AVR-Simulator

https://www.mikrocontroller.net/topic/420097

Das Programm soll Anfängern unter Linux helfen, die ersten Schritte in der Assemblerprogrammierung für AVR zu bewältigen, ohne in teure und aufwändige Hardwaredebugger (und Bausteine, die HW-Debugging unterstützen) investieren zu müssen. Auch wenn es nicht den Funktionsumfang des Simulators im AVR-Studio erreichen wird, wird es auch interessierten Windows-Nutzern als Binärpaket zum Ausprobieren, zur Verfügung gestellt.

Hintergrund

Das Programm entstand vor dem Hintergrund, das AVR-Studio 4 unter WINE nur sehr unzuverlässig funktioniert. Die Kombination aus "simulavr" + "avr-gdb" und gegebenenfalls "DDD" funktioniert gut, für Projekte die überwiegend in C erstellt sind. Kleine Projekte, die ausschließlich in Assembler programmiert sind, lassen sich nur auf Binärebene debuggen, was wohl auch daran liegt, das der gdb auf die Debug-Informationen in der ELF-Datei angewiesen ist, welche aber nicht alle Assembler erzeugen. Außerdem erscheint dieses Konstrukt gerade für Jemanden, der gerade erst in die Mikrocontrollertechnik einsteigen möchte, meiner Meinung nach, etwas abschreckend.

Zweck

Simuliert wird primär der Prozessorkern (CPU). Die zukünftige Implementierung allgemeiner Peripherie ist angedacht. Hier kommen primär Komponenten in Frage, die in vielen Bauelementen in gleicher Weise implementiert sind. Bis dahin werden die I/O-Register wie RAM behandelt, können also beschrieben, und auch wieder zurück gelesen werden.

verarbeitbare Dateiformate

OBJ

Gelesen werden Objektdateien (.obj), die sowohl vom ursprünglichen, proprietären AVRASM2, dem freien avra, sowie dem ebenfalls freien Universalassembler "AS" von Alfred Arnold geschrieben werden können. Diese enthalten neben dem eigentlichen Programm, auch Informationen, zu den ursprünglichen Zeilennummern in den Quelldateien.

ELF

Elf-Dateien können Programmcode für viele verschiedene Plattformen und Prozessortypen enthalten, unter Anderem auch für AVR. Zusätzlich können sie sehr detaillierte Debuginformationen enthalten. Die Verarbeitung der Debuginformationen funktioniert grundsätzlich schon, ist aber noch nicht fertig.

BIN,HEX

Ebenfalls ist es auch möglich, Intel-Hexdateien (.hex), oder rohe Binärdateien (.bin) einzulesen. Da hier aber keine Debug-Informationen vorliegen, kann der Programmablauf hier nur anhand der Disassemblerausgabe verfolgt werden. Binärdateien werden ab Adresse 0 in den virtuellen Flash geladen, Hexdateien enthalten Zieladressen, und werden in diese geladen.

Installation

Der Simulator wird im Quelltext bereitgestellt und benutzt das FLTK-Framework.

Linux

Bevor das Programm compiliert werden kann, ist sicherzustellen, das FLTK installiert ist:

```
sudo apt-get install libfltk1.3-dev
sudo apt-get install libx11-dev
```

Im Verzeichnis "src" befindet sich das Shellscript "c", welches die Compilierung durchführt.

Je nachdem, wie die Quelldateien ausgepackt wurden (tar -xf, oder Archivverwaltung), kann es erforderlich sein, das "c"-Script ausführbar zu machen (chmod u+x c), dann führt man es aus: (./c)

Die dabei entstandene Programmdatei "avrsim" kann dann bei Bedarf, an einen geeigneteren Ort verschoben, oder an diesem Ort ausgeführt werden (./avrsim).

Als Beispiel, soll hier gezeigt werden, wie der Simulator auf einem Raspberry Pi gebaut wird.

Benötigt wird das jeweils neueste Archiv mit den Quelldateien "src_datum-uhrzeit.tar.gz", sowie später die Beschreibungen der Periperien "iodefs_datum-uhrzeit.tar.gz", dazu am Besten, den Thread von unten nach oben durchsuchen. Hier wird davon ausgegangen, das sich die Dateien im Unterverzeichnis "Downloads" des Heimatverzeichnisses befinden.

```
pi@raspi-11:~ $ cd Downloads/
pi@raspi-11:~/Downloads $ ll
insgesamt 624
-rw-r--r-- 1 pi pi 500012 Aug  8 20:30 iodefs_20190808-2020.tar.gz
-rw-r--r-- 1 pi pi 131776 Aug  8 20:30 src_20190427-1040.tar.gz
pi@raspi-11:~/Downloads $
```

Jetzt gehen wir zurück in unser Heimatverzeichnis, und legen dort ein Verzeichnis "avrsim" an:

```
pi@raspi-11:~/Downloads $ cd
pi@raspi-11:~ $ mkdir avrsim
pi@raspi-11:~ $ cd avrsim
pi@raspi-11:~/avrsim $
```

In dieses wechseln wir hinein, und entpacken die beiden Archive:

```
pi@raspi-11:~/avrsim $ tar -xf ~/Downloads/iodefs_20190808-2020.tar.gz
pi@raspi-11:~/avrsim $ tar -xf ~/Downloads/src_20190427-1040.tar.gz
pi@raspi-11:~/avrsim $ ll
insgesamt 8
drwxr-xr-x 3 pi pi 4096 Mär  4  2017 iodefs
drwxr-xr-x 4 pi pi 4096 Apr 27 10:41 src
pi@raspi-11:~/avrsim $
```

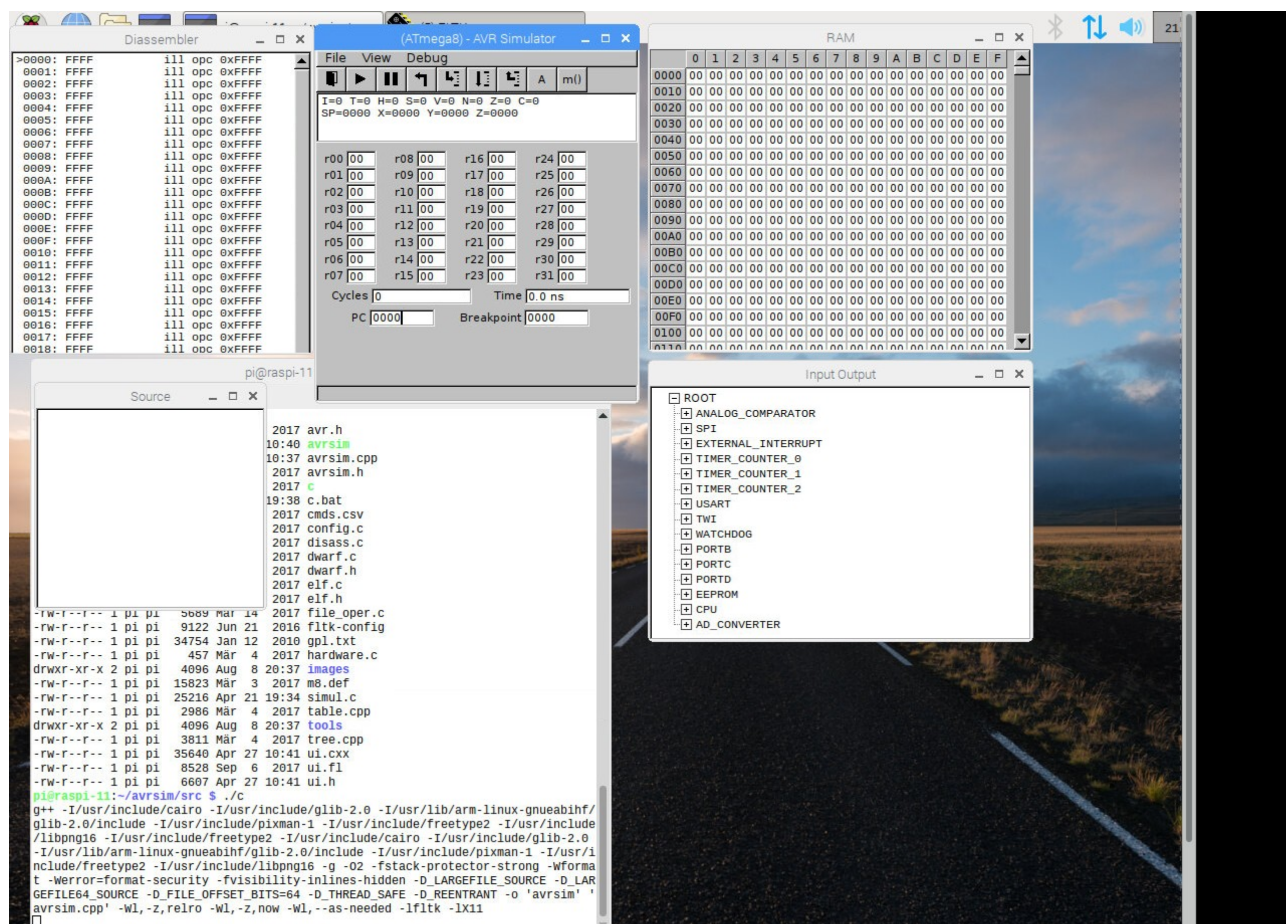
hier sind jetzt die beiden Verzeichnisse "iodefs" und "src" entstanden, in letzteres wechseln wir jetzt:

```
pi@raspi-11:~/avrsim $ cd src
pi@raspi-11:~/avrsim/src $
```

Hier finden wir ein Shellscript namens "c", welches ausführbar sein sollte:

```
pi@raspi-11:~/avrsim/src $ ll
insgesamt 572
-rw-r--r-- 1 pi pi 160237 Mär  4  2017 avr.h
-rwxr-xr-x 1 pi pi 149752 Apr 27 10:40 avrsim
-rw-r--r-- 1 pi pi   9376 Apr 27 10:37 avrsim.cpp
-rw-r--r-- 1 pi pi   1045 Mär 13  2017 avrsim.h
-rwxr--r-- 1 pi pi     87 Feb 25  2017 c
-rw-r--r-- 1 pi pi    315 Apr 21 19:38 c.bat
-rw-r--r-- 1 pi pi   1549 Feb 21  2017 cmds.csv
-rw-r--r-- 1 pi pi   7943 Mär  4  2017 config.c
-rw-r--r-- 1 pi pi  13764 Feb 25  2017 disass.c
-rw-r--r-- 1 pi pi   4026 Mär 11  2017 dwarf.c
-rw-r--r-- 1 pi pi  16653 Mär  2  2017 dwarf.h
-rw-r--r-- 1 pi pi   4285 Sep  4  2017 elf.c
-rw-r--r-- 1 pi pi   1080 Mär 10  2017 elf.h
-rw-r--r-- 1 pi pi   5689 Mär 14  2017 file_oper.c
-rw-r--r-- 1 pi pi   9122 Jun 21  2016 fltk-config
-rw-r--r-- 1 pi pi  34754 Jan 12  2010 gpl.txt
-rw-r--r-- 1 pi pi    457 Mär  4  2017 hardware.c
drwxr-xr-x 2 pi pi   4096 Aug  8 20:37 images
-rw-r--r-- 1 pi pi  15823 Mär  3  2017 m8.def
-rw-r--r-- 1 pi pi  25216 Apr 21 19:34 simul.c
-rw-r--r-- 1 pi pi   2986 Mär  4  2017 table.cpp
drwxr-xr-x 2 pi pi   4096 Aug  8 20:37 tools
-rw-r--r-- 1 pi pi   3811 Mär  4  2017 tree.cpp
-rw-r--r-- 1 pi pi  35640 Apr 27 10:41 ui.cxx
-rw-r--r-- 1 pi pi   8528 Sep  6  2017 ui.fl
-rw-r--r-- 1 pi pi   6607 Apr 27 10:41 ui.h
pi@raspi-11:~/avrsim/src $
```

Falls Sie die Archive mit einer anderen Archivverwaltung geöffnet haben, könnte es sein, dass das Ausführungsrecht (x) hier fehlt. In diesem Falle, kann es mit "chmod u+x c" aber auch nachträglich gesetzt werden. Dieses Script wird mit "./c" gestartet:



Die Compilierung dauert etwa eine Minute, im Erfolgsfalle, wird das Programm auch gleich gestartet.

Später kann das Programm "avrsim" mit "~/avrsim/src/avrsim" aufgerufen werden.

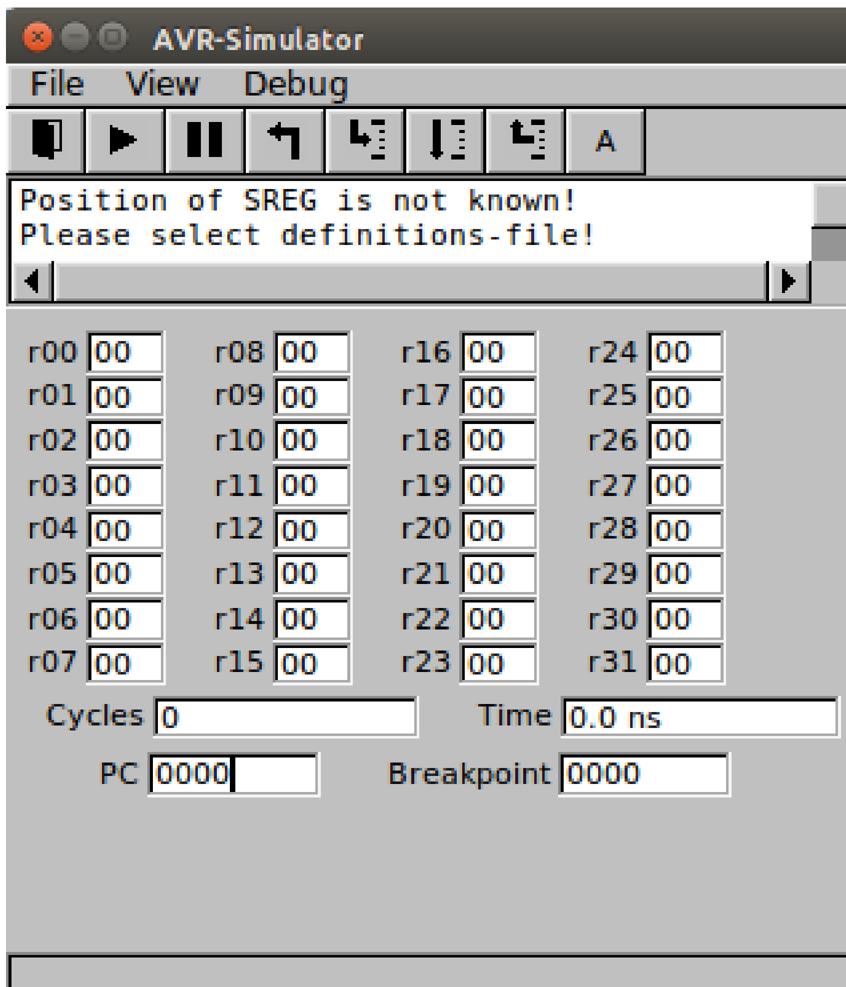
MS-Windows

Für win32, wird ein kompiliertes Paket bereitgestellt, welches die erforderlichen MinGW-Bibliotheken, das statisch gelinkte Programm, sowie erforderliches Zubehör enthält.

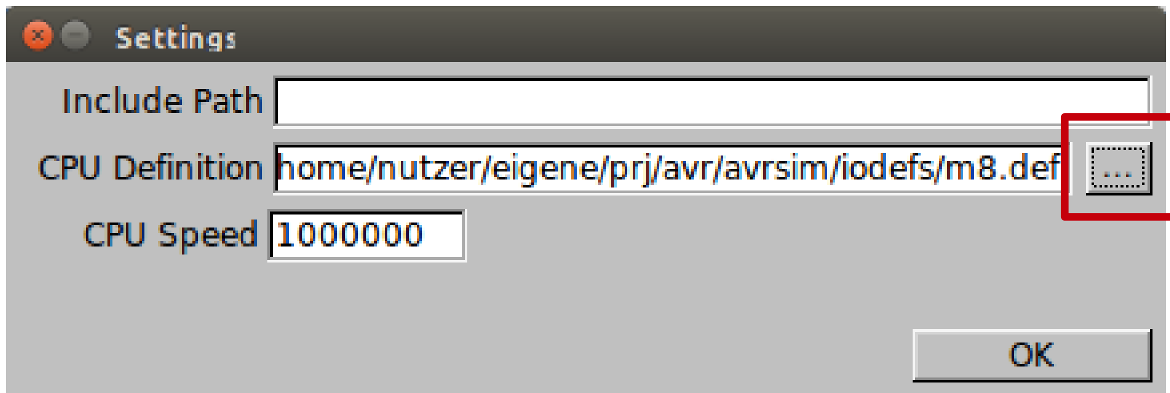
Für beide Plattformen, werden zusätzlich, die Hardwaredefinitionen (Iodefs) mit den Beschreibungen für die, zu simulierenden Bausteine benötigt, diese können an einen beliebigen Ort entpackt werden, bei Bedarf können auch die Dateien für Bausteine, welche nicht genutzt werden, gelöscht werden, um die Liste übersichtlich zu halten.

Einrichtung

Wird das Programm das erste Mal gestartet, liegen noch keine Informationen über die Art des zu simulierenden Prozessors vor. Die Baumansicht der I/O-Ports ist daher noch leer.



Diese Informationen werden aus Definitionsdateien (*.def) gelesen, die aus Standard-Includedateien des AVRASM2 abgeleitet und manuell bearbeitet wurden. Diese Dateien können aus dem Archiv "iodefs", an einen Ort nach Wahl entpackt werden. Anschließend wird die Definitionsdatei für den gewünschten Prozessortyp ausgewählt (View->Settings):



Aus diesen Dateien wird insbesondere die Lage des Statusregister, der Stackpointer, die Speicherlimits, sowie die Struktur und Benennung der I/O-Register bezogen.

Die CPU-Geschwindigkeit wird zur Berechnung der abgelaufenen Zeit im Hauptfenster benötigt.

Include Suchpfad

Je nach Gestaltung der Quellen, enthält die Objektdatei unter Umständen nicht den vollständigen Pfad zur entsprechenden Quelldatei. Falls eine Datei ohne Pfad angegeben ist, wird als Erstes versucht, sie im Verzeichnis der Objektdatei zu finden.

Sollten die Includedateien "anderswo" stehen und dieser Pfad dem Assembler per Kommandozeilenargument mitgegeben werden, kann der Simulator unter "View->Settings", Includepath darüber unterrichtet werden. Hier können bis zu 10 Verzeichnisse, durch Doppelpunkt voneinander getrennt, angegeben werden.

Pfade zu Dateien, die nur Definitionen von Konstanten, aber selbst keinen ausführbaren Code enthalten, werden aber eigentlich nicht zwingend benötigt.

Benutzung

Die Benutzeroberfläche besteht aus dem Hauptfenster, sowie einzeln ein/ausblendbaren Fenstern für Disassemblerausgabe, Quelltext, RAM, Flash, In/Output und einem Terminalfenster. Letzteres hat noch experimentellen Charakter und zeigt die Zeichen an, die in das SBUF-Register geschrieben werden.

Die zu simulierende Programmdatei wird mit "File->Load Code File" in den virtuellen Flash geladen.

Die Oberfläche für die Benutzereingaben, orientiert sich für die bisher verfügbaren Funktionen am AVR-Studio 4.

Diassembler

```
000B: D001          rcall 000D
000C: CFFA          rjmp 0007
000D: 931F          push r17
000E: 932F          push r18
000F: 933F          push r19
0010: E31B          ldi r17,0x3b
0011: E02D          ldi r18,0x0d
0012: E033          ldi r19,0x03
0013: 5011          subi r17,0x01
0014: 4020          sbci r18,0x00
0015: 4030          sbci r19,0x00
0016: F7E0          brcc 0x0013
0017: 913F          pop r19
0018: 912F          pop r18
0019: 911F          pop r17
001A: 9508          ret
001B: FFFF          ill opc 0xFFFF
001C: FFFF          ill opc 0xFFFF
001D: FFFF          ill opc 0xFFFF
001E: FFFF          ill opc 0xFFFF
001F: FFFF          ill opc 0xFFFF
0020: FFFF          ill opc 0xFFFF
0021: FFFF          ill opc 0xFFFF
0022: FFFF          ill opc 0xFFFF
0023: FFFF          ill opc 0xFFFF
0024: FFFF          ill opc 0xFFFF
0025: FFFF          ill opc 0xFFFF
0026: FFFF          ill opc 0xFFFF
0027: FFFF          ill opc 0xFFFF
0028: FFFF          ill opc 0xFFFF
0029: FFFF          ill opc 0xFFFF
002A: FFFF          ill opc 0xFFFF
002B: FFFF          ill opc 0xFFFF
002C: FFFF          ill opc 0xFFFF
002D: FFFF          ill opc 0xFFFF
002E: FFFF          ill opc 0xFFFF
002F: FFFF          ill opc 0xFFFF
0030: FFFF          ill opc 0xFFFF
0031: FFFF          ill opc 0xFFFF
0032: FFFF          ill opc 0xFFFF
0033: FFFF          ill opc 0xFFFF
0034: FFFF          ill opc 0xFFFF
0035: FFFF          ill opc 0xFFFF
0036: FFFF          ill opc 0xFFFF
0037: FFFF          ill opc 0xFFFF
0038: FFFF          ill opc 0xFFFF
0039: FFFF          ill opc 0xFFFF
003A: FFFF          ill opc 0xFFFF
003B: FFFF          ill opc 0xFFFF
003C: FFFF          ill opc 0xFFFF
003D: FFFF          ill opc 0xFFFF
003E: FFFF          ill opc 0xFFFF
003F: FFFF          ill opc 0xFFFF
0040: FFFF          ill opc 0xFFFF
0041: FFFF          ill opc 0xFFFF
0042: FFFF          ill opc 0xFFFF
0043: FFFF          ill opc 0xFFFF
0044: FFFF          ill opc 0xFFFF
0045: FFFF          ill opc 0xFFFF
0046: FFFF          ill opc 0xFFFF
```

test.asm

```
        .include "Appnotes/m8def.inc"

        ldi     r16,low(ramend)
        out     spl,r16
        ldi     r16,high(ramend)
        out     sph,r16

        ldi     r16,0xff
        out     ddrb,r16
        ldi     r16,1
loop:
        lsr     r16
        brcc    no_carry
        sbr     r16,0x80
no_carry:
        out     portb,r16
        rcall   delay
        rjmp    loop

        .include "zweite.asm"
```

RAM

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0010	80	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0030	00	00	00	00	00	00	00	FF	80	00	00	00	00	00	00	00
0040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0050	00	00	00	00	00	00	00	00	00	00	00	00	00	5F	04	15
0060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
01A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

test.obj (ATmega8) - AVR Simulator

File View Debug

A

```
I=0 T=0 H=0 S=1 V=0 N=1 Z=0 C=1
SP=045f X=0000 Y=0000 Z=0000
```

r00	00	r08	00	r16	80	r24	00
r01	00	r09	00	r17	00	r25	00
r02	00	r10	00	r18	00	r26	00
r03	00	r11	00	r19	00	r27	00
r04	00	r12	00	r20	00	r28	00
r05	00	r13	00	r21	00	r29	00
r06	00	r14	00	r22	00	r30	00
r07	00	r15	00	r23	00	r31	00

Cycles 11 Time 11.00 µs

PC 000B Breakpoint 0000

Input Output

```
ROOT
  ANALOG_COMPARATOR
  SPI
  EXTERNAL_INTERRUPT
  TIMER_COUNTER_0
  TIMER_COUNTER_1
  TIMER_COUNTER_2
  USART
  TWI
  WATCHDOG
  PORTB
    38 PORTB       = 80 ;Port B Data Register
    37 DDRB        = ff ;Port B Data Direction Reg
    36 PINB        = 00 ;Port B Input Pins
  PORTC
  PORTD
  EEPROM
  CPU
    5f SREG        = 15 ;Status Register
    55 MCUCR       = 00 ;MCU Control Register
    54 MCUCSR      = 00 ;MCU Control And Status Re
    51 OSCCAL      = 00 ;Oscillator Calibration Va
    57 SPMCR       = 00 ;Store Program Memory Cont
    50 SFIOR       = 00 ;Special Function IO Regis
    5d SPL         = 5f ;SPL
    5e SPH         = 04 ;SPH
  AD_CONVERTER
```

Hauptfenster

Im Hauptfenster befinden sich Eingabefelder für die Register r0 bis r31. Die Registerinhalte können manuell geändert werden. Der geänderte Wert wird wirksam, wenn das Eingabefeld den Fokus verliert (Tabulator). Über das Hauptfenster wird der Simulator gesteuert (Menüs, Hotkeys oder Schaltflächen), sowie werden die restlichen Fenster sichtbar/unsichtbar gemacht (View-Menü).

Run

Das Programm läuft im Hintergrund, ohne Aktualisierung der Anzeigen, mit voller Geschwindigkeit, bis zum gegebenenfalls gesetzten Haltepunkt.

Break

Damit kann das Programm abgebrochen und inspiziert werden.

Reset

Ein gegebenenfalls laufendes Programm wird abgebrochen, der Befehlszähler und die Stoppuhr zurückgesetzt und das Programm neu in den Flash geladen (für den Fall, dass es inzwischen geändert worden sein sollte).

Step Into

Es wird ein Prozessorbefehl ausgeführt. Sollte es sich dabei um einen Call handeln, ist der nächste Befehl, der erste der Subroutine.

Step Over

Sollte es sich hier um einen Call handeln, wird die gesamte Subroutine ausgeführt. Falls das Programm in der Subroutine "hängen" sollte, kann die mit Break unterbrochen und inspiziert werden.

Step Out

Hier werden alle Befehle ausgeführt, bis die Routine mit RET beendet wird ausgeführt.

Auto Step

Das Programm läuft, bis es mit Break unterbrochen wird. Dabei wird der Prozessorstatus auf dem Bildschirm angezeigt, was ein Vielfaches der Zeit benötigt ;-)

Die Stoppuhr

Im Eingabefeld "Cycles" wird ein Zähler der ausgeführten Taktzyklen geführt. Dieser kann bei Bedarf manuell "genullt", oder wie gewünscht geändert werden. Von Diesem abhängig, ist die Anzeige "Time". Diese setzt voraus, das unter "View->Settings" die richtige Taktfrequenz (CPU Speed, in Hz) eingestellt ist.

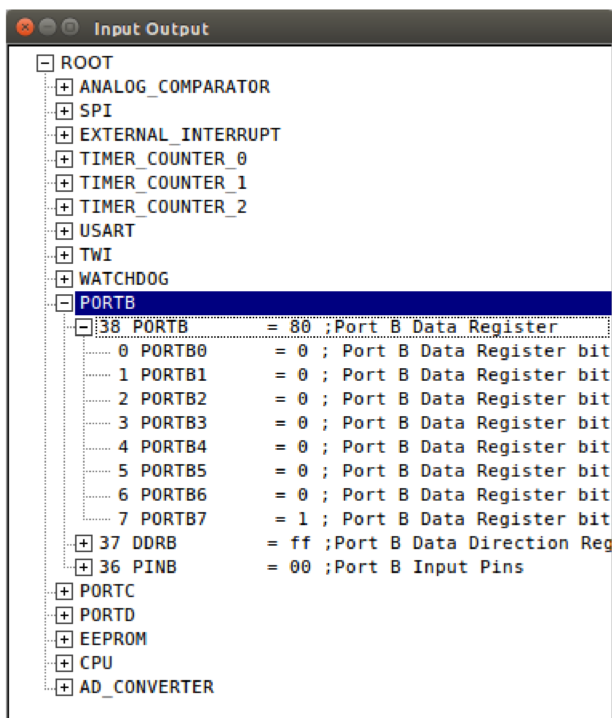
Breakpoint

Bisher ist ein Haltepunkt einstellbar (wird sich hoffentlich bald ändern). Ein eingestellter Wert von 0 kommt nur zum Tragen, wenn der PC überläuft, oder ein Sprungbefehl auf diese Adresse stattfindet.

RAM und Flash

Diese Speicherbereiche werden als Tabelle dargestellt. Eine Zelle kann durch Anklicken geändert werden, dann erscheint ein Eingabedialog (derzeit bitte als Hexadezimal).

Ein/Ausgabe

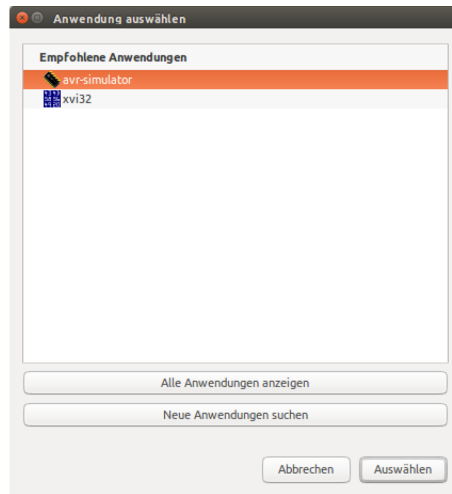


Die Ports und Bits sind zu Funktionsgruppen, in einer Baumansicht integriert. Wird ein Port angeklickt, erscheint ein Dialog, zur Eingabe eines neuen (Hex)-Wertes.

Wird ein Bit angeklickt (erhält es den Fokus), wird es getoggelt. Soll es zurücktoggeln, muss der Fokus erstmal woanders hin (die Gruppe anklicken, das bewirkt nichts), bevor das Bit wieder angeklickt werden kann. Vielleicht fällt mir dafür noch was besseres ein..

Starter Erzeugen

Benutzer von Gnome (und kompatiblen) Desktops, können mit "File->Create Starter" eine Desktopdatei in ihrem persönlichem Applications-Verzeichnis erzeugen. Damit lässt sich dann die Erweiterung ".obj" mit dem Programm verknüpfen, und die Dateien lassen sich dann mit Doppelklick öffnen.



Definitionsdateien

Der Simulator, sowie die Benutzeroberfläche, wird durch Dateien gesteuert, welche den zu simulierenden Prozessor beschreiben. Diese wurden automatisch aus den Assembler-Includedateien abgeleitet, mussten aber manuell nachbearbeitet werden. Hier ein Beispiel für die, für den EEPROM zuständigen Register in der Rohdatei:

```
; ***** EEPROM ***********************
; EEDR - EEPROM Data Register
.equ EEDR0      = 0   ; EEPROM Data Register bit 0
.equ EEDR1      = 1   ; EEPROM Data Register bit 1
.equ EEDR2      = 2   ; EEPROM Data Register bit 2
.equ EEDR3      = 3   ; EEPROM Data Register bit 3
.equ EEDR4      = 4   ; EEPROM Data Register bit 4
.equ EEDR5      = 5   ; EEPROM Data Register bit 5
.equ EEDR6      = 6   ; EEPROM Data Register bit 6
.equ EEDR7      = 7   ; EEPROM Data Register bit 7

; EECR - EEPROM Control Register
.equ EERE       = 0   ; EEPROM Read Enable
.equ EEWE       = 1   ; EEPROM Write Enable
.equ EEMWE      = 2   ; EEPROM Master Write Enable
.equ EEWEE      = EEMWE    ; For compatibility
.equ EERIE      = 3   ; EEPROM Ready Interrupt Enable
```

Man sieht hier eine Abschnittsüberschrift (mit den Sternen), die beiden Überschriften mit den Registernamen (die Adressen wurden weiter oben definiert), sowie die Definitionen der Bits.

Was hier fehlt, sind die beiden Register EEARH und EEARL, die oben zwar definiert, hier aber nicht referenziert werden.

In der aufbereiteten Definitionsdatei sieht das dann so aus:

```
Group EEPROM
 Port EEDR=3D ;EEPROM Data Register
  Bit EEDR0=0 ; EEPROM Data Register bit 0
```

```
  Bit EEDR1=1 ; EEPROM Data Register bit 1
  Bit EEDR2=2 ; EEPROM Data Register bit 2
  Bit EEDR3=3 ; EEPROM Data Register bit 3
  Bit EEDR4=4 ; EEPROM Data Register bit 4
  Bit EEDR5=5 ; EEPROM Data Register bit 5
  Bit EEDR6=6 ; EEPROM Data Register bit 6
  Bit EEDR7=7 ; EEPROM Data Register bit 7
 Port EECR=3C ;EEPROM Control Register
  Bit EERE=0 ; EEPROM Read Enable
  Bit EEWE=1 ; EEPROM Write Enable
  Bit EEMWE=2 ; EEPROM Master Write Enable
  Bit EERIE=3 ; EEPROM Ready Interrupt Enable
```

Die beiden Zeilen

```
 Port EEARL=3E
 Port EEARH=3F
```

konnten nicht zugeordnet werden und erscheinen in der Gruppe "verwaiste Ports". Diese verwaisten Ports habe ich dann manuell in die (hoffentlich) richtigen Gruppen verschoben.

In der Mehrheit der Bausteine betraf dies den Stackpointer (SPL und SPH), die in die Gruppe CPU gehören, wie genannt, die EEPROM-Adressen, UART-Baudratenregister und Register von Timern.

An der Reihenfolge der Gruppen hab ich nichts geändert, die sind noch in der, von ATMEL vorgegebenen Reihenfolge. Wer möchte, kann die gern ändern. Ebenso können meiner Meinung nach, auch etliche Bitdefinitionen entfernt werden (im EEDR, zum Beispiel, wird man eher selten einzelne Bits ansprechen).

Dies habe ich für die Bausteine erledigt, die ich für gebräuchlich und relevant halte. Für die Dateien im Unterordner "low_prio" ist dies noch nicht geschehen. Dies hole ich bei Bedarf nach, teilweise müsste ich mir dafür auch erst Datenblätter runterladen, da ich die Bausteine und ihre Architektur nicht kenne.

Einstellungen

Das Programm speichert seine Einstellungen unter $HOME/.config/avrsim.conf (GNU/Linux) beziehungsweise %userprofile%\avrsim.ini (MS-Windows).

Sollte das Programm nicht mehr korrekt starten, oder sich sonst unerwartet verhalten, können diese Dateien gelöscht werden, das Programm verhält sich dann "wie neu".

Aussicht

Als Nächstes, möchte ich mich mit dem ELF-Dateiformat beschäftigen, welches vom AVR-GCC (LD) ausgegeben wird. Ansonsten werden erste Periperiefunktionen integriert (EEPROM)...

Chronik

12.03.2017	Elf-Dateien können jetzt gelesen werden, Debuginformationen können teilweise verarbeitet werdem.

04.03.2017	Ansicht der Peripherie überarbeitet (Baumansicht), Arbeitsregister haben jetzt Eingabefelder bekommen. Definitionsdateien wurden neu erstellt und angepasst.