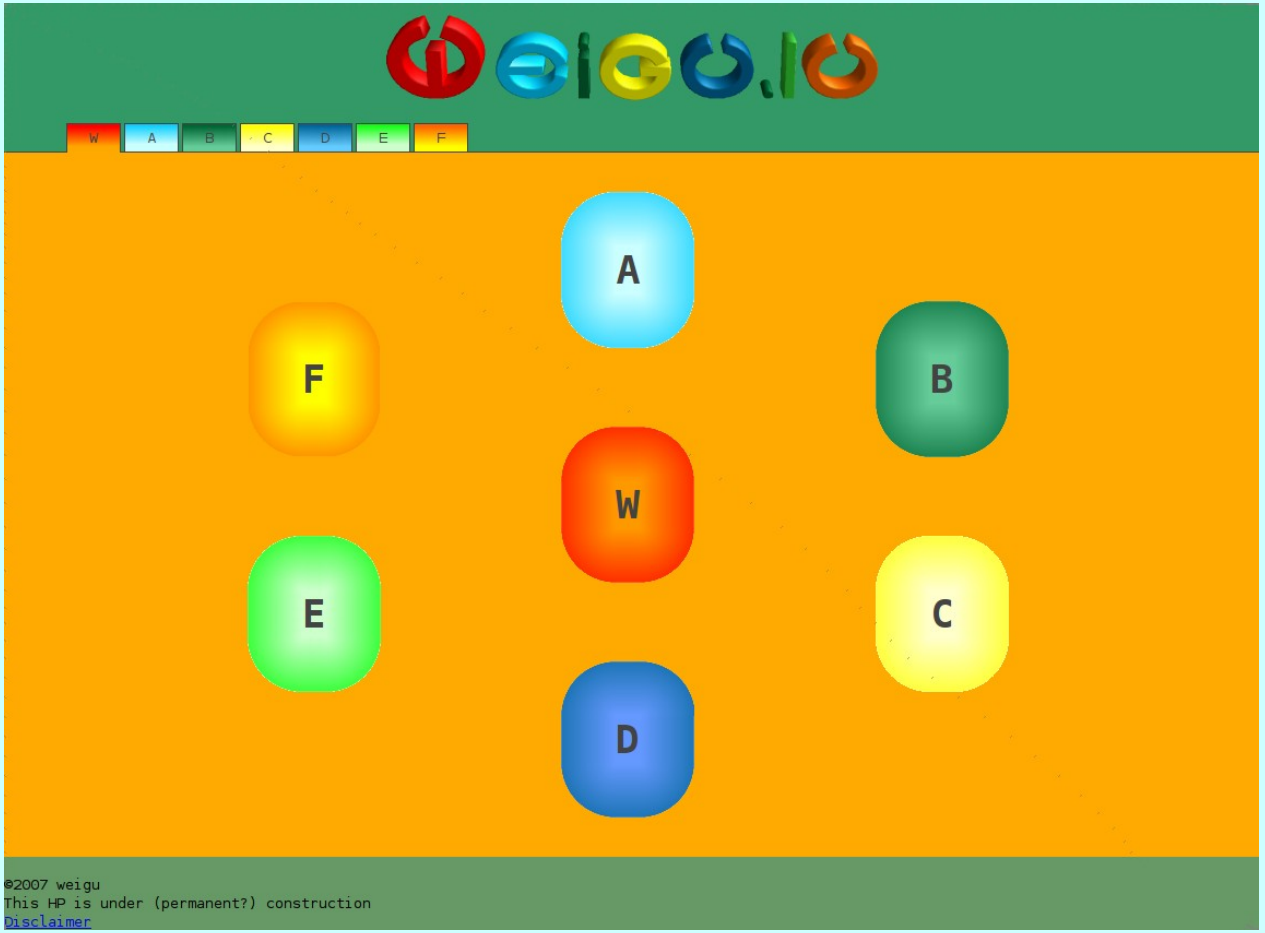
Mikrocontrollerkurs auf weigu.lu



Nach einem Klick auf den Buchstaben A befindet man sich auf der Seite

http://www.weigu.lu/a

Schülermaterial

- **MODUL A Kurs**
- **USB-Stick**
- **ANHANG F**
- **Schnellheft mit gelösten Aufgaben**

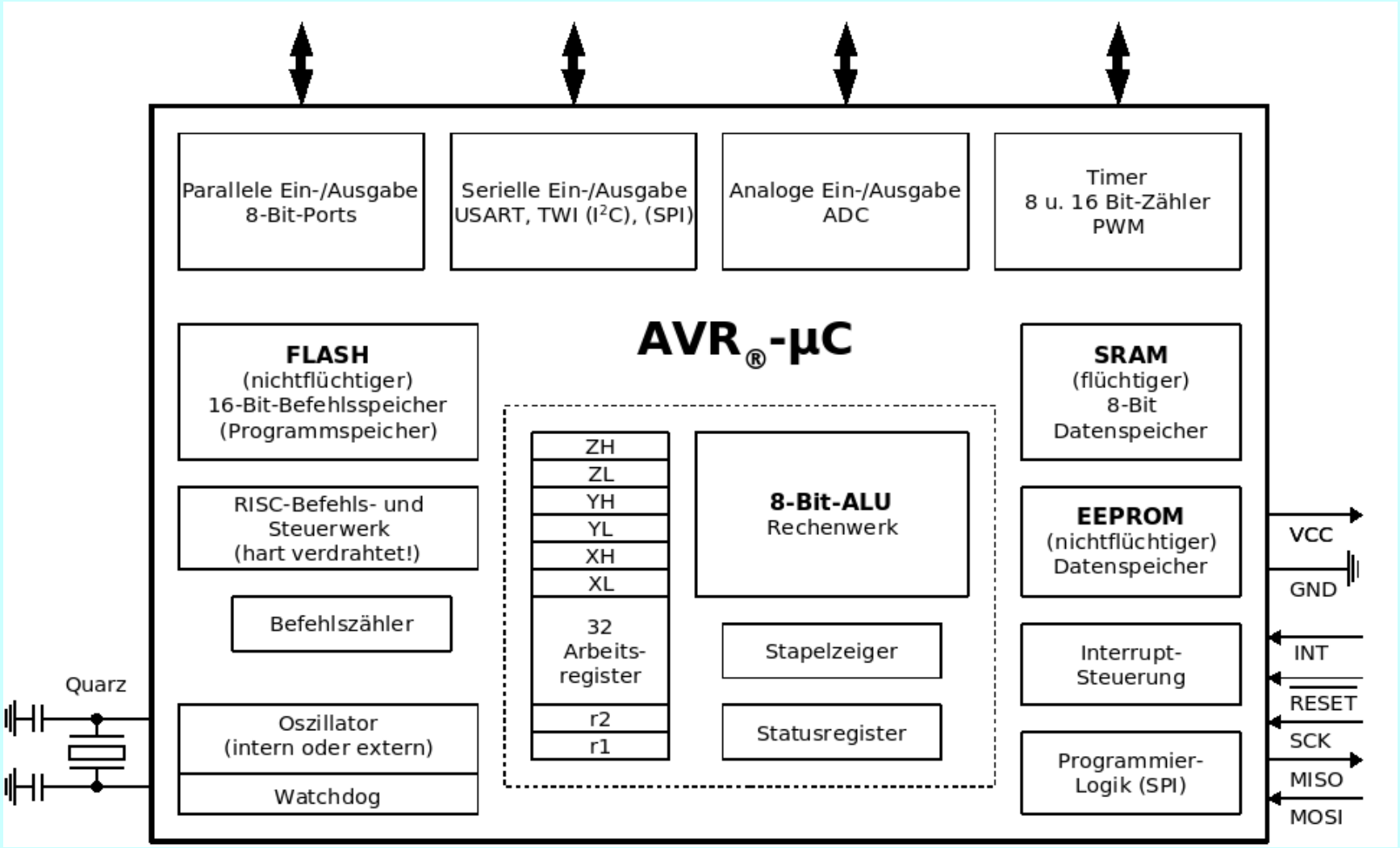
Eigenschaften der AVR-8Bit-RISC Mikrocontroller

Sehr hohe Effizienz durch:

- **RISC** (Reduced Instruction Set Computing):
  einfache kurze Befehle in Hardware implementiert → schnell!
  Meist nur ein Taktzyklus pro Befehl (1Mips/MHz)
  Effizienz durch 32 Arbeitsregister (ALU)
  Befehle sind registerorientiert, nur `load` und `store` greifen auf Speicher zu.
- **Harvard**-Architektur:
  Getrennter Befehls- und Datenspeicher (unterschiedliche Breite möglich, gleichzeitiges Laden und Speichern, parallele Rechenwerke).

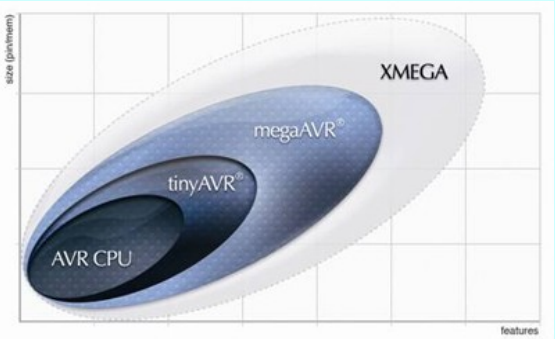
Eigenschaften der AVR-Controller

Mikrocontroller beinhaltet CPU, Speicher und mehrere Ein-/Ausgabebausteine



Eigenschaften der AVR-Controller

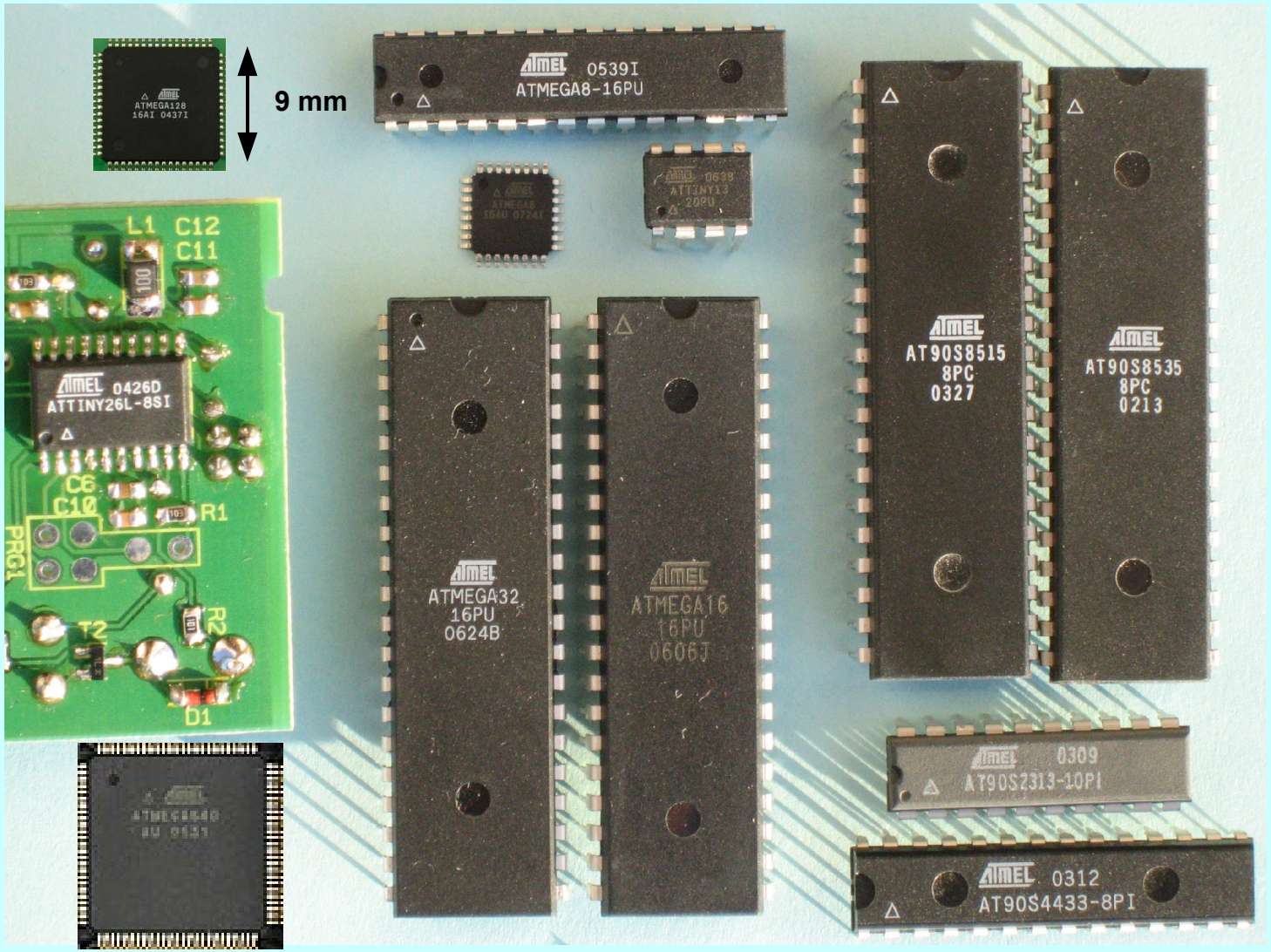
- Viele unterschiedliche Controller und Gehäuseformen mit AVR-Kern: http://www.atmel.com/dyn/products/param_table.asp?family_id=607&OrderBy=part_no&Dir



Quelle: ATMEL

- Systemtakt bis 20MHz!
- Geringer Energieverbrauch:
  1,8V bis 5,5V, 6 verschiedene Schlafmodi,
  schnelles Erwachen, unterschiedliche Frequenzen
- Neu und Umprogrammierung während des Betriebes (Bootloader)
- In-System Programming (ISP)
  On-Chip Debugging and In-system verification (JTAG-Interface)
- Gratis Entwicklungsumgebung Studio 4

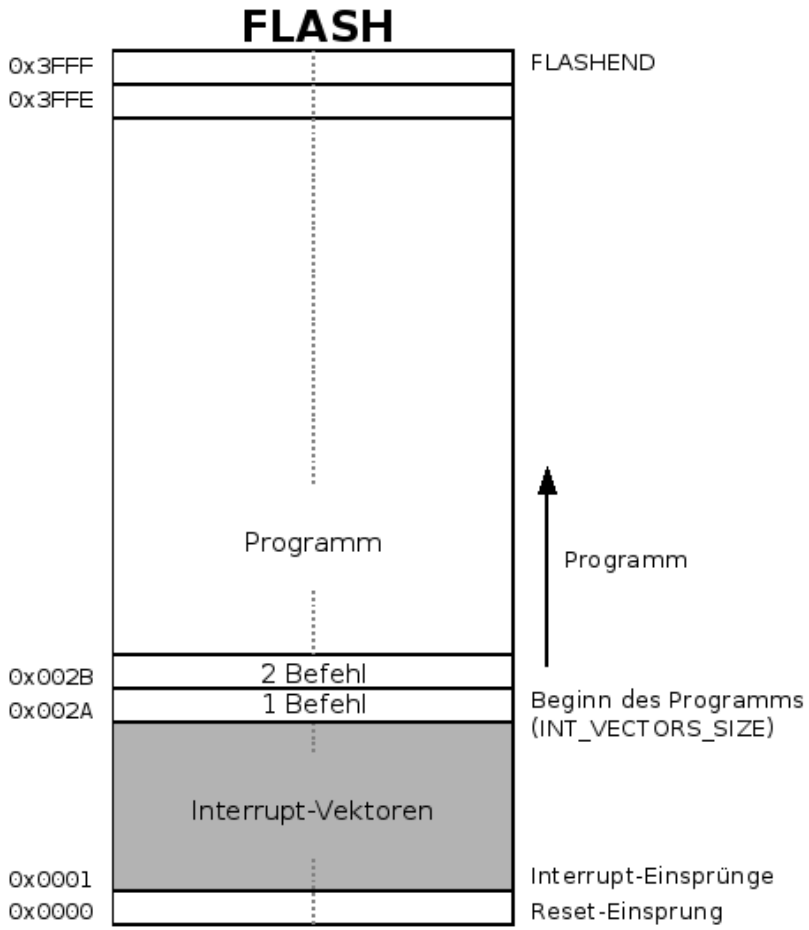
Verschiedene Controller



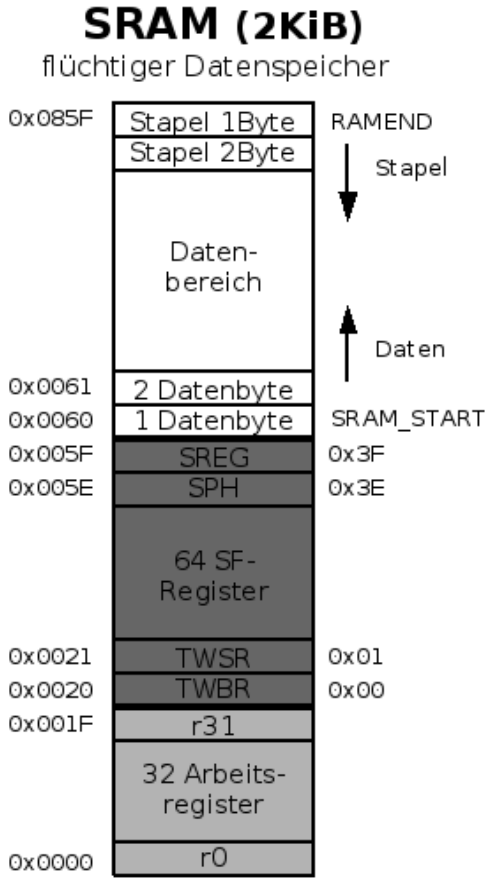
Memory Map

ATMega32

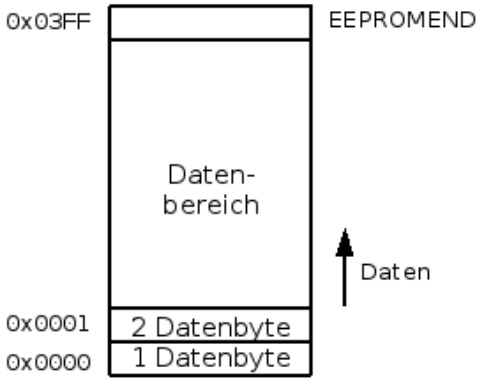
Programmspeicher (32KiB)
nichtflüchtiger Speicher, 16 Bit breit



Datenspeicher
8 Bit breit



EEPROM (1KiB)
nichtflüchtiger Datenspeicher



Assemblerprogrammierung

- Wort **Assembler** hat **2 Bedeutungen**:
  1. **maschinennahe Computersprache**
  2. **Übersetzungsprogramm** Assembler –Maschinensprache
- **Assemblerprogramm = reine Textdatei**
  (mit der Endung .asm). Wird durch den Assembler in die Maschinensprache (Hexdatei *.hex) übersetzt.
- **strukturierter übersichtlicher Code** durch:
  Assemblerdirektiven, Include Dateien, Unterprogramme, bedingte Adressierung und Makros.
- **Assembler-Programmiervorlage**
  um die Programmierung zu Beschleunigen wird mit beim Programmieren mit einer Vorlage (A2_template.asm) gearbeitet.

Assemblerprogrammierung

```
[Label:]Direktive oder Befehl [Operanden]  [;Kommentar]
```

- **Label** sind erfundene **Namen** für Speicheradressen
  Der Assembler kümmert sich um die Zuweisung.
- **Direktiven** sind **Pseudobefehle** des Assembler
  Diese Assembleranweisungen dienen der Steuerung des Übersetzungsvorgangs und werden nicht in Maschinencode übersetzt. Sie beginnen mit einem Punkt und stehen meist am Anfang der Zeile.
- **Operanden** sind von Befehlen oder Direktiven benötigte zusätzliche Informationen.
- **Kommentare** beginnen mit einem Strichpunkt.

Assembleranweisungen

- **.DEF** Name = Register — Bsp.: .DEF Tmp1 = r16
- **.EQU** Name = Ausdruck — Bsp.: .EQU Tab = 0x1AF
- **.SET** Name = Ausdruck — Bsp.: .SET Cnt = 20 (änderbar)
- **.INCLUDE** "Textdatei" — Bsp.: .INCLUDE "m32def.inc"
- **.ORG** Adresse — Bsp.: .ORG 0xD000
- **.CSEG (.DSEG, .ESEG)**
- **.DB (.DW)** Liste — Bsp.: .DB 0,0xFF,'A', "Hallo",0b11001100
- **.BYTE** Anzahl — Bsp.: .BYTE 5
- **.EXIT**

Assemblerbefehle

- Besteht aus **Opcode** und **Operand**
  Bsp: **`cli, ser r16, ldi r16,100`**
- Bei **zwei Operanden**: Befehl **Ziel,Quelle**
- Arithmetische und logische Befehle (Arbeitsregister, beeinflussen Flags (SREG), **`add, eor, inc, sbiw, mul`**), Datentransferbefehle (**`mov, ldi, in, out, st, ld, sts, lds, push, pop, lpm`**) Sprungbefehle (**`rjmp, jmp, breq, call, reti`**), Bitorientierte Befehle (z.B. Flags, **`sec, lsl`**), Sonstige Befehle.(**`nop`**)

Besonderheiten:

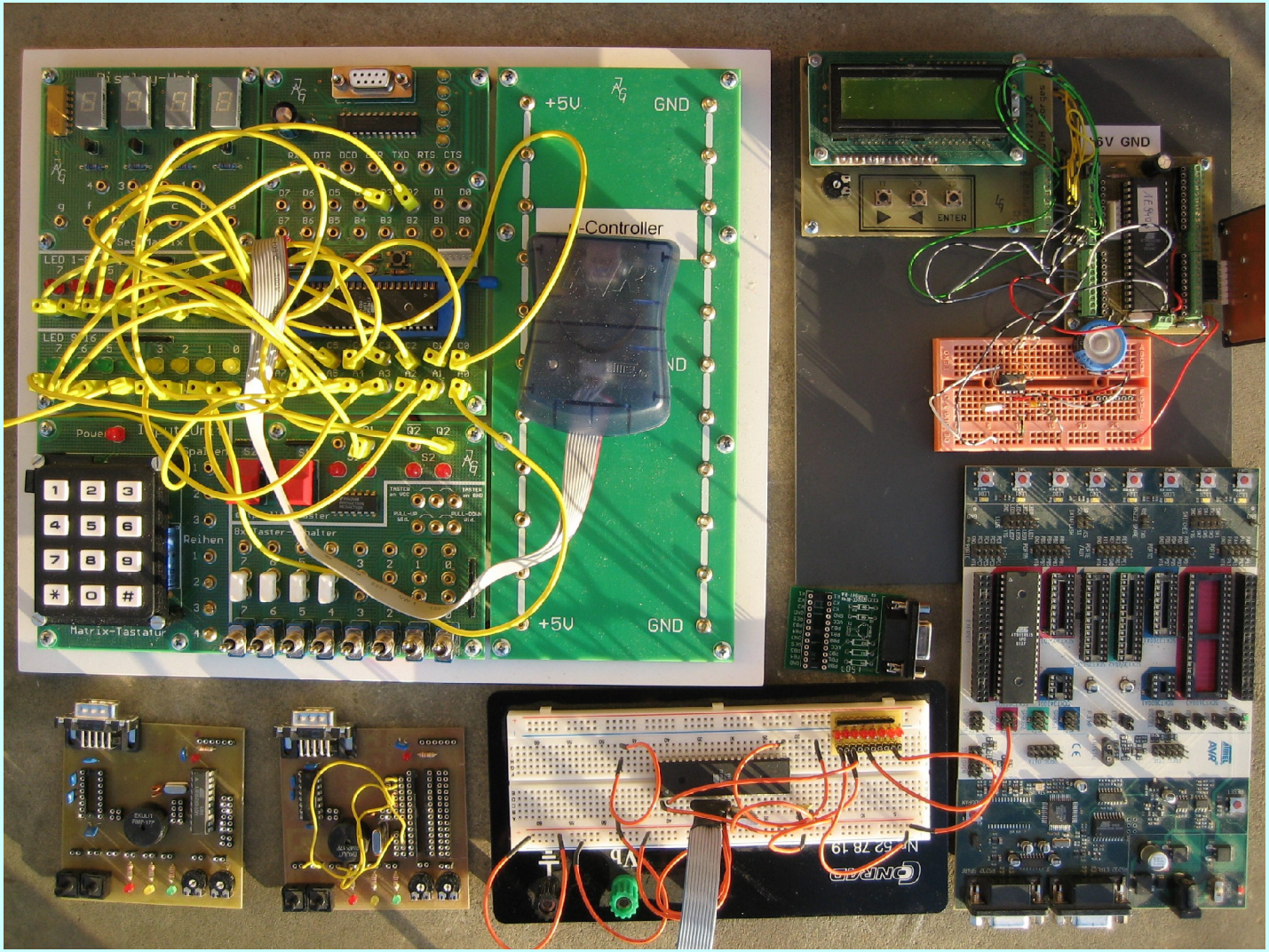
- unmittelbare Adressierung (**`ldi`**) nur bei Arbeitsregister r16-r31
- Bitweise Adressierung der unteren 32 SF-Register mit **`sbi`** und **`cbi`** und bitweise Abfrage mit **`sbic`** bzw. **`sbis.`**

A2_template.asm

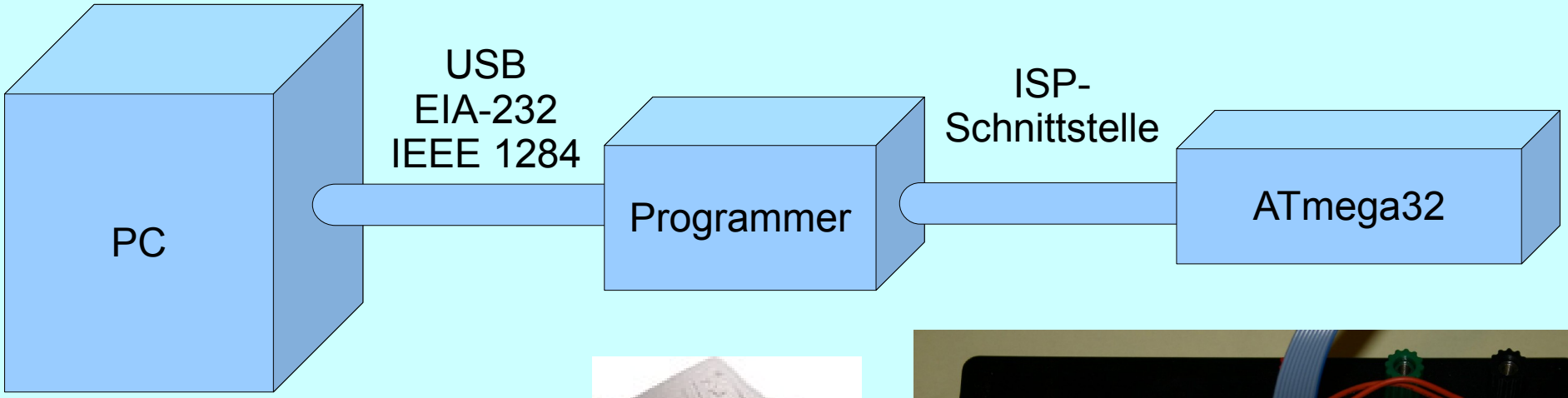
http://www.weigu.lu/a/asm/A2_template.asm

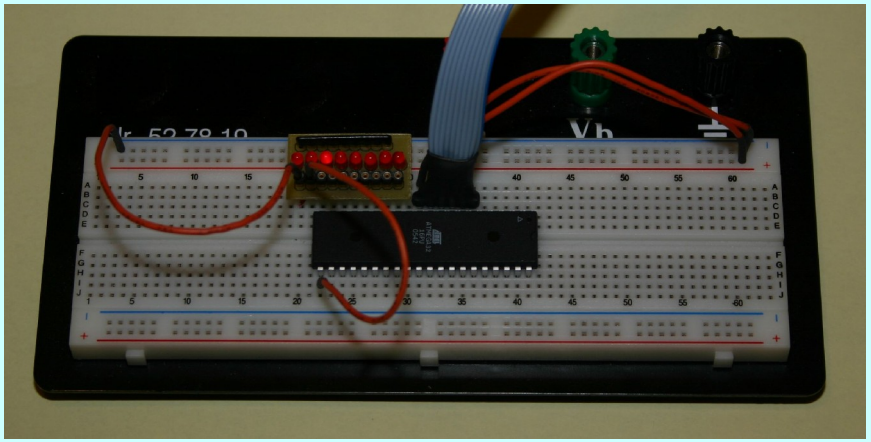
http://www.weigu.lu/a/asm/m32def.inc

Benötigte Hardware



Benötigte Hardware



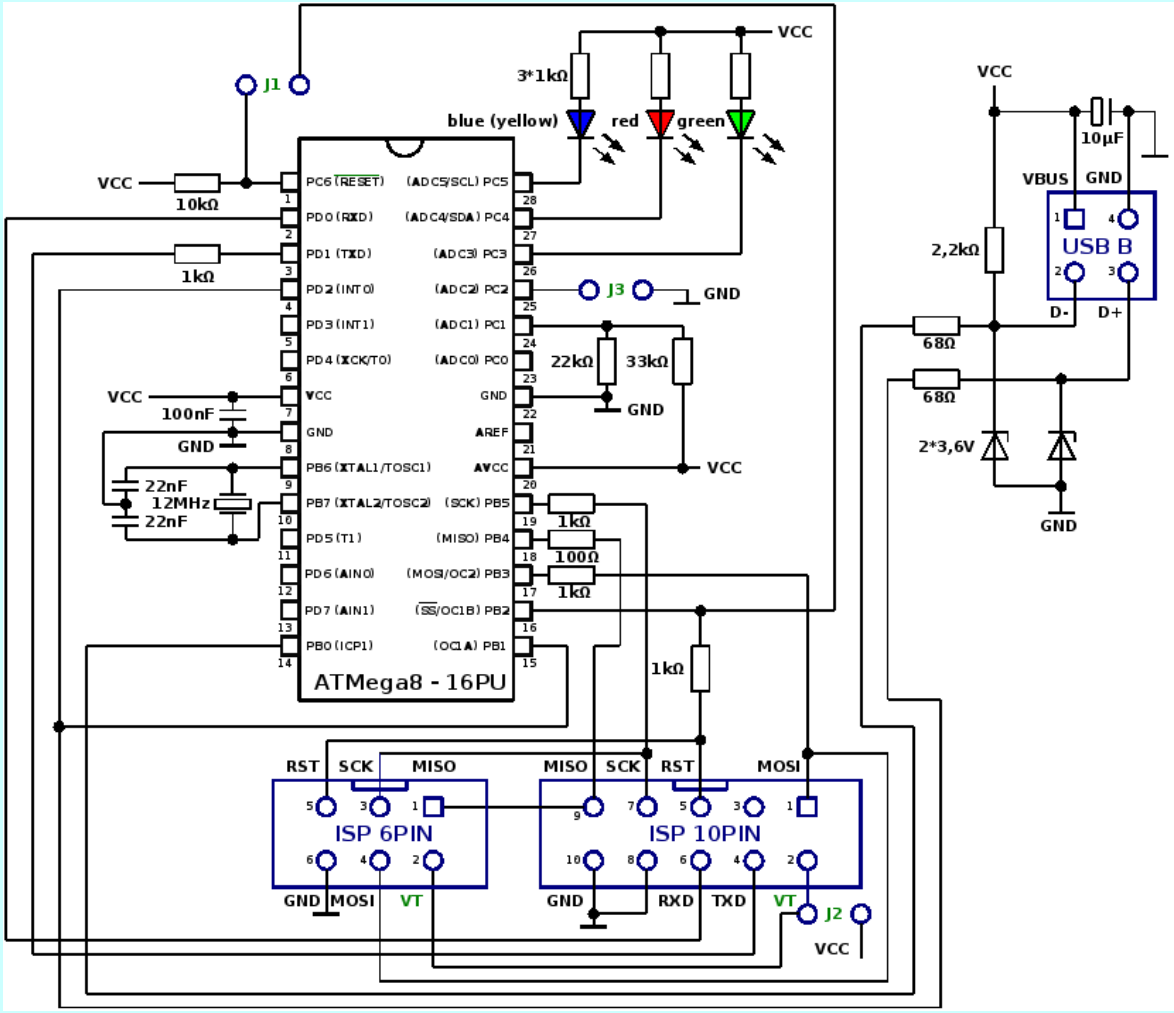


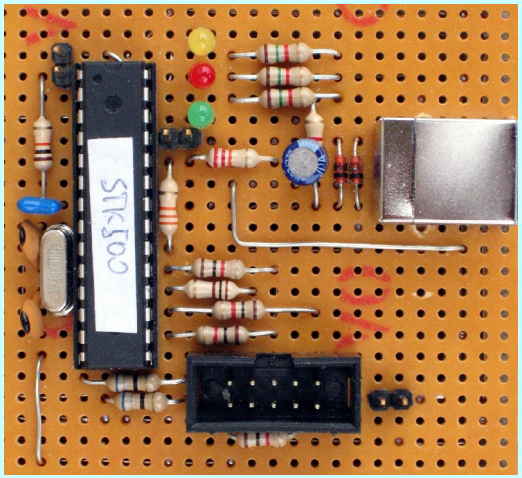
Programmer

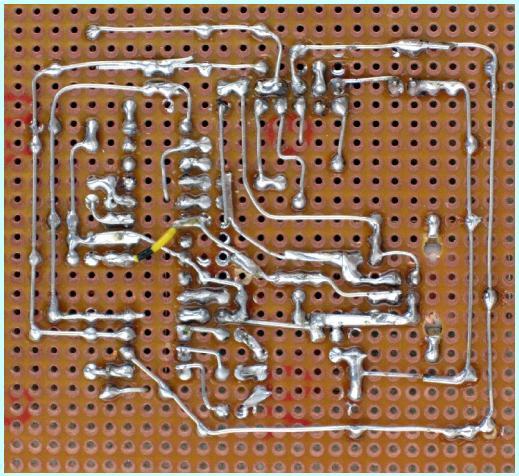
- **Original ATMEL** (kompatibel Studio 4 u. Bascom):
  Parallel: STK200, STK300
  Seriell: STK500, AVRISP, AVR-JTAGICE mkII
  USB : **AVRISP mkII, AVR Dragon, AVR-JTAGICE mkII**
- **Nachgebaut:**
  Parallel: STK200-kompatibel, SP12 Programmer
  Seriell: AVR910, SI-Prog, Sercon2
  USB: **USB AVR-ISP, usbprog** (elektor 10/07)
  USBisp, USB avrisp, Evertool, USBasp, AvrUsb500, AVR-Doper, USBtinyISP, UCOM-IR,

http://www.mikrocontroller.net/articles/AVR_In_System_Programmer

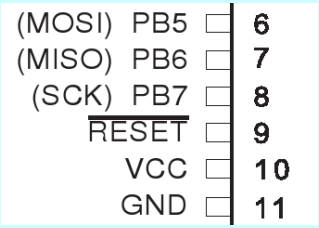
USB-AVR-ISP

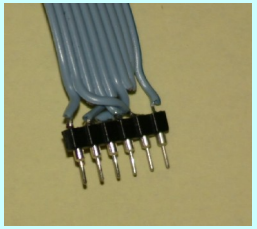

VCC
J1
3*1kΩ
blue (yellow)
red
green
VCC
10kΩ
1kΩ
PC6 (RESET)
PD0 (RXD)
PD1 (TXD)
PD2 (INT0)
PD3 (INT1)
PD4 (XCK/T0)
VCC
GND
PB6 (XTAL1/TOSC1)
PB7 (XTAL2/TOSC2)
PD5 (T1)
PD6 (AIN0)
PD7 (AIN1)
PB0 (ICP1)
(ADC5/SCL) PC5
(ADC4/SDA) PC4
(ADC3) PC3
(ADC2) PC2
(ADC1) PC1
(ADC0) PC0
GND
AREF
AVCC
(SCK) PB5
(MISO) PB4
(MOSI/OC2) PB3
(SS/OC1B) PB2
(OC1A) PB1
ATMega8 - 16PU
J3
GND
22kΩ
33kΩ
GND
VCC
VCC
100nF
GND
22nF
12MHz
22nF
1kΩ
100Ω
1kΩ
1kΩ
RST SCK MISO
ISP 6PIN
GND MOSI VT
MISO SCK RST MOSI
ISP 10PIN
GND RXD TXD VT
J2
VCC
VCC
10µF
VBUS GND
USB B
D- D+
2,2kΩ
68Ω
68Ω
2*3,6V
GND

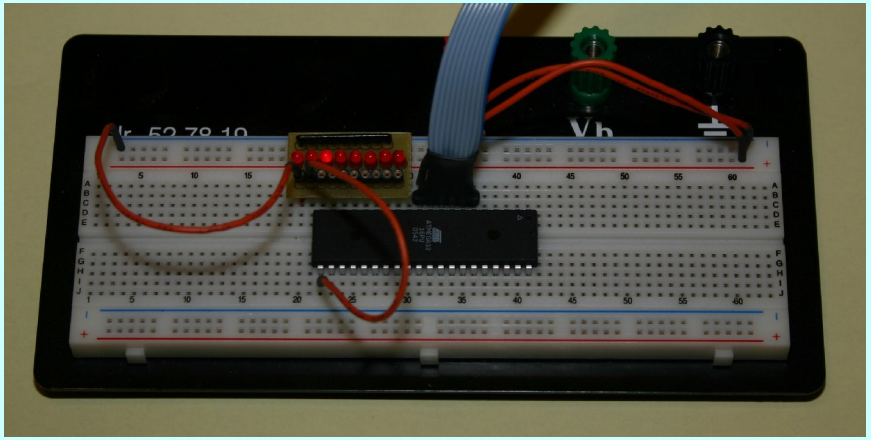


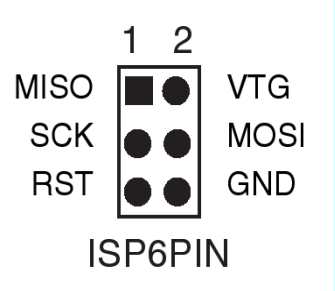


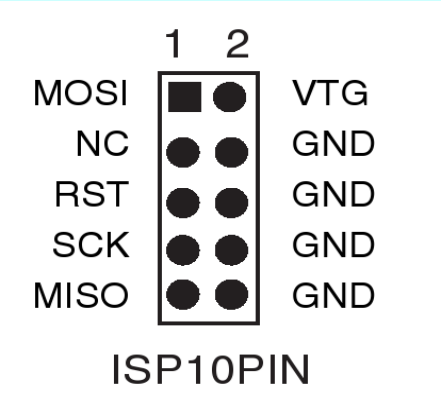
ISP-Schnittstelle

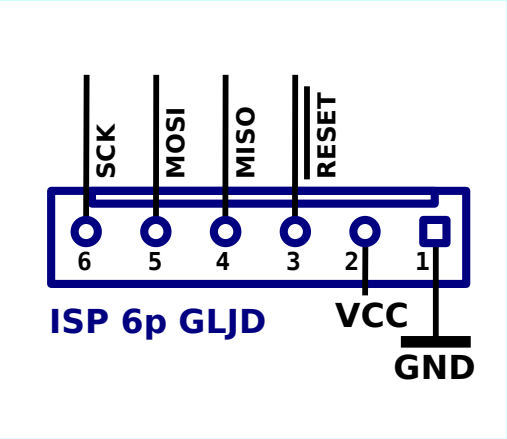












Software

- Entwicklungsumgebungen:
  Studio 4 (Win), WinAVR (AVR-GCC), AVR Eclipse (Win + Linux), KontrollerLab (Linux), CodeVisionAVR C (Win, komm.), Bascom (Win)
- freie Assembler: tavrasm, avra, ava, avr-as, gavrasm
- freie Debugger: simulavr, AVR-GDB
- freie C-compiler: avr-gcc (avr-libc)
- freie Programmer: avrdude

Studio 4

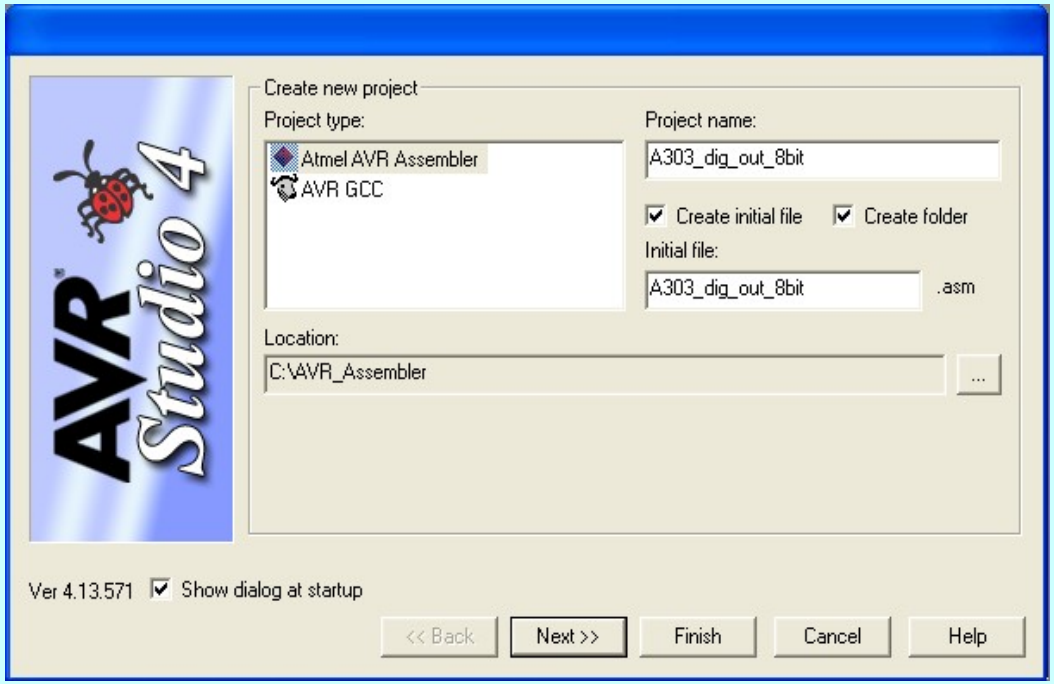
1. Doppelklick auf
   Icon Studio 4
   oder
   "Start" "Programs" "Atmel AVR Tools" "Studio 4".

2. Für jede Aufgabe ein
   neues Assemblerprojekt!

   Projektname = Name der Aufg.
   (zB A303_dig_out_8bit).

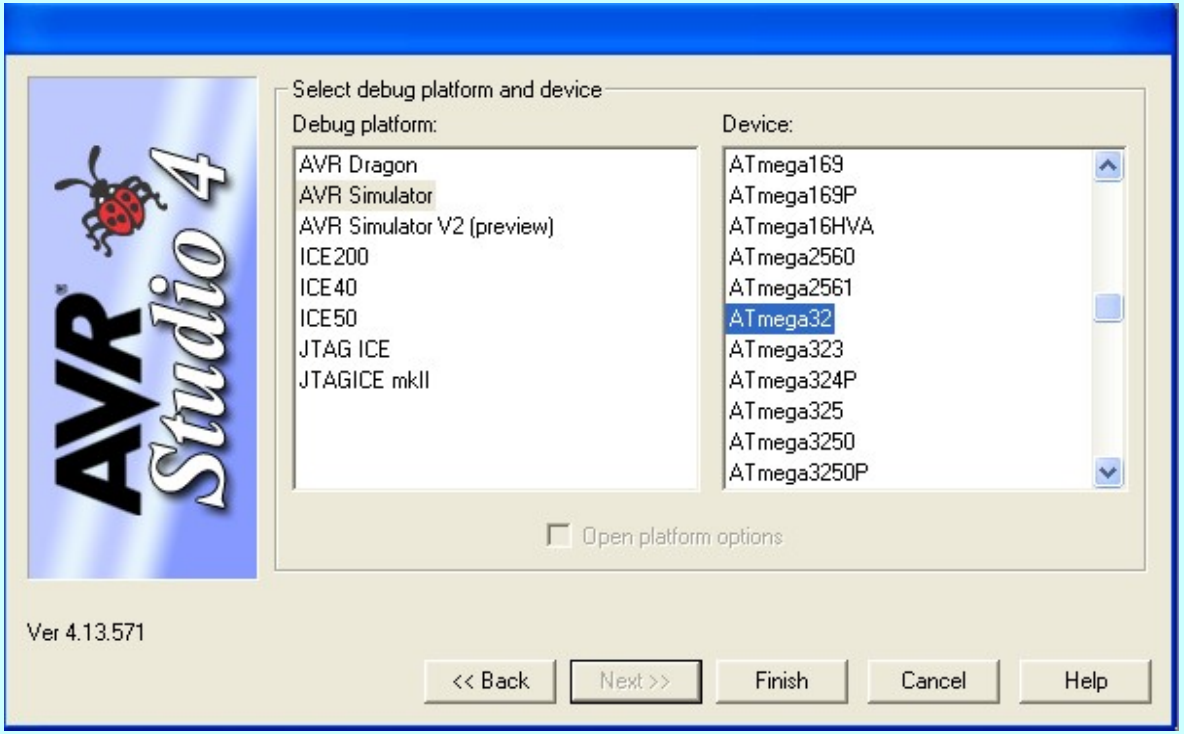
   Neues Unterverzeichnis und
   neue Datei erstellen.

   Richtiges Verzeichnis
   (E:\T2EC_1 statt C\AVR-Assembler)
   in "Location" angeben dann
   "Next>>".



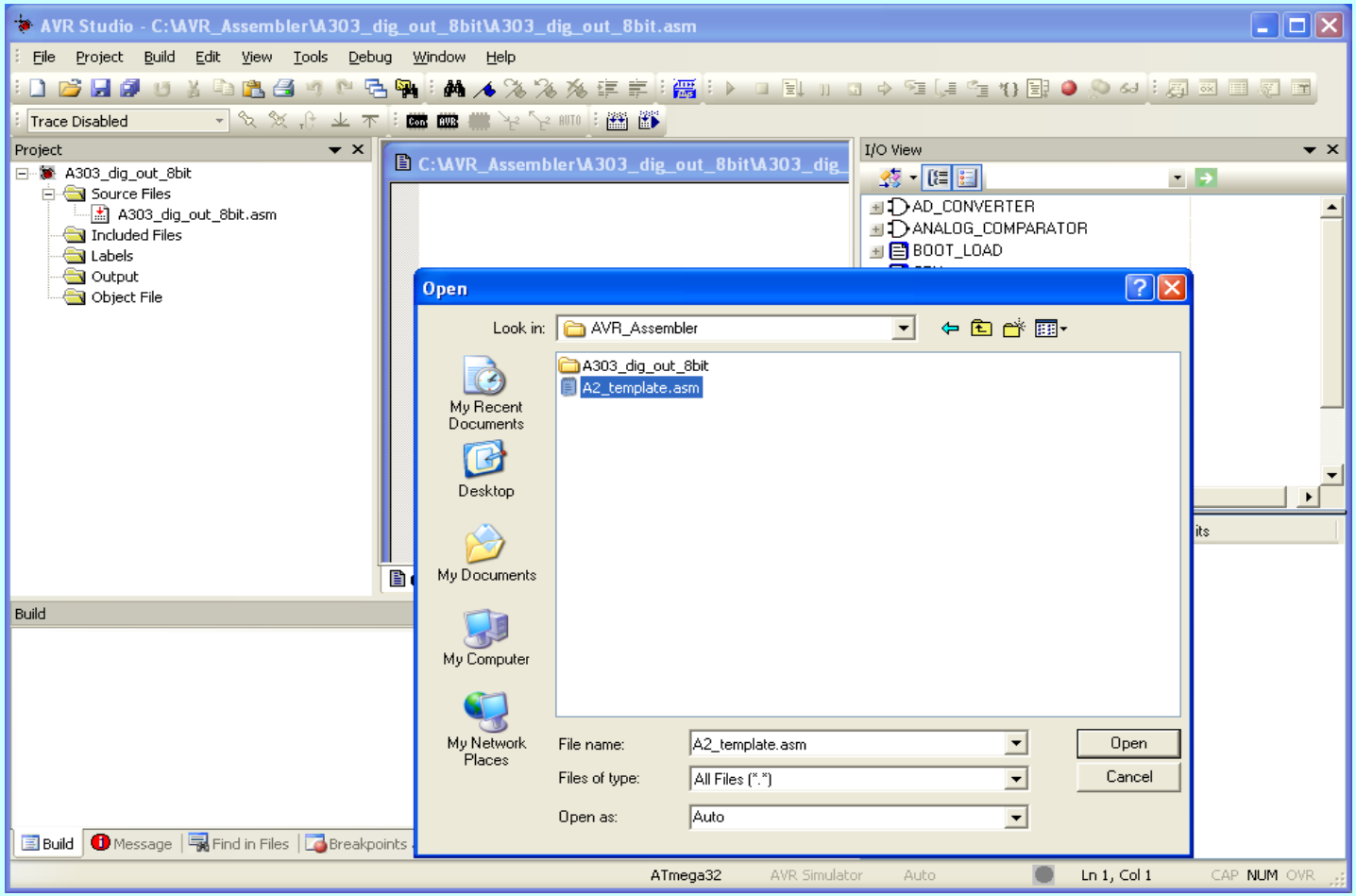
Studio 4

3. AVR Simulator und ATmega32 auswählen.
   Dann "Finish".



Studio 4

4. Klick "File" "Open File...". Datei "A2_template.asm" öffnen. Inhalt nach A303_dig_out_8bit.asm kopieren.



Studio 4

5. Programm beliebig erweitern und abspeichern.
   (Achtung: Richtige Pfadangabe für Definitionsdatei!)

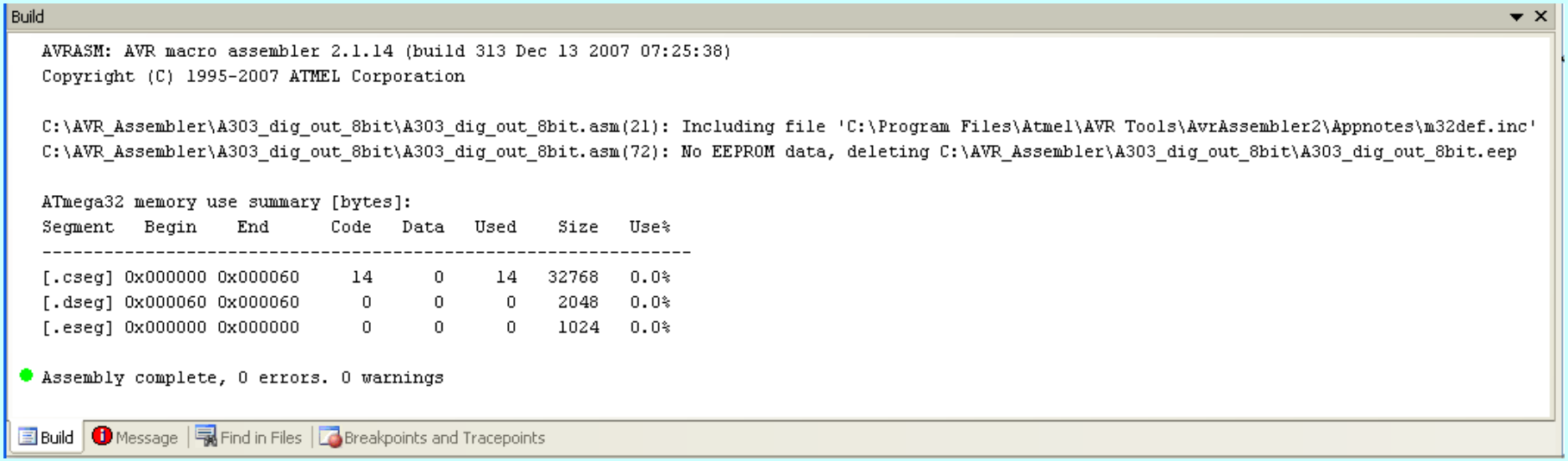
C:\AVR_Assembler\A303_dig_out_8bit\A303_dig_out_8bit.asm *

```
;*****************************************************************************
;*
;*      Titel:  Mein erstes Programm zur digitalen Ausgabe
;*              (A303_dig_out_8bit.asm)
;*      Datum:  11/02/08        Version:        0.1
;*      Autor:  Pitti
;*
;*      Informationen zur Beschaltung:
;*      Prozessor:                      Quarzfrequenz:
;*      Eingaenge:
;*      Ausgaenge:
;*
;*      Informationen zur Funktionsweise:
;*
;*
;*****************************************************************************

;-----------------------------------------------------------------------------
;       Assembler Direktiven (u.a. Definitionsdatei einbinden)
;-----------------------------------------------------------------------------
.NOLIST                         ;List-Output ausschalten
.INCLUDE "E:\AVR_Assembler\m32def.inc"
                                ;AVR-Definitionsdatei einbinden
.LIST                           ;List-Output wieder einschalten
.CSEG                           ;was ab hier folgt kommt in den Flash-Speicher

;-----------------------------------------------------------------------------
;       Startadresse 0x0000 und ueberspringe ISR Vektoren
;-----------------------------------------------------------------------------
.ORG    0x0000                  ;Programm beginnt an der SRAM-Adresse 0h
RESET:  rjmp    INIT            ;springe nach INIT
```

C:\AVR_Assembler\A303_dig_out_8bit\A303_dig_out_8bit.asm *

Studio 4

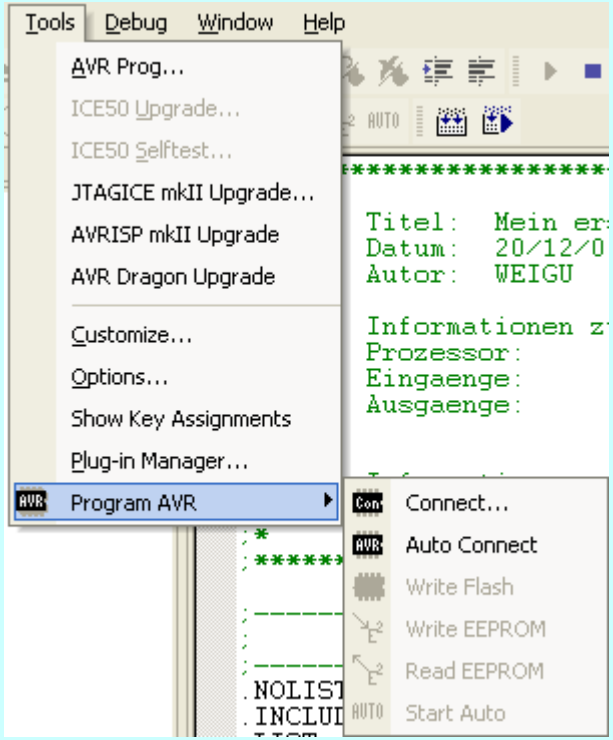
6. Programm assemblieren mit "Build".

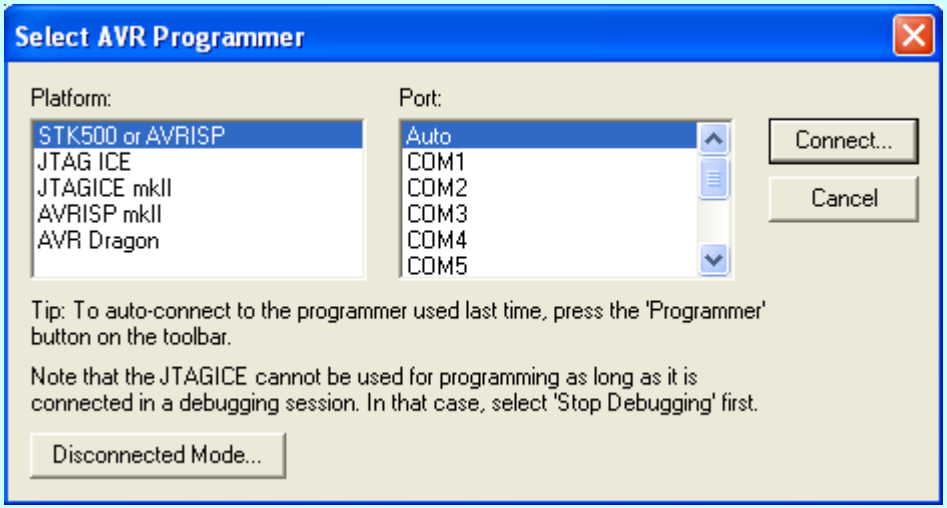


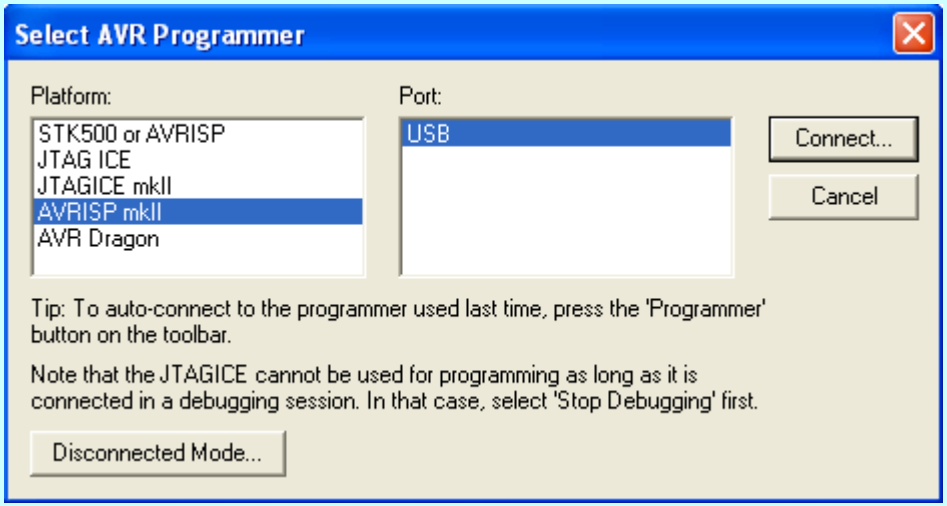
7. Falls Fehler debuggen.

Studio 4

8. "Connect"-Icon anklicken und Programmiergerät auswählen.

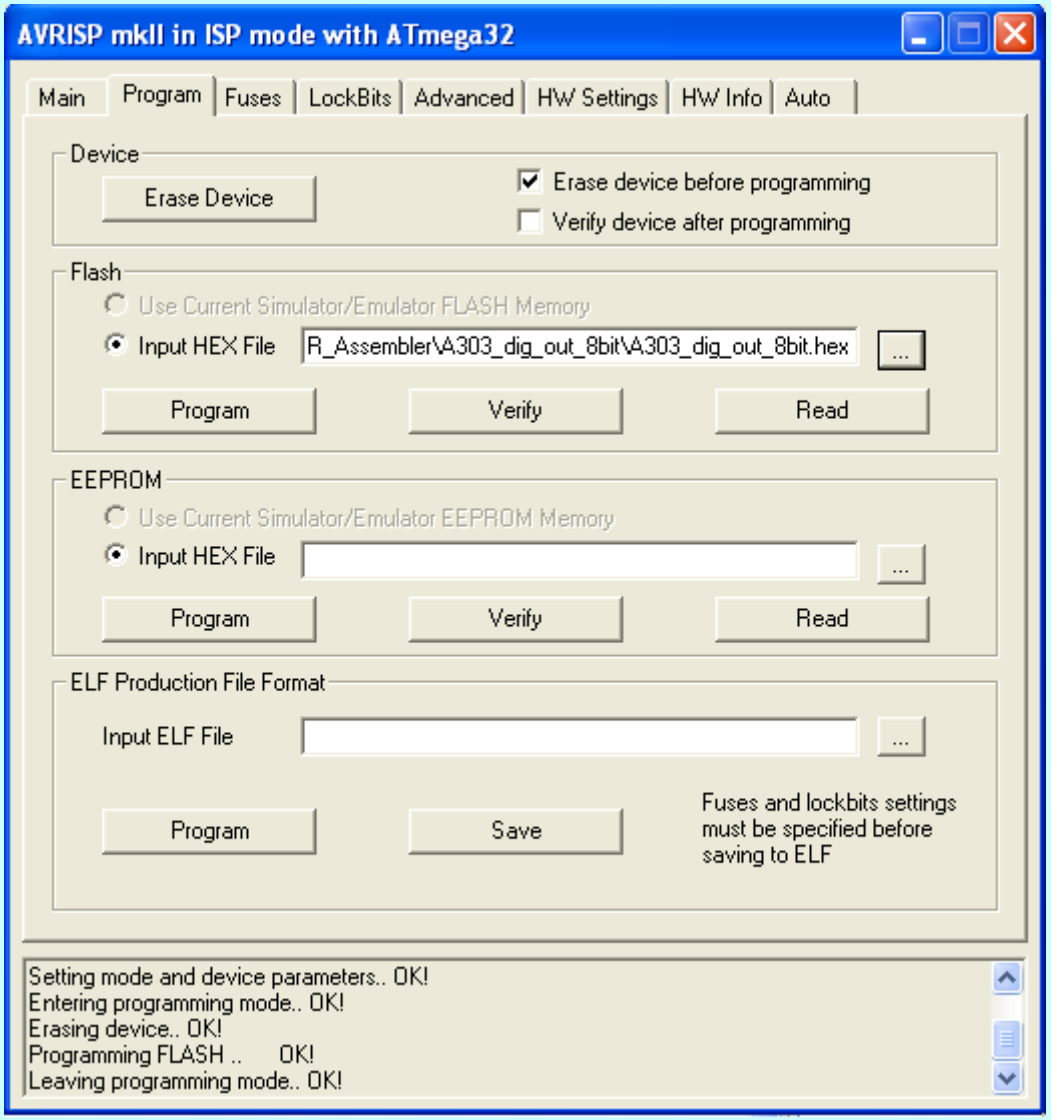






Studio 4

9. Hexdatei auswählen und "Program" anklicken.



Studio 4 Zusammenfassung:

- Projekt öffnen
- Vorlage (A2_template.asm) in "Entry"-Datei kopieren und Programm erstellen
- Assemblieren mit "Build" 
- Falls Fehler ev. Debuggen (Simulieren)  
- Programmieren