

## **Einleitung**

Mit dem Erscheinen der Entwicklungsumgebung STUDIO5 von Atmel und mit den XMEGA-Prozessoren hat das Unternehmen mehr Gewicht auf die proprietäre Funktionalität gelegt. So wird die firmenspezifische Debugging-Schnittstelle PDI verstärkt unterstützt und die bisherige weitgehend auf Open-Source-Software basierte Entwicklungsumgebung STUDIO4 eingestellt.

Die bis dato genutzte und auch weiterhin angebotene JTAG-Schnittstelle wurde undokumentiert geändert.

Dies betrifft insbesondere die Kommunikation mit den Debugger- / Programmier-Tools, wie z.B. der DRAGON. Mehrfach ist auch die Firmware der Debugger- / Programmier-Tools verändert worden.

Aufgrund der unveröffentlichten Änderungen ist damit den freien Entwicklerwerkzeugen wie z.B. AVARICE und AVRDUDE die funktionelle Basis entzogen.

STUDIO5 ist Windows-basiert und stützt sich funktionell auf die .NET-Entwicklungsumgebung von Microsoft ab.

Atmel hat bis dato keine aktualisierte Dokumentation veröffentlicht, um den Entwicklern der freien Werkzeuge eine Anpassung ihrer Software zu ermöglichen.

## **Problem**

Für Entwickler, die Software für AVR-Prozessoren bisher in einer Linux-Umgebung mittels den obengenannten freien Werkzeugen erstellt haben, ist damit die direkte Arbeitsgrundlage entzogen.

Das Übertragen von AVR-Applikationen auf den Mikroprozessor per AVRDUDE funktioniert derzeit noch, aber das Debuggen via GDB und AVARICE funktioniert nicht mehr.

## **Projekt**

Es soll für die Anpassung der freien Werkzeuge die notwendige Information gesammelt werden.

## **Ergebnis**

Erstellen einer Dokumentation über die einzelnen Schnittstellenabläufe beim Übertragen und beim Debuggen von Applikationen auf den AVR-Mikroprozessoren, im Besonderen derer vom Typ XMEGA.

Das Dokument fasst die wesentlichen Erkenntnisse aus den Debug-Abläufen zusammen, um damit eine Diskussion der weiteren Entwicklungsschritte starten zu können. Soweit als möglich werden die einzelnen Abläufe und Schritte identifiziert und kommentiert. Im Schwerpunkt sollen die Debugging Funktionen geklärt sein, da hier zur Zeit keinerlei Funktionalität bei AVARICE für XMEGA Prozessoren mehr vorhanden ist.

### Test-Umgebung

STUDIO5 wurde auf einer Windows-XP SP3 Basis installiert. Als Programmier- / Debugging-Tool wird ein AVR-DRAGON Version A09 eingesetzt. Die Firmware des DRAGON ist auf dem letztaktuellen Stand Version 7.0.

Als Ziel-Prozessor steht ein ATXMEGA128A1 auf einem Atmel-Xplained Board zur Verfügung. Die Kommunikation zum Programmier- / Debugging Tool erfolgt über eine USB2 Schnittstelle, die mittels eines Usb-Lib Filtertreibers modifiziert wurde.

USB-TRACE Version 2.6 von SysNucleus wird als USB-Monitor verwandt.

Ein einfaches C-Programm aus wenigen Befehlen dient als Programmier- und Debugging-Objekt (Anlage).

### Vorgehensweise

Ausgehend von den im STUDIO zur Nutzung der Programmier- und Debugging-Tools angebotenen Funktionen ist der Ablauf der Untersuchung aufgebaut. Soweit sinnvoll und technisch möglich sind möglichst kleine Tracepakete gebildet worden:

1. Tool-Device und Schnittstellenauswahl
  - a. SIGN ON Phase
  - b. Lesen der Device-Informationen
  - c. Lesen der Fuses
  - d. Lesen der Lock-Bits
  - e. Beenden der Kommunikation
2. Start Debugging mit anschließendem impliziten Break
  - a. Impliziter Halt bei der Funktion main()
  - b. Weiter mit F11 = Einzelschritt
  - c. Weiter mit F5 bis zu einem expliziten Haltepunkt
3. Start Debugging ohne impliziten Halt, mit vorher gesetztem Haltepunkt
  - a. Setzen eines Haltepunktes, Start Debugging
  - b. Setzen eines weiteren Haltepunktes, weiter mit F5
  - c. Rücksetzen des Haltepunktes, weiter mit F11
  - d. Setzen mehrere Haltepunkte, weiter mit F11
  - e. Rücksetzen der Haltepunkte, weiter mit F11
4. Übertragung des Programmcodes
5. Standardabläufe
  - a. Register lesen
  - b. Speicherbereiche lesen
  - c. Speicherbereiche schreiben / ändern
6. Nicht geklärt
  - a. Neue, undokumentierte Kommandos und Parameter

Hinweis: Basis der Dokumentation ist:

AVR-Note 067 JTAGICE mkII Communication Protocol Rev.2587D-AVR-11/09

AVR-XMEGA Manual

Source-Listing: IIR.c die zugehörige .LSS –Liste und die Disassembler-Liste (Anhang)

*Die in dem Dokument wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. sind Eigentum der jeweiligen Besitzer und durch gesetzliche Bestimmungen geschützt.*

## Zu 1 Tool- Device und Schnittstellenauswahl

a: SIGN ON Phase

### Menü-Auswahl

**STUDIO -> TOOLS -> AVR-PROGRAMMING**

**Tool: AVR Dragon**

**Device: ATxmega128A1**

**Interface: JTAG**

OUT 1B 00 00 01 00 00 00 0E  
01 **GET\_SIGN\_ON**  
F3 97 **CRC**

IN 1B 00 00 1A 00 00 00 0E  
86 **RSP\_SIGNON**  
01 **Communication Protocol Version**  
FF ??  
0E 07 **Master Firmware Version 7.14**  
01 **Hardware Version**  
FF ??  
0E 07 **Slave Firmware Version 7.14**  
07 **Hardware Version**  
00 A2 00 01 97 2C **Serial number 00A20001972C**  
41 56 52 44 52 41 47 4F 4E 00 **DRAGON**  
B4 9B **CRC**

*Wiederholung der SIGN\_ON Sequenz*

OUT 1B 01 00 01 00 00 00 0E  
01  
4C 16

IN 1B 01 00 1A 00 00 00 0E  
86 01 FF 0E 07 01 FF 0E 07 07 00 A2 00 01 97 2C 41 56 52 44 52 41 47 4F 4E 00 D7 DB

### Ergebnis in STUDIO

AVR Dragon  
Debug host 127.0.0.1  
Debug port 4625  
Serial number 00A20001972C  
Connection com.atmel.avrdbg.connection.jungousb  
Master Firmware Version 7.14  
Slave Firmware Version 7.14  
Hardware Version 17

## Zu 1 Tool- Device und Schnittstellenauswahl

b: Lesen der Device-Informationen

### Device-Informationen im Menü ausgewählt

OUT 1B 02 00 03 00 00 00 0E  
02 **SET\_PARAMETER**  
13 00 **External Reset: NO**  
EA B5 **CRC**  
IN 1B 02 00 01 00 00 00 0E 80 1D 09

OUT 1B 03 00 03 00 00 00 0E  
02 **SET\_PARAMETER**  
03 02 **MODE: JTAG, default**  
4E 2F **CRC**  
IN 1B 03 00 01 00 00 00 0E 80 A2 88

OUT 1B 04 00 03 00 00 00 0E  
02 **SET\_PARAMETER**  
03 05 **MODE: JTAG, XMEGA**  
04 9F **CRC**  
IN 1B 04 00 01 00 00 00 0E 80 AC 14

OUT 1B 05 00 33 00 00 00 0E  
**36 Neues Kommando für PDI - Setup**  
02 00 2F **???????**  
00 00 80 00 **APP\_SECTION\_OFFSET**  
00 00 82 00 **BOOT\_SECTION\_OFFSET**  
00 00 8C 00 **EEPROM\_OFFSET**  
20 00 8F 00 **FUSE\_REGISTERS\_OFFSET**  
27 00 8F 00 **LOCK\_REGISTERS\_OFFSET**  
00 04 8E 00 **USER\_SIGNATURES\_OFFSET**  
00 02 8E 00 **PROD\_SIGNATURES\_OFFSET**  
00 00 00 01 **DATAMEM\_OFFSET**  
00 00 02 00 **BOOT-Section**  
00 20 00 02 00 08 20  
C0 01 **NVM-Base-Address**  
90 00 **MCU-Base-Address**  
DC B4 **CRC**  
IN 1B 05 00 01 00 00 00 0E 80 13 95

OUT 1B 06 00 06 00 00 00 0E  
02 **SET\_PARAMETER**  
1B 00 00 00 00 **Daisy Chain Info**  
5D 79 **CRC**  
IN 1B 06 00 01 00 00 00 0E 80 C3 1F

OUT 1B 07 00 03 00 00 00 0E  
 02 **SET\_PARAMETER**  
 38 01 **RUN after Programming Allow Target to run**  
 43 FF **CRC**

IN 1B 07 00 01 00 00 00 0E 80 7C 9E

OUT 1B 08 00 01 00 00 00 0E  
 14 **ENTER\_PROGRAMMING\_MODE**  
 63 FD **CRC**

IN 1B 08 00 01 00 00 00 0E 80 CE 2F

OUT 1B 09 00 02 00 00 00 0E  
 03 **GET\_PARAMETER**  
 0E **JTAGID**  
 14 85 **CRC**

IN 1B 09 00 05 00 00 00 0E  
 81 **RSP\_PARAMETER**  
 3F C0 74 79 **JTAG ID: 7974C03F**  
 99 6F

OUT 1B 0A 00 0B 00 00 00 0E  
 05 **READ\_MEMORY**  
 B4 **SIGN\_JTAG**  
 03 00 00 00 **BYTE\_CNT**  
 00 **Register-Index**  
 00 00 00 00 **START\_ADRESS**  
 18 8D **CRC**

IN 1B 0A 00 04 00 00 00 0E  
 82 **RSP\_MEMORY**  
 1E 97 4C **Device Signature: 1E974C**  
 85 13 **CRC**

OUT 1B 0B 00 01 00 00 00 0E  
 15 **LEAVE\_PROGMODE**  
 3A 66 **CRC**

IN 1B 0B 00 01 00 00 00 0E 80 1E A5

OUT 1B 0C 00 03 00 00 00 0E  
 02 **SET\_PARAMETER**  
 03 02 **MODE: JTAG, default**  
 92 82 **CRC**

IN 1B 0C 00 01 00 00 00 0E 80 10 39

## Ergebnis in Studio

Device: ATxmega128A1  
 Device signature: 1E974C  
 JTAG ID: 7974C03F  
 Revision: H

## Zu 1 Tool- Device und Schnittstellenauswahl

c: Lesen der Fuses

### FUSES-Untermenü ausgewählt

#### FUSES

```
1B 0D 00 03 00 00 00 0E
02 SET_PARAMETER
13 00 External Reset: NO
OUT 36 18 CRC
IN 1B 0D 00 01 00 00 00 0E 80 AF B8
```

```
1B 0E 00 03 00 00 00 0E
02 SET_PARAMETER
03 02 MODE: JTAG, default
OUT DC DA CRC
IN 1B 0E 00 01 00 00 00 0E 80 7F 32
```

```
1B 0F 00 03 00 00 00 0E
02 SET_PARAMETER
03 05 MODE: JTAG; XMEGA
OUT 44 82 CRC
IN 1B 0F 00 01 00 00 00 0E 80 C0 B3
```

```
1B 10 00 33 00 00 00 0E
36 Neues Kommando für PDI - Setup
02 00 2F ???????
00 00 80 00 APP_SECTION_OFFSET
00 00 82 00 BOOT_SECTION_OFFSET
00 00 8C 00 EEPROM_OFFSET
20 00 8F 00 FUSE_REGISTERS_OFFSET
27 00 8F 00 LOCK_REGISTERS_OFFSET
00 04 8E 00 USER_SIGNATURES_OFFSET
00 02 8E 00 PROD_SIGNATURES_OFFSET
00 00 00 01 DATAMEM_OFFSET
00 00 02 00 BOOT-Section
00 20 00 02 00 08 20 ???????
C0 01 NVM-Base-Adresse
90 00 MCU-Base-Adresse
OUT D2 A0 CRC
IN 1B 10 00 01 00 00 00 0E 80 0A 59
```

```
1B 11 00 06 00 00 00 0E
02 SET_PARAMETER
1B 00 00 00 00 Daisy Chain Info
OUT CE 3B CRC
IN 1B 11 00 01 00 00 00 0E 80 B5 D8
```

1B 12 00 03 00 00 00 0E  
02 **SET\_PARAMETER**  
38 01 *Run after Programming: Allow Target to run*  
OUT AA B1 **CRC**  
IN 1B 12 00 01 00 00 00 0E 80 65 52

1B 13 00 01 00 00 00 0E  
14 **ENTER\_PROGRAMMING\_MODE**  
OUT 77 01 **CRC**  
IN 1B 13 00 01 00 00 00 0E 80 DA D3

1B 14 00 02 00 00 00 0E  
03 **GET\_PARAMETER**  
0E **JTAGID**  
OUT 52 77 **CRC**  
1B 14 00 05 00 00 00 0E  
81 **RSP\_PARAMETER**  
3F C0 74 79 **JTAG ID: 7974C03F**  
IN 8E F0 **CRC**

1B 15 00 0B 00 00 00 0E  
05 **READ\_MEMORY**  
B4 **SIGN\_JTAG**  
03 00 00 00 **BYTE\_CNT**  
00 **Register-Index**  
00 00 00 00 **START\_ADRESS**  
OUT 4F 7E **CRC**  
1B 15 00 04 00 00 00 0E  
82 **RSP\_MEMORY**  
1E 97 4C *Device Signature: 1E974C*  
IN 8C 7E **CRC**

1B 16 00 0B 00 00 00 0E  
05 **READ\_MEMORY**  
B2 **FUSE\_BITS**  
01 00 00 00 **BYTE\_CNT**  
00 **Register-Index**  
00 00 00 00 **START\_ADRESS**  
OUT 5F 0C **CRC**  
1B 16 00 02 00 00 00 0E  
82 **RSP\_MEMORY**  
FF **FUSEBYTE0**  
IN BA 9F **CRC**

1B 17 00 0B 00 00 00 0E  
05 **READ\_MEMORY**  
B2 **FUSE\_BITS**  
01 00 00 00 **BYTE\_CNT**  
01 **Register-Index**  
00 00 00 00 **START\_ADRESS**  
OUT FC FF **CRC**

1B 17 00 02 00 00 00 0E  
82 **RSP\_MEMORY**  
00 **FUSEBYTE1**  
IN 3F DD **CRC**

1B 18 00 0B 00 00 00 0E  
05 **READ\_MEMORY**  
B2 **FUSE\_BITS**  
01 00 00 00 **BYTE\_CNT**  
02 **Register-Index**  
00 00 00 00 **START\_ADRESS**  
OUT E8 E7 **CRC**

1B 18 00 02 00 00 00 0E  
82 **RSP\_MEMORY**  
BF **FUSEBYTE2**  
IN 6B 06 **CRC**

1B 19 00 0B 00 00 00 0E  
05 **READ\_MEMORY**  
B2 **FUSE\_BITS**  
01 00 00 00 **BYTE\_CNT**  
04 **Register-Index**  
00 00 00 00 **START\_ADRESS**  
OUT 97 24 **CRC**

1B 19 00 02 00 00 00 0E  
82 **RSP\_MEMORY**  
FE **FUSEBYTE4**  
IN 1B 18 **CRC**

1B 1A 00 0B 00 00 00 0E  
05 **READ\_MEMORY**  
B2 **FUSE\_BITS**  
01 00 00 00 **BYTE\_CNT**  
05 **Register-Index**  
00 00 00 00 **START\_ADRESS**  
OUT EB 2E **CRC**

1B 1A 00 02 00 00 00 0E  
82 **RSP\_MEMORY**  
FF **FUSEBYTE5**  
IN 95 DF **CRC**

1B 1B 00 01 00 00 00 0E  
15 **LEAVE\_PROGMODE**  
OUT 42 3D **CRC**  
IN 1B 1B 00 01 00 00 00 0E 80 66 FE

1B 1C 00 03 00 00 00 0E  
02 **SET\_PARAMETER**  
03 02 **MODE: JTAG, default**  
OUT C0 50 **CRC**  
IN 1B 1C 00 01 00 00 00 0E 80 68 62



### **Ergebnis in STUDIO:**

JTAGUSERID = 0xFF  
WDWP = 8CLK  
WDP = 8CLK  
DVSDON = [ ]  
BOOTRST = BOOTLDR  
BODPD = DISABLED  
RSTDISBL = [ ]  
SUT = 0MS  
WDLOCK = [ ]  
JTAGEN = [X]  
BODACT = DISABLED  
EESAVE = [ ]  
BODLVL = 1V6

FUSEBYTE0 = 0xFF (valid)  
FUSEBYTE1 = 0x00 (valid)  
FUSEBYTE2 = 0xBF (valid)  
FUSEBYTE4 = 0xFE (valid)  
FUSEBYTE5 = 0xFF (valid)

## Zu 1 Tool- Device und Schnittstellenauswahl

d: Lesen der Lock-Bits

### LOCK-Bits Untermenü ausgewählt

OUT 1B 1D 00 03 00 00 00 0E  
02 **SET\_PARAMETER**  
13 00 **External Reset: NO**  
64 CA **CRC**

IN 1B 1D 00 01 00 00 00 0E 80 D7 E3

OUT 1B 1E 00 03 00 00 00 0E  
02 **SET\_PARAMETER**  
03 02 **MODE: JTAG, default**  
8E 08 **CRC**

IN 1B 1E 00 01 00 00 00 0E 80 07 69

OUT 1B 1F 00 03 00 00 00 0E  
02 **SET\_PARAMETER**  
03 05 **MODE: JTAG; XMEGA**  
16 50 **CRC**

IN 1B 1F 00 01 00 00 00 0E 80 B8 E8

OUT

1B 20 00 33 00 00 00 0E  
36 **Neues Kommando für PDI - Setup**  
02 00 2F **??????**  
00 00 80 00 **APP\_SECTION\_OFFSET**  
00 00 82 00 **BOOT\_SECTION\_OFFSET**  
00 00 8C 00 **EEPROM\_OFFSET**  
20 00 8F 00 **FUSE\_REGISTERS\_OFFSET**  
27 00 8F 00 **LOCK\_REGISTERS\_OFFSET**  
00 04 8E 00 **USER\_SIGNATURES\_OFFSET**  
00 02 8E 00 **PROD\_SIGNATURES\_OFFSET**  
00 00 00 01 **DATAMEM\_OFFSET**  
00 00 02 00 **BOOT-SECTION**  
00 20 00 02 00 08 20 **??????**  
C0 01 **NVM-Base-Adresse**  
90 00 **MCU-Base-Adresse**  
33 D4 **CRC**

IN 1B 20 00 01 00 00 00 0E 80 82 B4

OUT 1B 21 00 06 00 00 00 0E  
02 **SET\_PARAMETER**  
1B 00 00 00 00 **Daisy Chain Info**  
BE D4 **CRC**

IN 1B 21 00 01 00 00 00 0E 80 3D 35

OUT 1B 22 00 03 00 00 00 0E  
02 **SET\_PARAMETER**  
38 01 *Run after Programming: Allow Target to run*  
4D CF **CRC**

IN 1B 22 00 01 00 00 00 0E 80 ED BF

OUT 1B 23 00 01 00 00 00 0E  
14 **ENTER\_PROGRMMING\_MODE**  
FF EC **CRC**

IN 1B 23 00 01 00 00 00 0E 80 52 3E

OUT 1B 24 00 02 00 00 00 0E  
03 **GET\_PARAMETER**  
0E **JTAGID**  
FF 7F **CRC**

IN 1B 24 00 05 00 00 00 0E  
81 **RSP\_PARAMETER**  
3F C0 74 79 **JTAG ID: 7974C03F**  
E6 CE **CRC**

OUT 1B 25 00 0B 00 00 00 0E  
05 **READ\_MEMORY**  
B4 **SIGN\_JTAG**  
03 00 00 00 **BYTE\_CNT**  
00 **Register-Index**  
00 00 00 00 **START\_ADRESS**  
CF 6D **CRC**

IN 1B 25 00 04 00 00 00 0E  
82 **RSP\_MEMORY**  
1E 97 4C *Device Signature: 1E974C*  
43 ED **CRC**

OUT 1B 26 00 0B 00 00 00 0E  
05 **READ\_MEMORY**  
B3 **LOCK\_BITS**  
01 00 00 00 **BYTE\_CNT**  
00 **Register-Index**  
00 00 00 00  
F8 33 **CRC**

IN 1B 26 00 02 00 00 00 0E  
82 **RSP\_MEMORY**  
FF **LOCK-Bits**  
17 97 **CRC**

OUT 1B 27 00 01 00 00 00 0E  
15 **LEAVE\_PROGMODE**  
A8 EB **CRC**

IN 1B 27 00 01 00 00 00 0E 80 8C 28

OUT 1B 28 00 03 00 00 00 0E  
02 **SET\_PARAMETER**  
03 02 **MODE: JTAG, default**  
BB 9E **CRC**

IN 1B 28 00 01 00 00 00 0E 80 3E 99

### Ergebnis in STUDIO:

BLBB = NOLOCK  
BLBA = NOLOCK  
BLBAT = NOLOCK  
LB = NOLOCK

LOCKBITS = 0xFF (valid)

### Zu 1 Tool- Device und Schnittstellenauswahl

e: Beenden der Kommunikation

### CLOSE-Button

1B 29 00 01 00 00 00 0E  
00 **SIGN\_OFF**  
OUT 89 9C **CRC**  
IN 1B 29 00 01 00 00 00 0E 80 81 18  
  
IN 08 00 00 00 00 00 00 00

## Zu 2 Start Debugging mit anschließendem impliziten Break

a: Impliziter Halt bei der Funktion main()

- ➔ Vorlaufendes SIGN ON und der Programmupload sind weggelassen!
- ➔ Die Vorlaufphasen sind separat erläutert!
- ➔ Die Nachlaufphasen sind separat erläutert!

OUT	1B 12 00 01 00 00 00 0E 07 D2 A2	Programm-Counter = PC auslesen
	1B 12 00 05 00 00 00 0E 84	
IN	01 00 00 00 DF 40	PC steht auf 0x00 00 00 01
	1B 13 00 01 00 00 00 0E	
OUT	07 6D 23	Wiederholung ??
	1B 13 00 05 00 00 00 0E	
IN	84 01 00 00 00 8A C5	
	1B 14 00 0E 00 00 00 0E 37	Neues undokumentiertes Kommando zur Haltepunktbehandlung
	00 00 00 00	Hier steht der PC-Wert für den HP 1
	00 00 00 00	Hier steht der PC-Wert für den HP 2
	00 00 00 00 00	
OUT	72 E9	
IN	1B 14 00 01 00 00 00 0E 80 D4 4F	
	1B 15 00 02 00 00 00 0E 0B	Reset Programm
OUT	01 98 0C	FLAG: Low Level
IN	1B 15 00 01 00 00 00 0E 80 6B CE	
	1B FF FF 08 00 00 00 0E E0	BREAK-EVENT
	00 00 01 00	PC = 0x00 01 00 00
	40	CAUSE = nicht dokumentiert, 0x40 nur in dieser Phase
	00 00	??????
IN	25 1B	
	<b>Ab hier ein Standardablauf: Auslesen des Programm-Kontextes</b>	
	1B 16 00 01 00 00 00 0E	
OUT	07 0C B4	Programm-Counter = PC auslesen
	1B 16 00 05 00 00 00 0E 84	Response PC
IN	00 00 01 00 CA 40	PC steht auf 0x00 01 00 00
	1B 17 00 0B 00 00 00 0E 05	READ_MEMORY
	B8	bei XMEGA neue Speicherklasse B8 für IO-Register
	01 00 00 00	Bytezähler
	1C	Register-Index = R28 -> Y-Low
OUT	00 00 00 00 AF 06	Start-Adresse, nicht relevant und CRC
	1B 17 00 02 00 00 00 0E 82	Antwort Lesen R28
IN	00 3F DD	1 Byte Daten = 0x00 und 2 Byte CRC
	1B 18 00 0B 00 00 00 0E 05	READ_MEMORY
	B8	bei XMEGA neue Speicherklasse B8 für IO-Register
	01 00 00 00	
	1D	Register-Index = R29 -> Y-High
OUT	00 00 00 00 33 08	
	1B 18 00 02 00 00 00 0E 82	
IN	00 17 4B	Antwort Lesen R29
	<b>Standardablauf: Auslesen aller General Purpose Register GPR: 0x00 – 0x1F</b>	

1B 19 00 0B 00 00 00 0E	
05	READ_MEMORY
B8	bei XMEGA neue Speicherklasse B8 für IO-Register
01 00 00 00	
00	Register-Index = R0
OUT 00 00 00 00E0 38	Start-Adresse, nicht relevant und CRC
1B 19 00 02 00 00 00 0E	
82	
IN 00 EA 06	
GPR werden fortlaufend gelesen	
1B 38 00 0B 00 00 00 0E	
05	
B8	bei XMEGA neue Speicherklasse B8 für IO-Register
01 00 00 00	
1F	Register-Index = R31
OUT 00 00 00 00 B4 FB	Start-Adresse, nicht relevant und CRC
1B 38 00 02 00 00 00 0E	
82	
IN 00 2E BC	
<b>Auslesen des CPU-Kontextes: Programm-Counter, Stackpointer und CPU-Statusregister</b>	
1B 39 00 01 00 00 00 0E	
OUT 07 4E B3	Programm-Counter = PC auslesen
1B 39 00 05 00 00 00 0E	
84	
IN 00 00 01 00 0E E3	PC steht auf 0x00 01 00 00
1B 3A 00 0B 00 00 00 0E	
05	READ_MEMORY
20	SRAM
01 00 00 00	
3D	Stackpointer -Low
OUT 00 00 00 00 FF 6D	
1B 3A 00 02 00 00 00 0E	
82	
IN FF AC 28	1 Byte Daten und CRC
1B 3B 00 0B 00 00 00 0E	
05	
20	
01 00 00 00	
3E	Stackpointer -High
OUT 00 00 00 00 D4 88	
1B 3B 00 02 00 00 00 0E	
82	
3F	
IN 5D A3	1 Byte Daten und CRC
1B 3C 00 0B 00 00 00 0E	
05	
20	
01 00 00 00	
3F	Status-Register
OUT 00 00 00 00 07 79	
1B 3C 00 02 00 00 00 0E	
82	
IN 00 CB 83	1 Byte Daten und CRC
1B 3D 00 0B 00 00 00 0E	
05	
B8	bei XMEGA neue Speicherklasse B8 für IO-Register
01 00 00 00	
1C	Register-Index = R28 -> Y-Low
00 00 00 00	
OUT 30 E5	
IN 1B 3D 00 02 00 00 00 0E	

82		
00 36 CE		1 Byte Daten und CRC
1B 3E 00 0B 00 00 00 0E		
05		
B8		bei XMEGA neue Speicherklasse B8 für IO-Register
01 00 00 00		
1D		Register-Index = R29 -> Y-High
OUT 00 00 00 00 4C EF		
1B 3E 00 02 00 00 00 0E		
82		
IN 00 31 18		1 Byte Daten und CRC
1B 3F 00 0B 00 00 00 0E		
05		
B8		bei XMEGA neue Speicherklasse B8 für IO-Register
01 00 00 00		
1C		Register-Index = R28 -> Y-Low
00 00 00 00		
OUT EF 1C		
1B 3F 00 02 00 00 00 0E		
IN 82 00 CC 55		1 Byte Daten und CRC
1B 40 00 0B 00 00 00 0E		
05		READ_MEMORY
B8		bei XMEGA neue Speicherklasse B8 für IO-Register
01 00 00 00		
1D		Register-Index = R29 -> Y-High
OUT 00 00 00 00 FC C3		
1B 40 00 02 00 00 00 0E		
IN 82 00 2A 2D		1 Byte Daten und CRC
1B 41 00 01 00 00 00 0E		
OUT 07 8B 16		Programm-Counter = PC auslesen
1B 41 00 05 00 00 00 0E		
84		
00 00 01 00		PC steht auf 0x00 01 00 00
IN EA 80		
1B 42 00 0E 00 00 00 0E		Neues undokumentiertes Kommando
37		zur Haltepunktbehandlung
00 00 00 00		Hier steht der PC-Wert für den HP 1
00 00 00 00		Hier steht der PC-Wert für den HP 2
00 00 00 00 00		
OUT 34 87		
IN 1B 42 00 01 00 00 00 0E 80 EC 6C		
1B 43 00 05 00 00 00 0E		
1C		RUN_TO_ADDRESS
0C 01 00 00		Starte zur Adresse
OUT 54 32		0x00 00 01 0C = main() -> impliziter Haltepunkt
IN 1B 43 00 01 00 00 00 0E 80 53 ED		
1B FF FF 08 00 00 00 0E		
E0		BREAK-EVENT
0C 01 00 00		PC = 0x00 00 01 0C = main() -> impliziter Haltepunkt
20		CAUSE = nicht dokumentiert, 0x20 wenn PC im gültigen Bereich
00 00		???????
IN 9D A0		
1B 44 00 01 00 00 00 0E		
OUT 07 EA 81		Programm-Counter = PC auslesen
1B 44 00 05 00 00 00 0E		
84		
0C 01 00 00		PC steht auf 0x00 00 01 0C
IN F9 D4		

Ab hier folgt der Ablauf dem obigen Standardablauf:

- Lesen des Programm-Counters, siehe Befehl vor diesem Text

- Lesen der Register R28 & R29 = Y-Reg
- Lesen aller GPR von R0 bis R31
- Lesen des CPU-Kontextes Programm-Counter, Stackpointer, S-Register

Der implizite Haltepunkt main() an Adresse 010C ist erreicht, damit hält der Prozess an.

### Ergebnis in STUDIO:

Processor	
Name	Value
Program Counter	0x0000010C
Stack Pointer	0x3FFC
X Register	0x0000
Y Register	0x3FFF
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	0
Frequency	
Stop Watch	
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00
R17	0x00
R18	0x00
R19	0x00
R20	0x00
R21	0x00
R22	0x00
R23	0x00
R24	0x00
R25	0x00
R26	0x00
R27	0x00
R28	0xFF
R29	0x3F
R30	0x00
R31	0x00



## Zu 2 Start Debugging mit anschließendem impliziten Break

b: Weiter mit F11 = Einzelschritt bis zum C-Befehl  $x = 0 \rightarrow x111$

- ➔ Vorlaufendes SIGN ON und der Programmupload sind weggelassen!
- ➔ Die Vorlaufphasen sind separat erläutert!
- ➔ Die Nachlaufphasen sind separat erläutert!

Der Debugger muss vom impliziten Haltepunkt `main() = 0x10C` weiterlaufen bis der PC auf `0x111`  $\rightarrow x=0$  steht.

OUT	1B B3 00 01 00 00 00 0E	
	07 7F 5E	Programm-Counter = PC auslesen
	1B B3 00 05 00 00 00 0E	
	84	
	0C 01 00 00	PC steht auf 0x00 00 01 0C = <code>main()</code>
IN	A9 91	
	1B B4 00 0E 00 00 00 0E	Neues undokumentiertes Kommando
	37	zur Haltepunktbehandlung
	00 00 00 00	Hier steht der PC-Wert für den HP 1
	00 00 00 00	Hier steht der PC-Wert für den HP 2
	00 00 00 00 00	Anmerkung: es ist KEIN expliziter HP gesetzt!
OUT	3F 5B	wir laufen nur 1 Step F11 weiter!
IN	1B B4 00 01 00 00 00 0E 80 C6 32	
	1B B5 00 03 00 00 00 0E	
	09	SINGLE_STEP durch Debugger
	01 01	Flag: Low-Level, Mode: Step-Into
OUT	00 33	
IN	1B B5 00 01 00 00 00 0E 80 79 B3	
	1B FF FF 08 00 00 00 0E	
	E0	BREAK-EVENT
	0D 01 00 00	PC = 0x00 00 01 0D = <code>main()</code> + 1
	20	CAUSE = nicht dokumentiert, 0x20 wenn PC im
	gültigen Bereich	
	00	
	00	??????
IN	48 3F	
	1B B6 00 0E 00 00 00 0E	
	37	wie oben
	00 00 00 00	
	00 00 00 00	
	00 00 00 00 00	
OUT	E7 35	
IN	1B B6 00 01 00 00 00 0E 80 A9 39	
	1B B7 00 03 00 00 00 0E	
	09 01 01	SINGLE_STEP durch Debugger
OUT	4E 6B	
IN	1B B7 00 01 00 00 00 0E 80 16 B8	
	1B FF FF 08 00 00 00 0E	
	E0	BREAK-EVENT
	0E 01 00 00	PC = 0x00 00 01 0E = <code>main()</code> + 2
	20	
	00 00	
IN	26 97	
	1B B8 00 0E 00 00 00 0E	
	37	
	00 00 00 00	
	00 00 00 00	
OUT	00 00 00 00 00	

```

FE 35
IN  1B B8 00 01 00 00 00 0E 80 A4 09
    1B B9 00 03 00 00 00 0E
    09 01 01
OUT B5 EA
IN  1B B9 00 01 00 00 00 0E 80 1B 88
    1B FF FF 08 00 00 00 0E
    E0
    0F 01 00 00
    20
    00 00
IN  F3 08
    1B BA 00 0E 00 00 00 0E
    37
    00 00 00 00
    00 00 00 00
OUT 00 00 00 00 00 26 5B
IN  1B BA 00 01 00 00 00 0E 80 CB 02
    1B BB 00 03 00 00 00 0E
    09 01 01
OUT FB B2
IN  1B BB 00 01 00 00 00 0E 80 74 83
    1B FF FF 08 00 00 00 0E
    E0
    10 01 00 00
    20
    00 00
IN  CE A4
    1B BC 00 0E 00 00 00 0E
    37
    00 00 00 00
    00 00 00 00
    00 00 00 00 00
OUT 4E E8
IN  1B BC 00 01 00 00 00 0E 80 7A 1F
    1B BD 00 03 00 00 00 0E
    09 01 01
OUT 29 5A
IN  1B BD 00 01 00 00 00 0E 80 C5 9E
    1B FF FF 08 00 00 00 0E
    E0
    11 01 00 00
    20
    00
    00
IN  1B 3B
    1B BE 00 01 00 00 00 0E
OUT 07 A2 E4
    1B BE 00 05 00 00 00 0E
    84
    11 01 00 00
IN  48 53

```

SINGLE\_STEP durch Debugger

BREAK-EVENT  
PC = 0x00 00 01 0F = main() + 3

BREAK-EVENT  
PC = 0x00 00 01 10 = main() + 4

SINGLE\_STEP durch Debugger

BREAK-EVENT  
PC = 0x00 00 01 11 = main() + 5  
???????

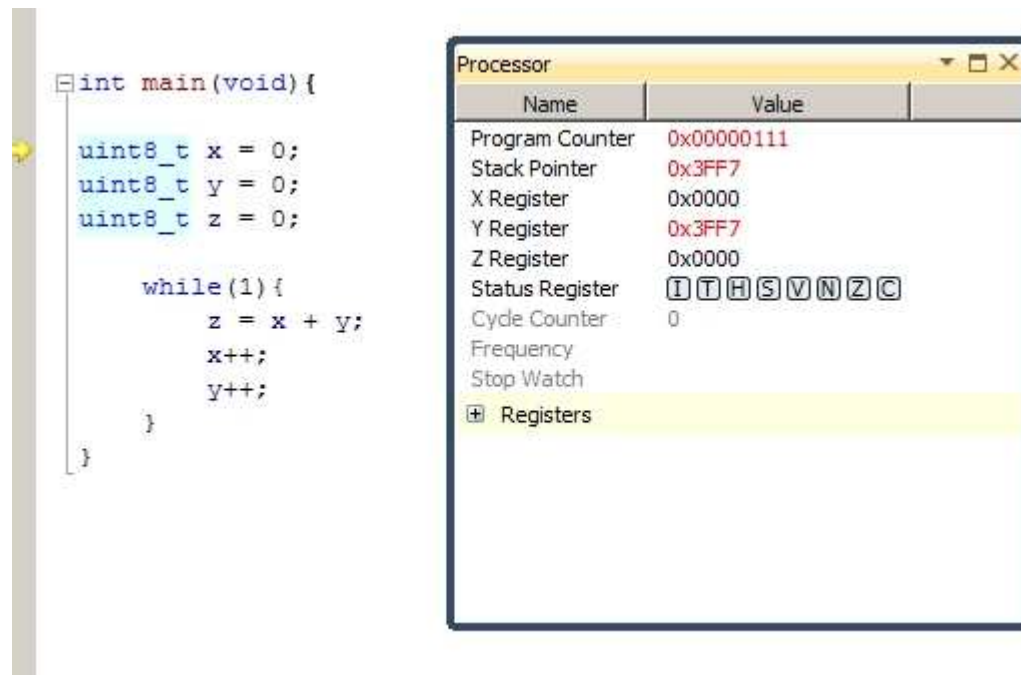
PC = 0x00 00 01 11 = main() + 5

Ab hier folgt der Ablauf dem Standardablauf:

- Lesen des Programm-Counters, siehe Befehl vor diesem Text
- Lesen der Register R28 & R29 = Y-Reg
- Lesen aller GPR von R0 bis R31
- Lesen des CPU-Kontextes Programm-Counter, Stackpointer, S-Register

Ein Step = F1 ist getan das C-Statement ist erreicht, damit hält der Prozess an.

## Ergebnis in STUDIO:



The image shows a screenshot of the Studio IDE. On the left, a C program is displayed in a code editor. The code defines a `main` function that initializes three `uint8_t` variables (`x`, `y`, and `z`) to 0. It then enters a `while(1)` loop where `z` is incremented by `x`, and `x` and `y` are each incremented by 1. On the right, the 'Processor' window is open, showing the state of the processor registers and status flags.

```
int main(void) {  
    uint8_t x = 0;  
    uint8_t y = 0;  
    uint8_t z = 0;  
  
    while(1) {  
        z = x + y;  
        x++;  
        y++;  
    }  
}
```

Name	Value
Program Counter	0x00000111
Stack Pointer	0x3FF7
X Register	0x0000
Y Register	0x3FF7
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	0
Frequency	
Stop Watch	
+ Registers	

## Zu 2 Start Debugging mit anschließendem impliziten Break

c: Weiter mit F5 bis zu einem expliziten Haltepunkt

- ➔ Vorlaufendes SIGN ON und der Programmupload sind weggelassen!
- ➔ Die Vorlaufphasen sind separat erläutert!
- ➔ Die Nachlaufphasen sind separat erläutert!

Es wird ein Haltepunkt auf dem C-Befehl `z = 0;` -> 0x113 gesetzt.

Der Debugger wird mit F5 = CONTINUE weitergestartet und muss vom impliziten Haltepunkt `main()` = 0x10C weiterlaufen bis der PC = 0x113 erreicht hat.

```
1B BB 00 01 00 00 00 0E
OUT 07 C3 73          Programm-Counter = PC auslesen
1B BB 00 05 00 00 00 0E 84
0C 01 00 00          PC steht auf 0x00 00 01 0C = main()
IN 45 9B
1B BC 00 0E 00 00 00 0E
37                  Neues undokumentiertes Kommando
13 01 00 00          zur Haltepunktbehandlung
00 00 00 00          Haltepunkt auf 0x113
00 00 00             Hier steht der PC-Wert für den HP 2
01                  ?????????
00                  Scheinbar Zähler für aktiven HP
OUT E4 20            ?????????
IN 1B BC 00 01 00 00 00 0E 80 7A 1F
1B BD 00 01 00 00 00 0E
OUT 08 85 96          Starte Programm Execution CMD_GO = F5
IN 1B BD 00 01 00 00 00 0E 80 C5 9E
1B FF FF 08 00 00 00 0E
E0                  BREAK-EVENT
13 01 00 00          PC = 0x00 00 01 13
10                  ???????
01 00               CAUSE = 0x01 Programm Break
IN D6 93
1B BE 00 01 00 00 00 0E
OUT 07 A2 E4          Programm-Counter = PC auslesen
1B BE 00 05 00 00 00 0E 84
13 01 00 00          PC steht auf 0x00 00 01 13 = gesetzter HP erreicht
IN 3E 6A
Ab hier folgt der Ablauf dem Standardablauf:
- Lesen des Programm-Counters, siehe Befehl vor diesem Text
- Lesen der Register R28 & R29 = Y-Reg
- Lesen aller GPR von R0 bis R31
- Lesen des CPU-Kontextes Programm-Counter, Stackpointer, S-Register

Hier nochmals am Fragment der Wert für das Y-Register = R28 & R29 gezeigt

1B BF 00 0B 00 00 00 0E
05                  READ_MEMORY
B8                  bei XMEGA neue Speicherklasse B8 für IO-Register
01 00 00 00
1C                  Lese R28
00 00 00 00
OUT E0 90
1B BF 00 02 00 00 00 0E
82
IN F7 2B 12          Daten und CRC
OUT 1B C0 00 0B 00 00 00 0E
```

05	READ_MEMORY
B8	bei XMEGA neue Speicherklasse B8 für IO-Register
01	
00 00 00	
1D	Lese R29
00 00 00 00	
F3 4F	
1B C0 00 02 00 00 00 0E	
82	
IN 3F 89 20	

Der Debugger wurde mit F5 gestartet, das Programm hat den Haltepunkt erreicht, damit hält der Prozess an. Das Debugging Tool selbst ist noch aktiv!

### Ergebnis in STUDIO:

```

int main(void){
    uint8_t x = 0;
    uint8_t y = 0;
    uint8_t z = 0;

    while(1){
        z = x + y;
        x++;
        y++;
    }
}

```

**Processor**

Name	Value
Program Counter	0x00000113
Stack Pointer	0x3FF7
X Register	0x0000
Y Register	0x3FF7
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	0
Frequency	
Stop Watch	
+ Registers	

### Zu 3 Start Debugging *ohne impliziten Halt*, mit vorher gesetztem Haltepunkt

a: Setzen eines Haltepunktes, danach Debugging starten

- ➔ Vorlaufendes SIGN ON und der Programmupload sind weggelassen!
- ➔ Die Vorlaufphasen sind separat erläutert!
- ➔ Die Nachlaufphasen sind separat erläutert!

Es wird ein Haltepunkt auf das C-Statement `x = 0;` -> 0x111 gesetzt, dann wird der Debugging-Prozess **direkt** gestartet.

Hier läuft noch der Startprozess des Debuggers, erst einmal muss der Einsprungpunkt `main()` erreicht werden, obwohl OHNE impliziten Halt gestartet wird!

1B 43 00 01 00 00 00 0E	
07 E4 1D	Programm-Counter = PC auslesen
1B 43 00 05 00 00 00 0E	
84	
00 00 01 00	PC steht auf 0x00 01 00 00
51 82	
1B 44 00 0E 00 00 00 0E	Neues undokumentiertes Kommando
37	zur Haltepunktbehandlung
11 01 00 00	Der gewählte Haltepunkt ist eingetragen
00 00 00 00	
00 00 00	
01	
00	
AC F7	
1B 44 00 01 00 00 00 0E 80 5D 71	
1B 45 00 05 00 00 00 0E	
1C	RUN_TO_ADDR
0C 01 00 00	PC = 0x10c -> main()
99 35	
1B 45 00 01 00 00 00 0E 80 E2 F0	
1B FF FF 08 00 00 00 0E	
E0	BREAK_EVENT
0C 01 00 00	PC = 0x10c -> main() ist erreicht
20	
00 00	
9D A0	
1B 46 00 01 00 00 00 0E	
07 85 8A	Programm-Counter = PC auslesen
1B 46 00 05 00 00 00 0E	
84	
0C 01 00 00	PC = 0x10c -> main()
42 D6	
1B 47 00 0E 00 00 00 0E	Neues undokumentiertes Kommando
37	zur Haltepunktbehandlung
11 01 00 00	Der gewählte Haltepunkt ist eingetragen
00 00 00 00	
00 00 00	
01	Haltepunktzähler???
00	
18 AE	
1B 47 00 01 00 00 00 0E 80 8D FB	
1B 48 00 01 00 00 00 0E	
08 7F 42	Start Programm Execution
1B 48 00 01 00 00 00 0E 80 3F 4A	

```

1B FF FF 08 00 00 00 0E
E0
11 01 00 00
10
01 00
6D A4
1B 49 00 01 00 00 00 0E
07 37 3B
1B 49 00 05 00 00 00 0E
84
11 01 00 00
18 16

```

BREAK\_EVENT  
PC = 0x111, der HP ist erreicht

Programm-Counter = PC auslesen

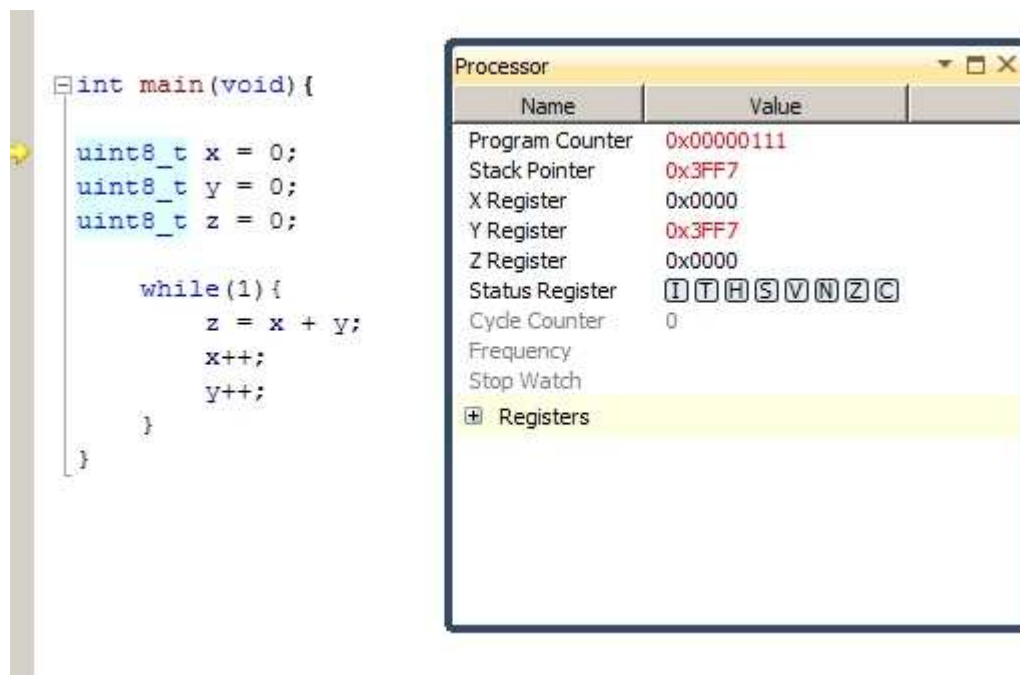
PC steht auf 0x111

Ab hier folgt der Ablauf dem Standardablauf:

- Lesen des Programm-Counters, siehe Befehl vor diesem Text
- Lesen der Register R28 & R29 = Y-Reg
- Lesen aller GPR von R0 bis R31
- Lesen des CPU-Kontextes Programm-Counter, Stackpointer, S-Register

Danach hält der Prozess an. Das Debugging Tool selbst ist noch aktiv!

## Ergebnis in STUDIO:



The screenshot shows the Studio IDE with a C program on the left and the Processor window on the right.

**Program Code:**

```

int main(void) {
    uint8_t x = 0;
    uint8_t y = 0;
    uint8_t z = 0;

    while(1) {
        z = x + y;
        x++;
        y++;
    }
}

```

**Processor Window:**

Name	Value
Program Counter	0x00000111
Stack Pointer	0x3FF7
X Register	0x0000
Y Register	0x3FF7
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	0
Frequency	
Stop Watch	
Registers	

### Zu 3 Start Debugging *ohne impliziten Halt*, mit vorher gesetztem Haltepunkt

b: Setzen eines weiteren Haltepunktes, weiter mit F11

- ➔ Vorlaufendes SIGN ON und der Programmupload sind weggelassen!
- ➔ Die Vorlaufphasen sind separat erläutert!
- ➔ Die Nachlaufphasen sind separat erläutert!

Es wird ein weiterer Haltepunkt auf das C-Statement  $z = 0$ ; -> 0x113 gesetzt, dann wird der Debugging-Prozess **mit F5** weitergestartet.

Vom vorhergehenden Test steht der PC auf dem Haltepunkt 1 = 0x111

OUT	1B C4 00 01 00 00 00 0E	
	07 08 4A	Programm-Counter = PC auslesen
	1B C5 00 0E 00 00 00 0E	Neues undokumentiertes Kommando
	37	zur Haltepunktbehandlung
	11 01 00 00	Der 1. Haltepunkt = 0x111 -> $x = 0$ ;
	13 01 00 00	Der 2. Haltepunkt = 0x113 -> $z = 0$ ;
	00 00 00	
	02	Zähler für aktive Haltepunkte?
OUT	00 2F 2D	
IN	1B C5 00 01 00 00 00 0E 80 00 3B	
	1B C6 00 01 00 00 00 0E	
OUT	08 90 B9	Start Programm Execution = F5
IN	1B C6 00 01 00 00 00 0E 80 D0 B1	
	1B FF FF 08 00 00 00 0E	
	E0	BREAK_EVENT
	13 01 00 00	PC = 0x113, der HP 2 ist erreicht
	10 02 00	CAUSE = ?????
IN	BE B9	
	1B C7 00 01 00 00 00 0E	
OUT	07 D8 C0	Programm-Counter = PC auslesen
	1B C7 00 05 00 00 00 0E	
	84	
	13 01 00 00	PC steht auf 0x113
IN	8F 8C	

Ab hier folgt der Ablauf dem Standardablauf:

- Lesen des Programm-Counters, siehe Befehl vor diesem Text
- Lesen der Register R28 & R29 = Y-Reg
- Lesen aller GPR von R0 bis R31
- Lesen des CPU-Kontextes Programm-Counter, Stackpointer, S-Register

Danach hält der Prozess an. Das Debugging Tool selbst ist noch aktiv!



## Ergebnis in STUDIO:

```
int main(void) {  
    uint8_t x = 0;  
    uint8_t y = 0;  
    uint8_t z = 0;  
  
    while(1) {  
        z = x + y;  
        x++;  
        y++;  
    }  
}
```

Processor	
Name	Value
Program Counter	0x00000113
Stack Pointer	0x3FF7
X Register	0x0000
Y Register	0x3FF7
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	0
Frequency	
Stop Watch	
+ Registers	

### Zu 3 Start Debugging *ohne impliziten Halt*, mit vorher gesetztem Haltepunkt

c: Rücksetzen des Haltepunktes, weiter mit F11

- ➔ Vorlaufendes SIGN ON und der Programmupload sind weggelassen!
- ➔ Die Vorlaufphasen sind separat erläutert!
- ➔ Die Nachlaufphasen sind separat erläutert!

Der Haltepunkt 1 auf das C-Statement  $x = 0$ ; -> 0x111 wird zurückgesetzt, dann wird der Debugging-Prozess **mit F11** weitergestartet.

Vom vorhergehenden Test steht der PC auf dem Haltepunkt 1 = 0x113

OUT	1B 18 01 01 00 00 00 0E	
	07 D4 1B	Programm-Counter = PC auslesen
	1B 18 01 05 00 00 00 0E	
	84	
	13 01 00 00	PC steht auf 0x113
IN	12 BD	
	1B 19 01 0E 00 00 00 0E	Neues undokumentiertes Kommando
	37	
	13 01 00 00	Jetzt steht hier nur der aktuelle Haltepunkt
	00 00 00 00	Eintrag gelöscht
	00 00 00	
	01	Zähler für aktive Haltepunkte?
OUT	00 23 A7	
IN	1B 19 01 01 00 00 00 0E 80 DC 6A	
	1B 1A 01 03 00 00 00 0E	
OUT	09 01 01 62 DD	SINGLE_STEP -> F11
IN	1B 1A 01 01 00 00 00 0E 80 0C E0	
	1B FF FF 08 00 00 00 0E	
	E0	BREAK_EVENT
	14 01 00 00	PC = 0x114,
	20 00 00	CAUSE =
IN	B8 CB	
	1B 1B 01 01 00 00 00 0E	
OUT	07 04 91	Programm-Counter = PC auslesen
	1B 1B 01 05 00 00 00 0E	
	84	
	14 01 00 00	PC = 0x114
IN	DD 6D	

Ab hier folgt der Ablauf dem Standardablauf:

- Lesen des Programm-Counters, siehe Befehl vor diesem Text
- Lesen der Register R28 & R29 = Y-Reg
- Lesen aller GPR von R0 bis R31
- Lesen des CPU-Kontextes Programm-Counter, Stackpointer, S-Register

Danach hält der Prozess an. Das Debugging Tool selbst ist noch aktiv!

## Ergebnis in STUDIO:

```
int main(void) {  
    uint8_t x = 0;  
    uint8_t y = 0;  
    uint8_t z = 0;  
  
    while(1) {  
        z = x + y;  
        x++;  
        y++;  
    }  
}
```

Processor	
Name	Value
Program Counter	0x00000114
Stack Pointer	0x3FF7
X Register	0x0000
Y Register	0x3FF7
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	0
Frequency	
Stop Watch	
+ Registers	

### Zu 3 Start Debugging *ohne impliziten Halt*, mit vorher gesetztem Haltepunkt

d: Setzen mehrerer Haltepunkte, weiter mit F11

- ➔ Vorlaufendes SIGN ON und der Programmupload sind weggelassen!
- ➔ Die Vorlaufphasen sind separat erläutert!
- ➔ Die Nachlaufphasen sind separat erläutert!

Es werden Haltepunkte auf die C-Statements  $x = 0$ ,  $y = 0$ ,  $x++$  und  $y++$  gesetzt, dann wird der Debugging-Prozess **mit F11** weitergestartet. Der nächste Haltepunkt liegt auf dem Programm-Counter 0x118, daher wird der Debugger intern 4 Einzelschritte ausführen.

Vom vorhergehenden Test steht der PC auf dem Haltepunkt 1 = 0x114

```
OUT 1B 6C 01 01 00 00 00 0E
    07 73 85
    1B 6C 01 05 00 00 00 0E
    84
    14 01 00 00          PC steht auf 0x113
IN   4D 86
    1B 6D 01 0E 00 00 00 0E
    37
    13 01 00 00          Historischer HP auf 0x113
    11 01 00 00          An 2.Position eingetragener HP
    00 00 00
    02
OUT  00 E5 63
IN   1B 6D 01 01 00 00 00 0E 80 7B F4
    1B 6E 01 08 00 00 00 0E
    11
    00
    00
    18 01 00 00          SET_BREAKPOINT
    00                  TYPE: ???
    00                  HP-Nummer: 0x00 = SW HP
    00                  An 4. Position eingetragener HP
    00                  MODE: Break on Memory read
OUT  84 A4
IN   1B 6E 01 01 00 00 00 0E 80 AB 7E
    1B 6F 01 08 00 00 00 0E
    11
    00
    00
    1B 01 00 00          SET_BREAKPOINT
    00                  TYPE: ???
    00                  HP-Nummer 0x00 = SW HP
    00                  An 5. Position eingetragener HP
    00                  MODE: Break on Memory read
OUT  62 F1
IN   1B 6F 01 01 00 00 00 0E 80 14 FF
    1B 70 01 08 00 00 00 0E
    11
    00
    00
    12 01 00 00          SET_BREAKPOINT
    00                  TYPE: ???
    00                  HP-Nummer: 0x00 = SW HP
    00                  An 2. Position eingetragener HP
    00                  MODE: Break on Memory read
OUT  57 A3
IN   1B 70 01 01 00 00 00 0E 80 DE 15
    1B 71 01 03 00 00 00 0E
OUT  09 01 01 EC 3D          SINGLE_STEP durch Debugger
IN   1B 71 01 01 00 00 00 0E 80 61 94
    1B FF FF 08 00 00 00 0E
    E0
IN   15 01 00 00          BREAK_EVENT
```

```

20 00 00
6D 54
1B 72 01 0E 00 00 00 0E
37
13 01 00 00
11 01 00 00
00 00 00
02
OUT 00 63 3A
IN 1B 72 01 01 00 00 00 0E 80 B1 1E
1B 73 01 03 00 00 00 0E
OUT 09 01 01 A2 65 SINGLE_STEP durch Debugger
IN 1B 73 01 01 00 00 00 0E 80 0E 9F
1B FF FF 08 00 00 00 0E
E0 BREAK_EVENT
16 01 00 00
20 00
IN 00 03 FC
1B 74 01 0E 00 00 00 0E
37
13 01 00 00
11 01 00 00
00 00 00
02
OUT 00 0B 89
IN 1B 74 01 01 00 00 00 0E 80 00 03
1B 75 01 03 00 00 00 0E
OUT 09 01 01 70 8D SINGLE_STEP durch Debugger
IN 1B 75 01 01 00 00 00 0E 80 BF 82
1B FF FF 08 00 00 00 0E
E0 BREAK_EVENT
17 01 00 00
20 00 00
IN D6 63
1B 76 01 0E 00 00 00 0E
37
13 01 00 00
11 01 00 00
00 00 00
02
OUT 00 D3 E7
IN 1B 76 01 01 00 00 00 0E 80 6F 08
1B 77 01 03 00 00 00 0E
OUT 09 01 01 3E D5 SINGLE_STEP durch Debugger
IN 1B 77 01 01 00 00 00 0E 80 D0 89
1B FF FF 08 00 00 00 0E
E0 BREAK_EVENT
18 01 00 00
20 00 00
IN 22 7A
1B 78 01 01 00 00 00 0E
OUT 07 D5 C8
1B 78 01 05 00 00 00 0E
84
18 01 00 00 PC = 0x118, Ziel erreicht
IN D7 01

```

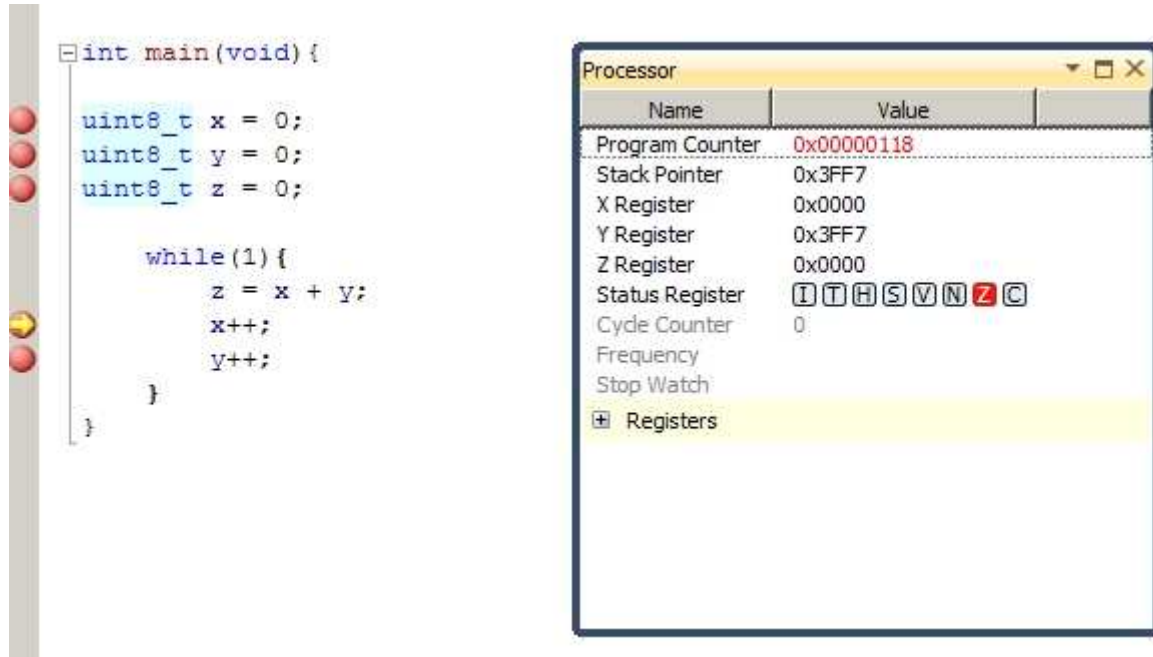
Ab hier folgt der Ablauf dem Standardablauf:

- Lesen des Programm-Counters, siehe Befehl vor diesem Text
- Lesen der Register R28 & R29 = Y-Reg
- Lesen aller GPR von R0 bis R31

- Lesen des CPU-Kontextes Programm-Counter, Stackpointer, S-Register

Danach hält der Prozess an. Das Debugging Tool selbst ist noch aktiv!

### Ergebnis in STUDIO:



The screenshot shows the Studio IDE with a C program and the Processor window open.

**Code:**

```
int main(void) {
    uint8_t x = 0;
    uint8_t y = 0;
    uint8_t z = 0;

    while(1) {
        z = x + y;
        x++;
        y++;
    }
}
```

**Processor Window:**

Name	Value
Program Counter	0x00000118
Stack Pointer	0x3FF7
X Register	0x0000
Y Register	0x3FF7
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	0
Frequency	
Stop Watch	
+ Registers	

### Zu 3 Start Debugging *ohne impliziten Halt*, mit vorher gesetztem Haltepunkt

e: Rücksetzen der Haltepunkte, weiter mit F11

- ➔ Vorlaufendes SIGN ON und der Programmupload sind weggelassen!
- ➔ Die Vorlaufphasen sind separat erläutert!
- ➔ Die Nachlaufphasen sind separat erläutert!

Es werden die Haltepunkte auf die C-Statements  $x = 0$ ,  $y = 0$ ,  $x++$  und  $y++$  zurückgesetzt. Im Kommando „37“ werden die beiden Einträge gelöscht und der HP-Zähler auf Null gesetzt, während die restlichen im vorhergehenden Test mit „11“ gesetzten Haltepunkte via Kommando „1A“ explizit gelöscht werden. Das verwendete Attribut NUMBER entspricht auch hier nicht der Dokumentation!

Dann wird der Debugging-Prozess **mit F11** weitergestartet. Da der nächste Halt des Debuggers auf dem C-Statement  $y++ \rightarrow 0x11B$  sein wird, muss der Debugger implizit drei SINGLE-STEP's ausführen (siehe lss und Disassembler).

Vom vorhergehenden Test steht der PC auf dem Haltepunkt 1 = 0x118

```
1B 11 01 01 00 00 00 0E
07                                Programm-Counter = PC auslesen
OUT D7 B7
1B 12 01 0E 00 00 00 0E          Neues undokumentiertes Kommando
37                                zur Haltepunktbehandlung
00 00 00 00                      HP1 ist gelöscht
00 00 00 00                      HP2 ist gelöscht
00 00 00
00
OUT 00 4C 85
IN 1B 12 01 01 00 00 00 0E 80 B0 CD
1B 13 01 06 00 00 00 0E
1A                                CLEAR_BREAKPOINT
00                                HP-Nummer
18 01 00 00                      Adresse HP 3 wird gelöscht
OUT 20 59
IN 1B 13 01 01 00 00 00 0E 80 0F 4C
1B 14 01 06 00 00 00 0E
1A                                CLEAR_BREAKPOINT
00                                HP-Nummer
1B 01 00 00                      Adresse HP 4 wird gelöscht
OUT AE 64
IN 1B 14 01 01 00 00 00 0E 80 01 D0
1B 15 01 03 00 00 00 0E
09 01 01                          SINGLE_STEP durch Debugger
OUT BE 70
IN 1B 15 01 01 00 00 00 0E 80 BE 51
1B FF FF 08 00 00 00 0E
E0                                BREAK_EVENT
19 01 00 00                      PC = 0x119
20
00 00
IN F7 E5
1B 16 01 0E 00 00 00 0E
37
00 00 00 00
OUT 00 00 00 00
```

```

00 00 00
00
00 FC 58
IN  1B 16 01 01 00 00 00 0E 80 6E DB
    1B 17 01 03 00 00 00 0E
    09 01 01                SINGLE_STEP durch Debugger

OUT  F0 28
IN  1B 17 01 01 00 00 00 0E 80 D1 5A
    1B FF FF 08 00 00 00 0E
    E0                    BREAK_EVENT
    1A 01 00 00          PC = 0x11A
    20
    00 00
IN  99 4D
    1B 18 01 0E 00 00 00 0E
    37
    00 00 00 00
    00 00 00 00
    00 00 00
    00
OUT  00 E5 58
IN  1B 18 01 01 00 00 00 0E 80 63 EB
    1B 19 01 03 00 00 00 0E
    09 01 01                SINGLE_STEP durch Debugger

OUT  0B A9
IN  1B 19 01 01 00 00 00 0E 80 DC 6A
    1B FF FF 08 00 00 00 0E
    E0                    BREAK_EVENT
    1B 01 00 00          PC = 0x11B
    20
    00 00
IN  4C D2
    1B 1A 01 01 00 00 00 0E
OUT  07 BB 10
    1B 1A 01 05 00 00 00 0E
    84
    1B 01 00 00          PC = 0x11B, Ziel erreicht
IN  71 5A

```

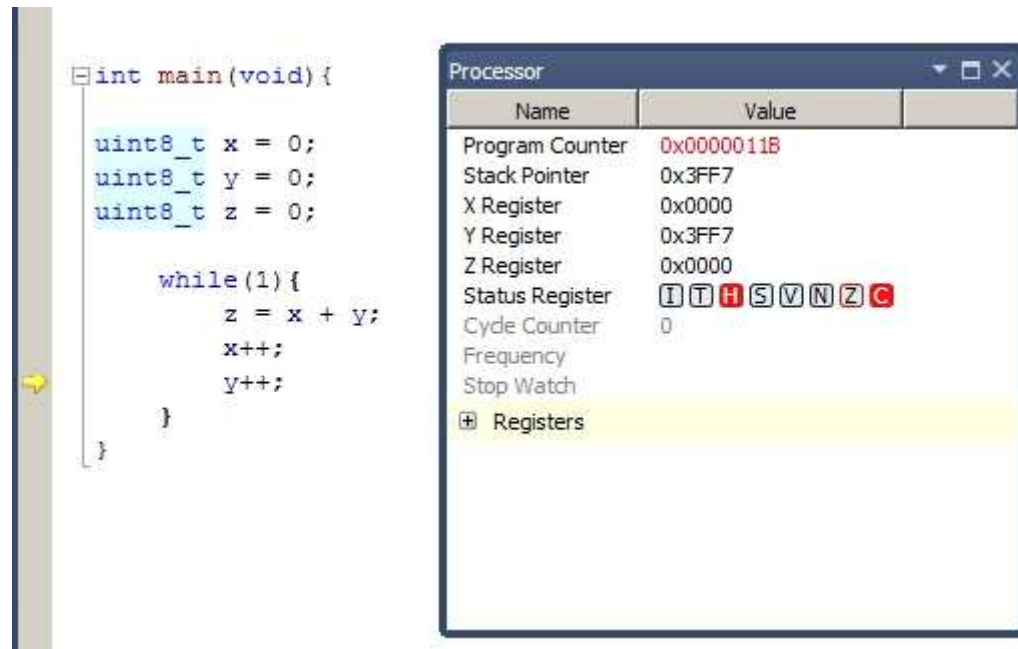
Ab hier folgt der Ablauf dem Standardablauf:

- Lesen des Programm-Counters, siehe Befehl vor diesem Text
- Lesen der Register R28 & R29 = Y-Reg
- Lesen aller GPR von R0 bis R31
- Lesen des CPU-Kontextes Programm-Counter, Stackpointer, S-Register

Danach hält der Prozess an. Das Debugging Tool selbst ist noch aktiv!



## Ergebnis in STUDIO:



The image shows the Studio IDE interface. On the left, a C program is displayed in the editor. The code defines three unsigned integer variables (x, y, z) and a while loop that increments x and y, and calculates their sum into z. A yellow arrow points to the start of the while loop. On the right, the 'Processor' window is open, showing the state of the microcontroller's registers and status flags.

```
int main(void) {  
    uint8_t x = 0;  
    uint8_t y = 0;  
    uint8_t z = 0;  
  
    while(1){  
        z = x + y;  
        x++;  
        y++;  
    }  
}
```

Name	Value
Program Counter	0x0000011B
Stack Pointer	0x3FF7
X Register	0x0000
Y Register	0x3FF7
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	0
Frequency	
Stop Watch	
+ Registers	

## Zu 4. Übertragung des Programmcodes

- ➔ Vorlaufendes SIGN ON ist weggelassen!
- ➔ Der Weiterstart nach dem Programm-Upload ist weggelassen!

Zu jedem Start des Debug-Prozesses wird im Allgemeinen der Programmcode auf das Ziel-Device geladen. Vorhergehendes Löschen und die Überprüfung des Programmspeichers nach der Übertragung sind wahlweise per Menü vorgebbar.

**Achtung: die *roten, kursiven Texte* sind hinzugefügte Hinweise zur Orientierung in der .lss- und Disassembler-Liste!**

```
Vorhergehender SIGN ON Prozess ist weggelassen

1B 08 00 01 00 00 00 0E
14                                Start Programming
OUT 63 FD
IN 1B 08 00 01 00 00 00 0E 80 CE 2F
1B 09 00 02 00 00 00 0E
03                                GET_PARAMETER
0E                                PARAMETER_ID
OUT 14 85                        CRC
1B 09 00 05 00 00 00 0E
81                                RSP_PARAMETER
3F C0 74 79                      Parameter = ????????
IN 99 6F                          CRC
1B 0A 00 0B 00 00 00 0E
05                                READ_MEMORY
B4                                SIGN_JTAG -> Device Signatur lesen
03 00 00 00                      BYTE_COUNT
00                                Register-Index
00 00 00 00                      START_ADDRESS
OUT 18 8D
1B 0A 00 04 00 00 00 0E
82                                RSP_MEMORY
1E 97 4C                          Daten = „1E 97 4C“ -> ATxmega128A1
IN 85 13
1B 0B 00 0B 00 00 00 0E
05                                READ_MEMORY
B2                                FUSE_BITS
01 00 00 00                      BYTE_COUNT
05                                Register-Index -> lese FUSE-Byte5
00 00 00 00
OUT 83 20                        CRC
1B 0B 00 02 00 00 00 0E
82                                RSP_MEMORY
FF                                Daten = 0xFF
IN FC 6D
1B 0C 00 06 00 00 00 0E
34                                XMEGA_ERASE
00                                ERASE_MODE: XMEGA_CHIP_ERASE
00 00 00 00                      ADDRESS -> Area to be erased
OUT E9 73
IN 1B 0C 00 01 00 00 00 0E 80 10 39
1B 0D 00 0B 00 00 00 0E
OUT 05                            READ_MEMORY
```

```

C0                                XMEGA_APPLICATION_FLASH
00 01 00 00                      BYTE_COUNT = 256 Byte
00                                Register-Index
02 00 00 00                      START_ADDRESS
41 86                             CRC
1B 0D 00 01 01 00 00 0E
82                                RSP_MEMORY
FF FF FF FF . . . . . FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
IN AD 76 CRC
1B 0E 00 0A 01 00 00 0E
04                                WRITE_MEMORY
C0                                XMEGA_APPLICATION_FLASH
00 01 00 00                      BYTE_COUNT = 256 Byte
00 00 00 00                      START_ADDRESS

```

Daten 256 Byte, beginnend bei 0x00 -> Interrupt-Vector-Code (siehe .iss und Disassembler)

```

00000000 <__vectors>:
OUT 0C 94 FA 00 0C 94 0A 01 .... 0C 94 0A 01 0C 94 0A 01 0C 94
OUT 0A 01 0C 94 0A 01 0C 94 .... 0C 94 0A 01 0C 94 0A 01 0C 94
OUT 0A 01 0C 94 0A 01 0C 94 .... 0C 94 0A 01 0C 94 0A 01 0C 94
OUT 0A 01 0C 94 0A 01 0C 94 .... 0C 94 0A 01 0C 94 0A 01 0C 94
    0A 01 0C 94 0A 01 0C 94 .... 0A 01 0C 94 0A 01 0C 94 0A 01
OUT 73 6D                                     CRC
    IN 1B 0E 00 01 00 00 00 0E 80 7F 32
        1B 0F 00 0A 01 00 00 0E
        04                                     WRITE_MEMORY
        C0                                     XMEGA_APPLICATION_FLASH
        00 01 00 00                             BYTE_COUNT = 256 Byte
        00 01 00 00                             START_ADDRESS
OUT 0C 94 0A 01 0C 94 0A 01 .... 0C 94 0A 01 0C 94 0A 01 0C 94
OUT 0A 01 0C 94 0A 01 0C 94 .... 0C 94 0A 01 0C 94 0A 01 0C 94
OUT 0A 01 0C 94 0A 01 0C 94 .... 0C 94 0A 01 0C 94 0A 01 0C 94
OUT 0A 01 0C 94 0A 01 0C 94 .... 0C 94 0A 01 0C 94 0A 01 0C 94
    0A 01 0C 94 0A 01
000001f4 <__ctors_end>:
    11 24 1F BE CF EF DF E3 DE BF CD BF
OUT 1C CE                                     CRC
    IN 1B 0F 00 01 00 00 00 0E 80 C0 B3
        1B 10 00 0A 01 00 00 0E
        04
        C0
        00 01 00 00
        00 02 00 00
        00 E0 0C BF 18 BE 19 BE 1A BE 1B BE 0E 94 0C 01 0C 94 1F 01
00000214 <__bad_interrupt>:
OUT 0C 94 00 00 DF 93 CF 93 00 D0 CD B7 DE B7 19 82 1A 82 1B 82 99 81 8A 81 89 0F
    8B 83 89 81 8F 5F 89 83 8A 81 8F 5F 8A 83 F5 CF
0000023e <__exit>:
    F8 94
00000240 <__stop_program>:
    FF CF
OUT FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ....
OUT FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ....
OUT FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ....
    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
OUT 53 07                                     CRC
    IN 1B 10 00 01 00 00 00 0E 80 0A 59
        1B 11 00 01 00 00 00 0E
OUT 15                                     Ende Programming Mode

```

91 1B CRC  
IN 1B 11 00 01 00 00 00 0E 80 B5 D8

Ab hier Start durch den Debugger z.B. bis zum impliziten Stop bei main()!

## Zu 5. Standardabläufe

a: Register lesen

- ➔ Vorlaufendes SIGN ON und der Programmupload sind weggelassen!
- ➔ Die Vorlaufphasen sind separat erläutert!
- ➔ Die Nachlaufphasen sind separat erläutert!

Zur Anzeige des Prozessor- und Programmkontextes zum Programmstart und nach jeder Unterbrechung des Programms werden die CPU-Register Programm-Counter, Stackpointer, S-Register und alle General-Purpose-Register ausgelesen. Hierzu wurde scheinbar für den XMEGA das READ\_MEMORY Kommando um den Parameter Register-Index und die Speicherdefinition um den Wert 0xB8 XMEGA\_IO\_REGISTER erweitert.

Die abgebildete Situation zeigt den Zustand zwischen dem Start des Debuggers und dem Start der Applikation. Wenn dieser Prozessschritt abgelaufen ist, sind das Y-Register für den Zugriff auf den Datenspeicher, der Stackpointer und der Programmcounter auf die Applikationswerte eingestellt.

Der grundsätzliche Ablauf ist immer:

- Programm-Counter auslesen,
- Y-Register lesen,
- Alle GPR lesen,
- CPU-Kontext lesen

1B 16 00 01 00 00 00 0E	
OUT 07 0C B4	Auslesen des Programm-Counters
1B 16 00 05 00 00 00 0E	
84	RSP_PC
00 00 01 00	Programm-Counter = 0x00 01 00 00 -> beim Debugger-Start!
IN CA 40	
1B 17 00 0B 00 00 00 0E	
05	READ_MEMORY
B8	XMEGA_IO_REGISTER
01 00 00 00	BYTE_COUNTER
1C	Register-Index: 0x1C => R28 = Y-Low
00 00 00 00	START_ADDRESS
OUT AF 06	
1B 17 00 02 00 00 00 0E	
82	RSP_MEMORY
00	Datenbyte
IN 3F DD	CRC
1B 18 00 0B 00 00 00 0E	
05	READ_MEMORY
B8	XMEGA_IO_REGISTER
01 00 00 00	BYTE_COUNTER
1D	Register-Index: 0x1D => R29 = Y-High
00 00 00 00	
OUT 33 08	
1B 18 00 02 00 00 00 0E	
82	RSP_MEMORY
00	Datenbyte
IN 17 4B	

```

1B 19 00 0B 00 00 00 0E
05
B8
01 00 00 00
00
00 00 00 00
OUTE0 38
1B 19 00 02 00 00 00 0E
82
00
INEA 06
1B 1A 00 0B 00 00 00 0E
05
B8
01 00 00 00
01
OUT00 00 00 00 9C 32
1B 1A 00 02 00 00 00 0E
82
00
INED D0

```

Register-Index: 0x00 => R0

Register-Index: 0x01 => R1

Es werden alle Register eingelesen, also R2, R3, ..... R29 und weiter

```

1B 37 00 0B 00 00 00 0E
05
B8
01 00 00 00
1E
00 00 00 00
OUT28 F5
1B 37 00 02 00 00 00 0E
82
00
IN06 2A
1B 38 00 0B 00 00 00 0E
05
B8
01 00 00 00
1F
00 00 00 00
OUTB4 FB
1B 38 00 02 00 00 00 0E
82
00
IN2E BC

```

Register-Index: 0x1E => R30 = Z-Low

Register-Index: 0x1E => R31 = Z-High

CPU-Kontext lesen

```

1B 39 00 01 00 00 00 0E
07
OUT4E B3
1B 39 00 05 00 00 00 0E
84
00 00 01 00
IN0E E3
1B 3A 00 0B 00 00 00 0E
05
20
01 00 00 00
OUT3D

```

Auslesen des Programm-Counters

Programm-Counter = 0x00 01 00 00 -> beim Debugger-Start!

READ\_MEMORY

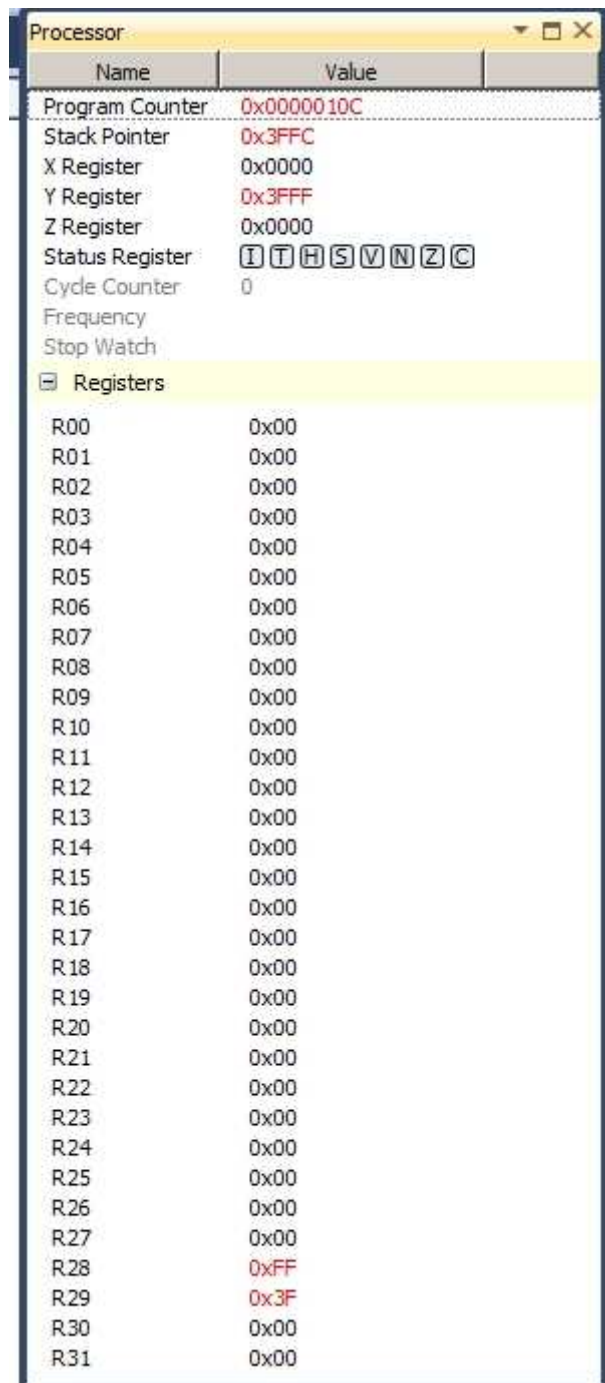
SRAM

BYTE\_COUNTER

Register-Index: 0x3D => SPL

00 00 00 00	
FF 6D	
1B 3A 00 02 00 00 00 0E	
82	RSP_MEMORY
FF	Datenbyte = 0xFF
IN AC 28	
1B 3B 00 0B 00 00 00 0E	
05	READ_MEMORY
20	SRAM
01 00 00 00	BYTE_COUNTER
3E	Register-Index: 0x3E => SPH
00 00 00 00	
OUT D4 88	
1B 3B 00 02 00 00 00 0E	
82	
3F	Datenbyte = 0x3F
IN 5D A3	
1B 3C 00 0B 00 00 00 0E	
05	READ_MEMORY
20	SRAM
01 00 00 00	
3F	Register-Index: 0x3F => SREG
00 00 00 00	
OUT 07 79	
1B 3C 00 02 00 00 00 0E	
82	
00	Datenbyte = 0x00
IN CB 83	
1B 3D 00 0B 00 00 00 0E	
05	
B8	XMEGA_IO_REGISTER
01 00 00 00	
1C	Register-Index: 0x1D => R29 = Y-High
00 00 00 00	
OUT 30 E5	
1B 3D 00 02 00 00 00 0E	
82	
00	
IN 36 CE	
1B 3E 00 0B 00 00 00 0E	
05	
B8	XMEGA_IO_REGISTER
01 00 00 00	
1D	Register-Index: 0x1D => R29 = Y-High
00 00 00 00	
OUT 4C EF	
1B 3E 00 02 00 00 00 0E	
82	
00	
IN 31 18	

## Ergebnis in STUDIO:



Name	Value
Program Counter	0x0000010C
Stack Pointer	0x3FFC
X Register	0x0000
Y Register	0x3FFF
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	0
Frequency	
Stop Watch	
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00
R17	0x00
R18	0x00
R19	0x00
R20	0x00
R21	0x00
R22	0x00
R23	0x00
R24	0x00
R25	0x00
R26	0x00
R27	0x00
R28	0xFF
R29	0x3F
R30	0x00
R31	0x00

Anzeige der Prozessor- und GPR-Register, hier ist bereits die Applikation aktiviert



## Zu 5. Standardabläufe

a: Speicherbereich: Variable lesen

- ➔ Vorlaufendes SIGN ON und der Programmupload sind weggelassen!
- ➔ Die Vorlaufphasen sind separat erläutert!
- ➔ Die Nachlaufphasen sind separat erläutert!

Die in einer Applikation verwandten Variablen, hier zum Beispiel die Variable y, kann separat überwacht werden. Der nachfolgende Ablauf zeigt dazu die Aktivitäten des Debuggers. Da die globalen Variablen x, y, z unterhalb des Aufruf-Stacks gespeichert werden ist die Adresse 0x3FF9:

Hierzu der relevante Code:

00000106	CALL 0x0000010C	Call subroutine	SP = 0x3FFC
int main(void){			
0000010C	PUSH R29	Push register on stack	SP = 0x3FFC
0000010D	PUSH R28	Push register on stack	SP = 0x3FFB
0000010E	RCALL PC+0x0001	Relative call subroutine	SP = 0x3FF7
0000010F	IN R28,0x3D	In from I/O location	Lade SP ins Y-Register
00000110	IN R29,0x3E	In from I/O location	Y = 0x3FF7

uint8\_t y = 0;  
 00000112 STD Y+2, R1      **Store indirect with displacement**  $0x3FF7 + 2 = \textcolor{red}{0x3FF9}$   
 uint8\_t z = 0;

1B 12 02 0B 00 00 00 0E	
05	READ_MEMORY
20	SRAM
01 00 00 00	BYTE_COUNTER
<b>F9 3F 00 00</b>	<b>Adresse der Variable y = 0x3FF9</b>
00	
OUT CD CF	CRC
1B 12 02 02 00 00 00 0E	
82	RSP_EMORY
<b>01</b>	<b>Datenbyte = 0x01</b>
IN C1 B5	CRC

**Ergebnis in STUDIO:**



## Zu 5. Standardabläufe

### c: Speicherbereiche schreiben / ändern

- ➔ Vorlaufendes SIGN ON und der Programmupload sind weggelassen!
- ➔ Die Vorlaufphasen sind separat erläutert!
- ➔ Die Nachlaufphasen sind separat erläutert!

Das Ändern direkt in einem der Speicher des ist über das Debug-Sub-Menü Memory möglich. Der Speichertyp und der Adressbereich ist auswählbar. Hier im Beispiel ändern eines Wertes im Stack. Stackadresse: 0x3FFF, der Stack liegt im Internal-SRAM.

## Auslesen des gewünschten Speichers und Bereiches

[illegible]

1. Auslesen der zu ändernden Speicherstelle: 0x3FFA
2. Einschreiben des neuen Wertes: 0x5A

	1B EB 00 0B 00 00 00 0E		
	05		READ_MEMORY
	20		SRAM
	01 00 00 00		BYTE_COUNTER
	FA 3F 00 00		ADRESSE
	00		????????
OUT	37 A8		
	1B EB 00 02 00 00 00 0E		
	82		RSP_MEMORY
	00		Datenbyte
IN	8C E5		
	1B EC 00 0B 00 00 00 0E		
	04		WRITE_MEMORY
	20		SRAM
	01 00 00 00		BYTE_COUNTER
	FA 3F 00 00		ADRESSE
	5A		Datenbyte
OUT	EE FA		
IN	1B EC 00 01 00 00 00 0E 80 F3 21		
	1B ED 00 0B 00 00 00 0E		Erneutes einlesen des Speicherbereiches
	05		READ_MEMORY
	20		
	6A 00 00 00		
OUT	96 3F 00 00		

IN

## Ergebnis in STUDIO:

Arbeitsspeicher 1

Memory:	Adresse:
0x2000,data	0x003FF0,data
data 0x003F60	00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x003F6D	00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x003F7A	00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x003F87	00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x003F94	00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x003FA1	00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x003FAE	00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x003FBB	00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x003FC8	00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x003FD5	00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x003FE2	00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x003FEF	00 00 00 00 00 00 00 00 00 5a 00 00 00 00
data 0x003FFC	00 00 01 08 ?? ?? ?? ?? ?? ?? ?? ??
data 0x004009	?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??

## Zu 6 Nicht geklärt

- b. Neue, undokumentierte Kommandos und Parameter

Studio5 und die aktuelle Firmware für den DRAGON enthält mehrere undokumentierte Änderungen des JTAGICE mkII Protokoll.

Hier eine Zusammenfassung der gefundenen Änderungen. Die nicht geklärten / dokumentierten Parameter sind mit ???? gekennzeichnet.

**Neues Kommando für PDI – Setup, ersetzt vermutlich Teile des Device-Descriptors. Ungeklärte Parameter!**

```
1B 10 00 33 00 00 00 0E
36
02 00 2F ???????
00 00 80 00 APP_SECTION_OFFSET
00 00 82 00 BOOT_SECTION_OFFSET
00 00 8C 00 EEPROM_OFFSET
20 00 8F 00 FUSE_REGISTERS_OFFSET
27 00 8F 00 LOCK_REGISTERS_OFFSET
00 04 8E 00 USER_SIGNATURES_OFFSET
00 02 8E 00 PROD_SIGNATURES_OFFSET
00 00 00 01 DATAMEM_OFFSET
00 00 02 00 BOOT-Section
00 20 00 02 00 08 20 ??????
C0 01 NVM-Base-Adresse
90 00 MCU-Base-Adresse
OUT D2 A0 CRC
IN 1B 10 00 01 00 00 00 0E 80 0A 59
```

**Neues Kommando zum Setzen von Haltepunkten, vermutlich werden hiermit die Hardware-Haltepunkte gesetzt.**

```
1B 14 00 0E 00 00 00 0E
37
00 00 00 00
00 00 00 00
00 00 00
00
00
OUT 72 E9
IN 1B 14 00 01 00 00 00 0E 80 D4 4F
```

Neues undokumentiertes Kommando  
zur Haltepunktbehandlung  
Hier steht der PC-Wert für den HP 1  
Hier steht der PC-Wert für den HP 2  
??????  
Zähler für aktive Haltepunkte????

**Definition einer zusätzlichen Speicherklasse B8 für die IO-Register bei XMEGA-Prozessoren**

```
1B 18 00 0B 00 00 00 0E
05
B8
01 00 00 00
OUT 1D
```

READ\_MEMORY  
**bei XMEGA neue Speicherklasse B8 für IO-Register**  
**Register-Index** = R29 -> Y-High

```

00 00 00 00 33 08
1B 18 00 02 00 00 00 0E
82
IN 00 17 4B

```

Antwort Lesen R29

**Achtung beim Lesen aus dem SRAM wird mit unterschiedlichen Parameterpositionen gearbeitet!**  
**Hier folgt nach dem Byte-Counter der Register-Index!**

```

1B 3B 00 0B 00 00 00 0E
05
20
01 00 00 00
3E
00 00 00 00
OUT D4 88
1B 3B 00 02 00 00 00 0E
82
3F
IN 5D A3

```

READ\_MEMORY  
**SRAM**  
 BYTE\_COUNTER  
**Register-Index: 0x3E => SPH**

Datenbyte = 0x3F

**Achtung beim Lesen aus dem SRAM wird mit unterschiedlichen Parameterpositionen gearbeitet!**  
**Hier folgt nach dem Byte-Counter die Speicheradresse!**

```

1B EB 00 0B 00 00 00 0E
05
20
01 00 00 00
FA 3F 00 00
00
OUT 37 A8
1B EB 00 02 00 00 00 0E
82
00
IN 8C E5

```

READ\_MEMORY  
**SRAM**  
 BYTE\_COUNTER  
**ADRESSE**  
**????????**

RSP\_MEMORY  
 Datenbyte

**Nicht dokumentierter Wert des Parameters TYPE.**

```

1B 6E 01 08 00 00 00 0E
11
00
18 01 00 00
00
OUT 84 A4
IN 1B 6E 01 01 00 00 00 0E 80 AB 7E

```

SET\_BREAKPOINT  
**TYPE: ???**  
 HP-Nummer: 0x00 = SW HP  
 An 4. Position eingetragener HP  
 MODE: Break on Memory read

*Im BREAK\_EVENT sind undokumentierte Parameter und Werte vorhanden!  
Die Zuordnung der CAUSE-Werte hat sich geändert!*

1B FF FF 08 00 00 00 0E	
E0	BREAK-EVENT
13 01 00 00	PC = 0x00 00 01 13
<i>10</i>	<i>??????</i>
<i>01 00</i>	<i>CAUSE = 0x01 Programm Break ????</i>
IN D6 93	

*Beim Debugger-Start wird ein Programm-Counter Wert benutzt, der ausserhalb des Adressraums liegt!*

1B 43 00 01 00 00 00 0E	
07 E4 1D	Programm-Counter = PC auslesen
1B 43 00 05 00 00 00 0E	
84	
<i>00 00 01 00</i>	<i>PC steht auf 0x00 01 00 00</i>
51 82	

## Anlagen:

### Kopie des Source-Codes:

```
#include <avr/io.h>
#include <stdio.h>
#include <inttypes.h>

int main(void){

    uint8_t x = 0;
    uint8_t y = 0;
    uint8_t z = 0;

    while(1){
        z = x + y;
        x++;
        y++;
    }
}
```

### Kopie der IIR.Iss (.elf)

IIR.elf: file format elf32-avr

#### Sections:

Idx	Name	Size	VMA	LMA	File off	Algn
0	.text	00000242	00000000	00000000	00000054	2**1
	CONTENTS, ALLOC, LOAD, READONLY, CODE					
1	.stab	00000714	00000000	00000000	00000298	2**2
	CONTENTS, READONLY, DEBUGGING					
2	.stabstr	00000085	00000000	00000000	000009ac	2**0
	CONTENTS, READONLY, DEBUGGING					
3	.debug_aranges	00000020	00000000	00000000	00000a31	2**0
	CONTENTS, READONLY, DEBUGGING					
4	.debug_pubnames	0000001b	00000000	00000000	00000a51	2**0
	CONTENTS, READONLY, DEBUGGING					
5	.debug_info	000000ae	00000000	00000000	00000a6c	2**0
	CONTENTS, READONLY, DEBUGGING					
6	.debug_abbrev	0000005d	00000000	00000000	00000b1a	2**0
	CONTENTS, READONLY, DEBUGGING					
7	.debug_line	000000f9	00000000	00000000	00000b77	2**0
	CONTENTS, READONLY, DEBUGGING					
8	.debug_frame	00000020	00000000	00000000	00000c70	2**2
	CONTENTS, READONLY, DEBUGGING					
9	.debug_str	00000079	00000000	00000000	00000c90	2**0
	CONTENTS, READONLY, DEBUGGING					
10	.debug_pubtypes	0000001e	00000000	00000000	00000d09	2**0
	CONTENTS, READONLY, DEBUGGING					

#### Disassembly of section .text:

```
00000000 <__vectors>:
  0: 0c 94 fa 00 jmp 0x1f4 ; 0x1f4 <__ctors_end>
  4: 0c 94 0a 01 jmp 0x214 ; 0x214 <__bad_interrupt>
      ----- gekürzt -----
1f0: 0c 94 0a 01 jmp 0x214 ; 0x214 <__bad_interrupt>

000001f4 <__ctors_end>:
1f4: 11 24 eor r1, r1
1f6: 1f be out 0x3f, r1 ; 63
```

```

1f8: cf ef      ldi    r28, 0xFF    ; 255
1fa: df e3      ldi    r29, 0x3F    ; 63
1fc: de bf      out    0x3e, r29    ; 62
1fe: cd bf      out    0x3d, r28    ; 61
200: 00 e0      ldi    r16, 0x00    ; 0
202: 0c bf      out    0x3c, r16    ; 60
204: 18 be      out    0x38, r1     ; 56
206: 19 be      out    0x39, r1     ; 57
208: 1a be      out    0x3a, r1     ; 58
20a: 1b be      out    0x3b, r1     ; 59
20c: 0e 94 0c 01 call   0x218 ; 0x218 <main>
210: 0c 94 1f 01 jmp     0x23e ; 0x23e <_exit>

00000214 <__bad_interrupt>:
214: 0c 94 00 00 jmp     0        ; 0x0 <__vectors>

00000218 <main>:
#include <avr/io.h>
#include <stdio.h>
#include <inttypes.h>

int main(void){
218: df 93      push   r29
21a: cf 93      push   r28
21c: 00 d0      rcall  .+0          ; 0x21e <main+0x6>
21e: cd b7      in     r28, 0x3d    ; 61
220: de b7      in     r29, 0x3e    ; 62

uint8_t x = 0;
222: 19 82      std    Y+1, r1      ; 0x01
uint8_t y = 0;
224: 1a 82      std    Y+2, r1      ; 0x02
uint8_t z = 0;
226: 1b 82      std    Y+3, r1      ; 0x03

    while(1){
        z = x + y;
228: 99 81      ldd    r25, Y+1      ; 0x01
22a: 8a 81      ldd    r24, Y+2      ; 0x02
22c: 89 0f      add    r24, r25
22e: 8b 83      std    Y+3, r24      ; 0x03
        x++;
230: 89 81      ldd    r24, Y+1      ; 0x01
232: 8f 5f      subi   r24, 0xFF      ; 255
234: 89 83      std    Y+1, r24      ; 0x01
        y++;
236: 8a 81      ldd    r24, Y+2      ; 0x02
238: 8f 5f      subi   r24, 0xFF      ; 255
23a: 8a 83      std    Y+2, r24      ; 0x02
    }
23c: f5 cf      rjmp   .-22        ; 0x228 <main+0x10>

0000023e <_exit>:
23e: f8 94      cli

00000240 <__stop_program>:
240: ff cf      rjmp   .-2        ; 0x240 <__stop_program>

```



## Kopie der Dissassemblierung

```
00000000 JMP 0x000000FA      Jump
00000002 JMP 0x0000010A    Jump

---- gekürzt -----

000000F8 JMP 0x0000010A      Jump
000000FA CLR R1             Clear Register
000000FB OUT 0x3F,R1        Out to I/O location
000000FC SER R28            Set Register
000000FD LDI R29,0x3F       Load immediate
000000FE OUT 0x3E,R29       Out to I/O location
000000FF OUT 0x3D,R28       Out to I/O location
00000100 LDI R16,0x00        Load immediate
00000101 OUT 0x3C,R16       Out to I/O location
00000102 OUT 0x38,R1        Out to I/O location
00000103 OUT 0x39,R1        Out to I/O location
00000104 OUT 0x3A,R1        Out to I/O location
00000105 OUT 0x3B,R1        Out to I/O location
00000106 CALL 0x0000010C    Call subroutine
00000108 JMP 0x0000011F     Jump
0000010A JMP 0x00000000     Jump
--- D:\Studio-WorkSpace\IIR\IIR\Debug\...\IIR.c -----
-
int main(void){
0000010C PUSH R29            Push register on stack
0000010D PUSH R28            Push register on stack
0000010E RCALL PC+0x0001     Relative call subroutine
0000010F IN R28,0x3D         In from I/O location
00000110 IN R29,0x3E         In from I/O location
--- D:\Studio-WorkSpace\IIR\IIR\Debug\...\IIR.c -----
-
uint8_t x = 0;
00000111 STD Y+1,R1          Store indirect with displacement
uint8_t y = 0;
00000112 STD Y+2,R1          Store indirect with displacement
uint8_t z = 0;
00000113 STD Y+3,R1          Store indirect with displacement
          z = x + y;
00000114 LDD R25,Y+1          Load indirect with displacement
00000115 LDD R24,Y+2          Load indirect with displacement
00000116 ADD R24,R25           Add without carry
00000117 STD Y+3,R24          Store indirect with displacement
          x++;
00000118 LDD R24,Y+1          Load indirect with displacement
00000119 SUBI R24,0xFF         Subtract immediate
0000011A STD Y+1,R24          Store indirect with displacement
          y++;
0000011B LDD R24,Y+2          Load indirect with displacement
0000011C SUBI R24,0xFF         Subtract immediate
--- D:\Studio-WorkSpace\IIR\IIR\Debug\...\IIR.c -----
-
0000011D STD Y+2,R24          Store indirect with displacement
          }
0000011E RJMP PC-0x000A      Relative jump
--- Keine Quelldatei -----
-
0000011F CLI                 Global Interrupt Disable
00000120 RJMP PC-0x0000      Relative jump
```