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.

Gelesen werden Objektdateien (.obj), die sowohl vom 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 Quelldateien.

Ebenfalls ist es auch möglich, Intel-Hexdateien (.hex) einzulesen. Da hier aber keine Debug-Informationen vorliegen, kann der Programmablauf hier nur anhand der Disassemblerausgabe verfolgt werden.

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.

Die entstandene Programmdatei "avrsim" kann dann bei Bedarf, an einen geeigneteren Ort verschoben werden.

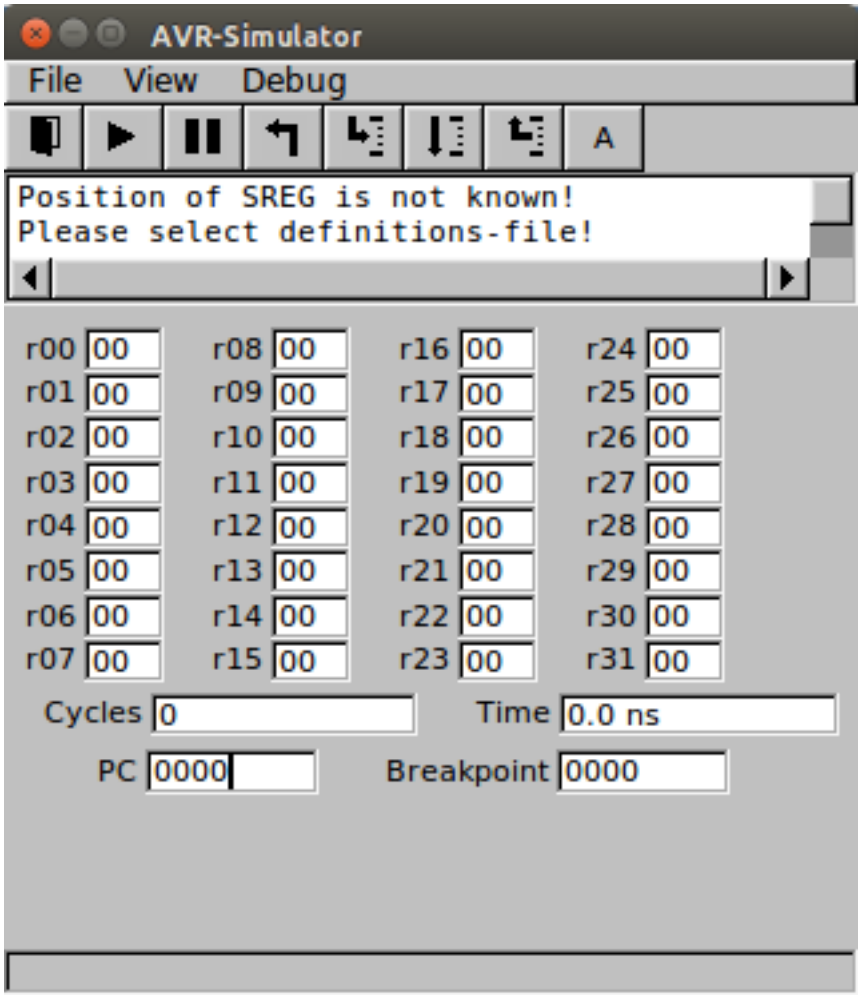
MS-Windows

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

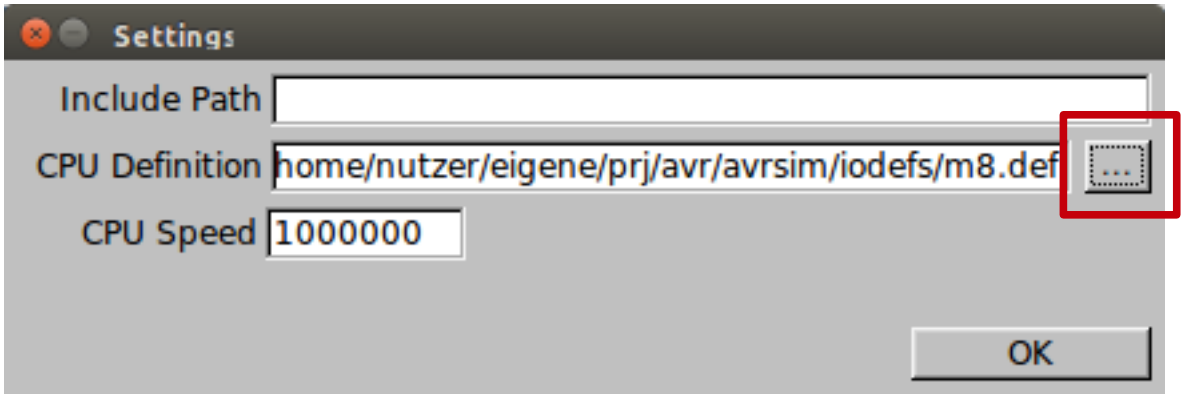
Für beide Plattformen, werden zusätzlich, die Hardwaredefinitionen (Iodefs) benötigt.

Einrichtung

Wird das Programm das erste Mal gestartet, liegen noch keine Informationen über die Art des zu simulierenden Prozessors vor. Für die I/O-Ports können daher keine Namen angezeigt werden.



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.

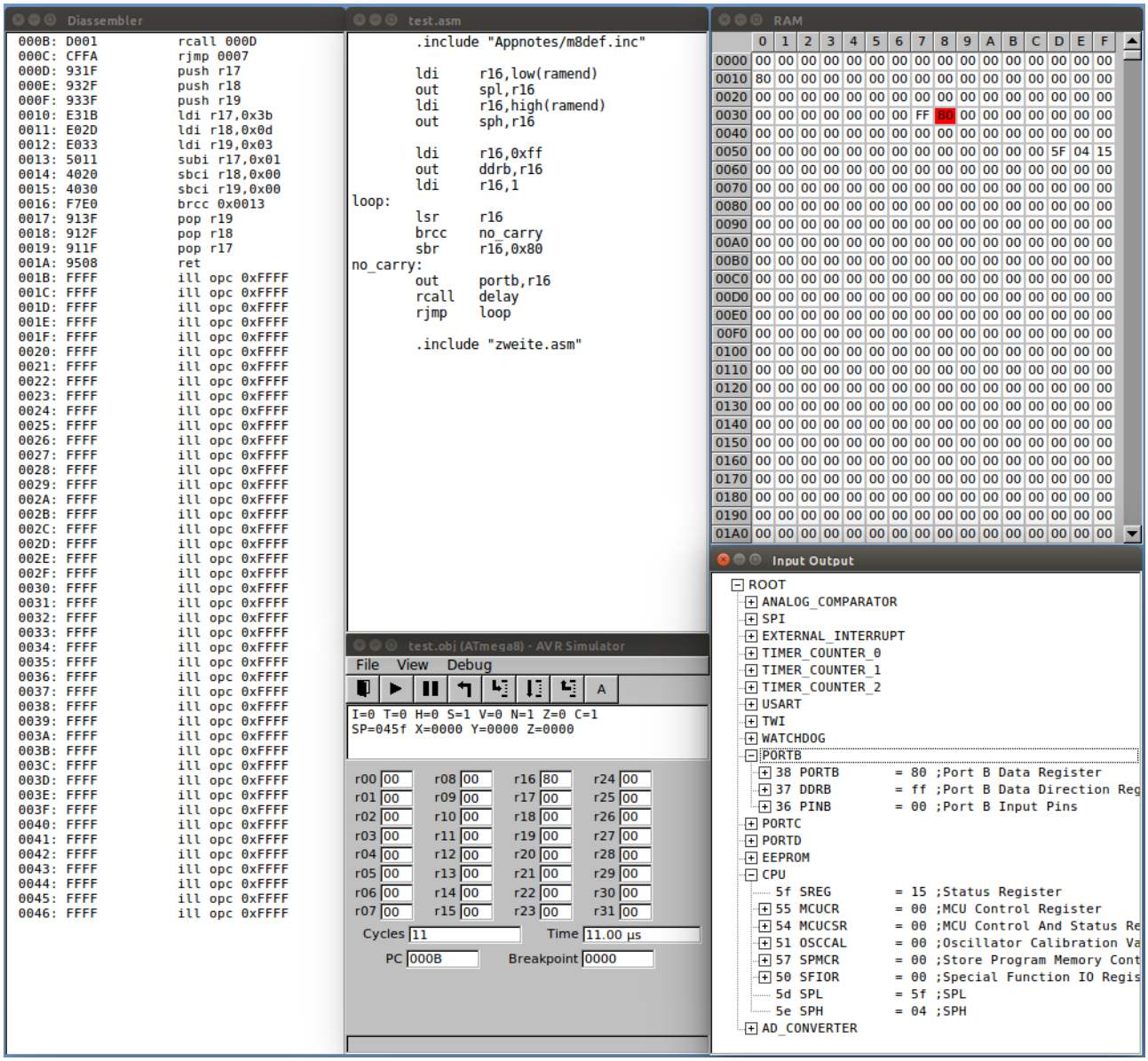
Sollten die Includedateien "anderswo" stehen und dieser Pfad dem Assembler per Kommandozeilenargument mitgegeben werden, kann der Simulator unter "View->Settings", Includepath darüber unterrichtet werden. Hier können bis zu 10 Verzeichnisse, durch Doppelpunkt voneinander getrennt, angegeben werden.

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.



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.

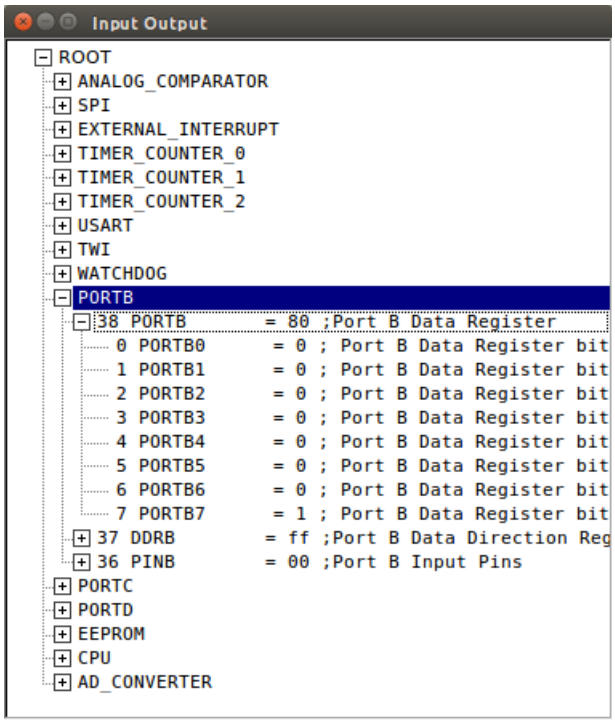
Breakpoint

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

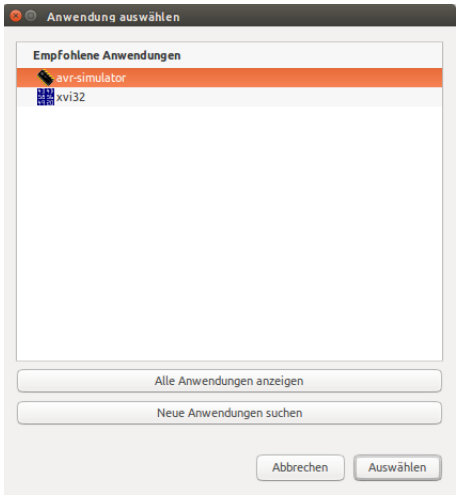


Die Ports und Bits sind zu Funktionsgruppen, in einer Baumansicht integriert. Wird ein Port angeklickt, erscheint ein Dialog, zur Eingabe eines neuen (Hex)-Wertes.

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

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