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 "<u>AS</u>" 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 <u>FLTK</u>-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.

800	AVR-S	imulato	r					
File Vi	ew	Debug						
		1	4	11	1	Α		
Positio	n of	SREG i	s n	ot k	nown	!		
Please	selec	t defi	nit	ions	-fil	e!		
◀								
r00 00 r01 00 r02 00 r03 00 r04 00 r05 00 r06 00	0 70 1 71 71 71 71	08 00 09 00 10 00 11 00 12 00 13 00 14 00		16 0 17 0 18 0 19 0 20 0 21 0 22 0	0 0 0 0 0	r24 r25 r26 r27 r28 r29 r30	00 00 00 00 00 00	
r07 00 r15 00 r23 00 r31 00 Cycles 0 Time 0.0 ns PC 0000 Breakpoint 0000								

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.

000B:	D001	rcall 000D	.include "Appnotes/m8def.inc"	0 1 2 3 4 5 6 7 8 9 A B C D E F 🔺
000C:	CFFA	rjmp 0007		
000D:	931F	push r17	ldi r16,low(ramend)	
000E:	932F	push r18	out spl,r16	
000F:	933F	push r19	ldi r16,high(ramend)	
0010:	E31D	ldi cl8 0x0d	out sph, r16	
0012:	E033	1di r19.0x03		
0013:	5011	subi r17.0x01	ldi r16,0xff	0050 00 00 00 00 00 00 00 00 00 00 00 00
0014:	4020	sbci r18,0x00	out ddrb, r16	0060 00 00 00 00 00 00 00 00 00 00 00 00
0015:	4030	sbci r19,0x00	ld1 r16,1	0070 00 00 00 00 00 00 00 00 00 00 00 00
0016:	F7E0	brcc 0x0013	Loop:	0080 00 00 00 00 00 00 00 00 00 00 00 00
0017:	913F	pop r19	lsr r16	0090 00 00 00 00 00 00 00 00 00 00 00 00
0018:	912F	pop r18	brcc no_carry	00 00 00 00 00 00 00 00 00 00 00 00 00
0019:	9508	ret	SDI 110,0X00	0080 00 00 00 00 00 00 00 00 00 00 00 00
001B:	FFFF	ill opc 0xFFFF	no_carry.	
001C:	FFFF	ill opc 0xFFFF	rcall delay	
001D:	FFFF	ill opc 0xFFFF	rimp loop	
001E:	FFFF	ill opc 0xFFFF	i jiip coop	
001F:	FFFF	ill opc 0xFFFF	include "zweite.asm"	
0020:	FFFF	ill opc 0xFFFF		0100 00 00 00 00 00 00 00 00 00 00 00 00
0021:		ill opc 0xFFFF		0110 00 00 00 00 00 00 00 00 00 00 00 00
0022:	FFFF	ill opc OXFFFF		0120 00 00 00 00 00 00 00 00 00 00 00 00 0
0023.	FFFF	ill onc OxFFFF		0130 00 00 00 00 00 00 00 00 00 00 00 00 0
0025:	FFFF	ill opc 0xFFFF		0140 00 00 00 00 00 00 00 00 00 00 00 00 0
0026:	FFFF	ill opc 0xFFFF		0150 00 00 00 00 00 00 00 00 00 00 00 00 0
0027:	FFFF	ill opc 0xFFFF		
0028:	FFFF	ill opc 0xFFFF		
0029:	FFFF	ill opc 0xFFFF		
002A:		ill opc 0xFFFF		
002B:		ill opc OxFFFF		
0020	FFFF	ill onc OxFFFF		
002E:	FFFF	ill opc 0xFFFF		😣 🔿 🕥 Japut Output
002F:	FFFF	ill opc 0xFFFF		
0030:	FFFF	ill opc 0xFFFF		ROOT
0031:	FFFF	ill opc 0xFFFF		
0032:	FFFF	ill opc 0xFFFF		+ SPI
0033:		ill opc OxFFFF	AVP Simulator	EXTERNAL_INTERRUPT
0034.	FFFF	ill onc AVEEEE	File View Debug	+ TIMER_COUNTER_0
0036:	FFFF	ill opc 0xFFFF	File view Debug	TIMER_COUNTER_1
0037:	FFFF	ill opc 0xFFFF	₩ ▶ ₩ 1 12 12 12 4	TIMER_COUNTER_2
0038:	FFFF	ill opc 0xFFFF	T-0 T-0 H-0 S-1 V-0 N-1 7-0 C-1	- • USART
0039:	FFFF	ill opc 0xFFFF	SP=045f X=0000 Y=0000 7=0000	+ TWI
003A:	FFFF	ill opc 0xFFFF		+ WATCHDOG
0038:		ill opc 0xFFFF		- PORTB
0030	FFFF	ill opc 0xFFFF	r00 00 r08 00 r16 80 r24 00	+ 38 PORTB = 80 ;Port B Data Register
003E:	FFFF	ill opc 0xFFFF	r01 00 r09 00 r17 00 r25 00	+ 37 DDRB = ff ;Port B Data Direction Reg
003F:	FFFF	ill opc 0xFFFF		+ 36 PINB = 00 ;Port B Input Pins
0040:	FFFF	ill opc 0xFFFF		+ PORTC
0041:	FFFF	ill opc 0xFFFF	r03 00 r11 00 r19 00 r27 00	+ PORTD
0042:	FFFF	ill opc 0xFFFF	r04 00 r12 00 r20 00 r28 00	+ EEPROM
0043:	FFFF	ill opc 0xFFFF	r05 00 r13 00 r21 00 r29 00	- E CPU
0044:	FFFF	ill opc OXFFFF	r06 00 r14 00 r22 00 r30 00	5f SREG = 15 ;Status Register
0045	FFFF	ill opc 0xFFFF	r07 00 r15 00 r23 00 r31 00	⊕ 55 MCUCR = 00 ;MCU Control Register
		and ope existin	Cycles 11	⊕ 54 MCUCSR = 00 ;MCU Control And Status Re
				+ 51 OSCCAL = 00 ;Oscillator Calibration Va
			PC 000B Breakpoint 0000	+ 57 SPMCR = 00 ;Store Program Memory Cont
				+ 50 SFIOR = 00 ;Special Function IO Regis
				5e SPH = 04 ;SPH
1				+ AD_CONVERTER
1				-

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

😕 🗇 🔍 Input Output	
- ROOT	
+ ANALOG COMPARATOR	
+ SPI	
EXTERNAL_INTERRUPT	
TIMER_COUNTER_0	
TIMER_COUNTER_1	
TIMER_COUNTER_2	
··· + USART	
I WI	
+ WATCHDOG	
- 38 PORTB = 80 ; Port I	B Data Register
0 PORTB0 = 0 ; Port	B Data Register bit
1 PORTB1 = 0 ; Port	B Data Register bit
2 PORTB2 = 0 ; Port	B Data Register bit
3 PORTB3 = 0; Port	B Data Register bit
4 PORTB4 = 0 ; Port	B Data Register bit
C DODTRG = 0 ; POFT	B Data Register bit
7 POPTP7 = 1 : Port	B Data Register bit
- F PORID/ = 1; PORI	B Data Register Dit
	B Jaca Direction Reg
	b input Fins

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.

Emproniene Anv	rendungen		
💊 avr-simulato	•		
xvi32			
	Alle Anwend	ungen anzeigen	
	Neue Anwe	ndungen suchen	

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	*	* * * :	* *	******	* * * * *	* * * * *		
; EEI	DR - EEPRO	M	Data	a .	Registe	r			
.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
; EE(CR - EEPRO	M	Cont	cr	ol Regi	ster			
.equ	EERE	=	0	;	EEPROM	Read	Enable		
.equ	EEWE	=	1	;	EEPROM	Write	e Enable		
.equ	EEMWE	=	2	;	EEPROM	Maste	er Write	Enabi	le
.equ	EEWEE	=	EEN	1WI	E ; Fo	or con	npatibili	ty	
.equ	EERIE	=	3	;	EEPROM	Ready	/ Interru	pt Ei	nabl

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.