flAVR - Funkenstein Little AVR Virtual Runtime

Generated by Doxygen 1.8.6

Sun Oct 26 2014 20:44:36

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 _BreakPoint Struct Reference

Node-structure for a linked-list of breakpoint addresses.

`#include <breakpoint.h>`

Data Fields

- struct _BreakPoint * next
  Pointer to next breakpoint.
- struct _BreakPoint * prev
  Pointer to previous breakpoint.
- uint16_t u16Addr
  Address of the breakpoint.

3.1.1 Detailed Description

Node-structure for a linked-list of breakpoint addresses.

Definition at line 33 of file breakpoint.h.

The documentation for this struct was generated from the following file:

- breakpoint.h

3.2 _IOClockList Struct Reference

Data Fields

- struct _IOClockList * **next**
- void * **pvContext**
- PeriphClock **pfClock**

3.2.1 Detailed Description

Definition at line 44 of file avr_io.h.

The documentation for this struct was generated from the following file:

- avr_io.h

3.3 _IOReaderList Struct Reference

Data Fields

- struct _IOReaderList * **next**
- void * **pvContext**
- PeriphRead **pfReader**

3.3.1 Detailed Description

Definition at line 28 of file avr_io.h.

The documentation for this struct was generated from the following file:

- avr_io.h

3.4 _IOWriterList Struct Reference

Data Fields

- struct _IOWriterList * **next**
- void * **pvContext**
- PeriphWrite **pfWriter**

3.4.1 Detailed Description

Definition at line 36 of file avr_io.h.

The documentation for this struct was generated from the following file:

- avr_io.h

3.5 _WatchPoint Struct Reference

Data Fields

- struct _WatchPoint * next

  Pointer to next watchpoint.
- struct _WatchPoint * prev

  Pointer to previous watchpoint.
- uint16_t u16Addr

  Address (in RAM) to watch on.

3.5.1 Detailed Description

Definition at line 31 of file watchpoint.h.

3.5.2 Field Documentation

3.5.2.1 uint16_t _WatchPoint::u16Addr

Address (in RAM) to watch on.

Definition at line 36 of file watchpoint.h.

The documentation for this struct was generated from the following file:

- watchpoint.h

3.6 AVR_CoreRegisters Struct Reference

This is a bit of overkill, but there are reasons why the struct is presented as more than just a single array of 32 8-bit uints.

`#include <avr_coreregs.h>`

Data Fields

3.6.1 Detailed Description

This is a bit of overkill, but there are reasons why the struct is presented as more than just a single array of 32 8-bit uints.

Here, we create anonymous unions between the following core registers representations: 1) 32, 8-bit registers, as an array (r[0] through r[31]) 2) 16, 16-bit register-pairs, as an array (r_word[0] through r_word[15]) 3) 32, 8-bit registers, as named registers (r0 through r31) 4) 16, 16-bit register-pairs, as named registers(r1_0, through r31_30) 5) X, Y and Z registers map to r27_26, r29_28, and r31_30

Definition at line 38 of file avr_coreregs.h.

The documentation for this struct was generated from the following file:

- avr_coreregs.h

3.7 AVR_CPU Struct Reference

This structure effectively represents an entire simulated AVR CPU - all memories, registers (memory-mapped or internal), peripherals and housekeeping information.

`#include <avr_cpu.h>`

Data Fields

- IOReaderList * **apstPeriphReadTable** [CONFIG_IO_ADDRESS_BYTES]
- IOWriterList * **apstPeriphWriteTable** [CONFIG_IO_ADDRESS_BYTES]
- IOClockList * **pstClockList**
- struct _WatchPoint * **pstWatchPoints**
- struct _BreakPoint * **pstBreakPoints**
- uint16_t **u16PC**
- uint64_t **u64InstructionCount**
- uint64_t **u64CycleCount**

- uint32_t **u32CoreFreq**
- uint32_t **u32WDTCount**
- uint16_t **u16ExtraPC**
- uint16_t **u16ExtraCycles**
- bool **bAsleep**
- uint16_t * **Rd16**
- uint8_t * **Rd**
- uint16_t * **Rr16**
- uint8_t * **Rr**
- uint16_t **K**

- uint8_t **A**

uint8_t **b**

uint8_t **s**

uint8_t **q**

uint16_t * **pu16ROM**

uint8_t * **pu8EEPROM**

AVR_RAM_t * **pstRAM**

uint32_t **u32ROMSize**

uint32_t **u32EEPROMSize**

uint32_t **u32RAMSize**

uint8_t **u8IntPriority**

uint32_t **u32IntFlags**

InterruptAck **apfInterruptCallbacks** [32]

3.7.1 Detailed Description

This structure effectively represents an entire simulated AVR CPU - all memories, registers (memory-mapped or internal), peripherals and housekeeping information.

All new CPU functionality added to the emulator eventually winds up tied to this structure.

Definition at line 63 of file avr_cpu.h.

The documentation for this struct was generated from the following file:

- avr_cpu.h

3.8 AVR_CPU_Config_t Struct Reference

Struct defining parameters used to initialize the AVR CPU structure on startup.

`#include <avr_cpu.h>`

Data Fields

- uint32_t **u32ROMSize**
- uint32_t **u32RAMSize**
- uint32_t **u32EESize**

3.8.1 Detailed Description

Struct defining parameters used to initialize the AVR CPU structure on startup.

Definition at line 142 of file avr_cpu.h.

The documentation for this struct was generated from the following file:

- avr_cpu.h

3.9 AVR_RAM_t Struct Reference

union structure mapping the first 256 bytes of IO address space to an aray of bytes used to represent CPU RAM.

`#include <avr_cpu.h>`

Data Fields

3.9.1 Detailed Description

union structure mapping the first 256 bytes of IO address space to an aray of bytes used to represent CPU RAM.

Note that based on the runtime configuration, we'll purposefully malloc() a block of memory larger than the size of this struct to extend the au8RAM[] array to the appropriate size for the CPU target.

Definition at line 47 of file avr_cpu.h.

The documentation for this struct was generated from the following file:

- avr_cpu.h

3.10 AVR_Variant_t Struct Reference

This struct contains the information necessary to effectively describe an AVR Microcontroller variant among the rest of the code.

`#include <variant.h>`

Data Fields

- const char * szName

  Name for the variant, used for identification (i.e.
- uint32_t u32RAMSize

  RAM size for this variant.
- uint32_t u32ROMSize

  ROM size (in bytes) for this variant.
- uint32_t u32EESize

EEPROM size of this variant.

- const uint8_t * u8Descriptors

  A bytestream composed of feature descriptors.

3.10.1 Detailed Description

This struct contains the information necessary to effectively describe an AVR Microcontroller variant among the rest of the code.

Definition at line 48 of file variant.h.

3.10.2 Field Documentation

3.10.2.1 const char* AVR_Variant_t::szName

Name for the variant, used for identification (i.e.

"atmega328p")

Definition at line 50 of file variant.h.

The documentation for this struct was generated from the following file:

- variant.h

3.11 AVRPeripheral Struct Reference

Data Fields

- PeriphInit **pfInit**
- PeriphRead **pfRead**
- PeriphWrite **pfWrite**
- PeriphClock **pfClock**
- void * **pvContext**
- uint8_t **u8AddrStart**
- uint8_t **u8AddrEnd**

3.11.1 Detailed Description

Definition at line 41 of file avr_peripheral.h.

The documentation for this struct was generated from the following file:

- avr_peripheral.h

3.12 AVRRegisterFile Struct Reference

The first 256 bytes of the AVR memory space is composed of the core 32 general-purpse registers (R0-R31), and 224 bytes for the remaining I/O registers (used by peripherals).

```
#include <avr_registerfile.h>
```

Data Fields

- AVR_CoreRegisters **CORE_REGISTERS**
- AVR_PIN **PINA**
- AVR_DDR **DDRA**
- AVR_PORT **PORTA**
- AVR_PIN **PINB**
- AVR_DDR **DDRB**
- AVR_PORT **PORTB**
- AVR_PIN **PINC**
- AVR_DDR **DDRC**
- AVR_PORT **PORTC**
- AVR_PIN **PIND**
- AVR_DDR **DDRD**
- AVR_PORT **PORTD**
- uint8_t **RESERVED_0x2C**
- uint8_t **RESERVED_0x2D**
- uint8_t **RESERVED_0x2E**
- uint8_t **RESERVED_0x2F**
- uint8_t **RESERVED_0x30**
- uint8_t **RESERVED_0x31**
- uint8_t **RESERVED_0x32**
- uint8_t **RESERVED_0x33**
- uint8_t **RESERVED_0x34**
- AVR_TIFR0 **TIFR0**
- AVR_TIFR1 **TIFR1**
- AVR_TIFR2 **TIFR2**
- uint8_t **RESERVED_0x38**
- uint8_t **RESERVED_0x39**
- uint8_t **RESERVED_0x3A**
- AVR_PCIFR **PCIFR**
- AVR_EIFR **EIFR**
- AVR_EIMSK **EIMSK**
- uint8_t **GPIOR0**
- AVR_EECR **EECR**
- uint8_t **EEDR**
- uint8_t **EEARL**
- uint8_t **EEARH**
- AVR_GTCCR **GTCCR**
- AVR_TCCR0A **TCCR0A**
- AVR_TCCR0B **TCCR0B**
- uint8_t **TCNT0**
- uint8_t **OCR0A**
- uint8_t **OCR0B**
- uint8_t **RESERVED_0x49**
- uint8_t **GPIOR1**
- uint8_t **GPIOR2**
- AVR_SPCR **SPCR**
- AVR_SPSR **SPSR**
- uint8_t **SPDR**
- uint8_t **RESERVED_0x4F**
- AVR_ACSR **ACSR**
- uint8_t **RESERVED_0x51**
- uint8_t **RESERVED_0x52**
- AVR_SMCR **SMCR**

- AVR_MCUSR **MCUSR**
- AVR_MCUCR **MCUCR**
- uint8_t **RESERVED_0x56**
- AVR_SPMCSR **SPMCSR**
- uint8_t **RESERVED_0x58**
- uint8_t **RESERVED_0x59**
- uint8_t **RESERVED_0x5A**
- uint8_t **RESERVED_0x5B**
- uint8_t **RESERVED_0x5C**
- AVR_SPL **SPL**
- AVR_SPH **SPH**
- AVR_SREG **SREG**
- AVR_WDTCSR **WDTCSR**
- AVR_CLKPR **CLKPR**
- uint8_t **RESERVED_0x62**
- uint8_t **RESERVED_0x63**
- AVR_PRR **PRR**
- uint8_t **RESERVED_0x65**
- uint8_t **OSCCAL**
- uint8_t **RESERVED_0x67**
- AVR_PCICR **PCICR**
- AVR_EICRA **EICRA**
- uint8_t **RESERVED_0x6A**
- AVR_PCMSK0 **PCMSK0**
- AVR_PCMSK1 **PCMSK1**
- AVR_PCMSK2 **PCMSK2**
- AVR_TIMSK0 **TIMSK0**
- AVR_TIMSK1 **TIMSK1**
- AVR_TIMSK2 **TIMSK2**
- uint8_t **RESERVED_0x71**
- uint8_t **RESERVED_0x72**
- uint8_t **RESERVED_0x73**
- uint8_t **RESERVED_0x74**
- uint8_t **RESERVED_0x75**
- uint8_t **RESERVED_0x76**
- uint8_t **RESERVED_0x77**
- uint8_t **ADCL**
- uint8_t **ADCH**
- AVR_ADCSRA **ADSRA**
- AVR_ADCSRB **ADSRB**
- AVR_ADMUX **ADMXUX**
- uint8_t **RESERVED_0x7F**
- AVR_DIDR0 **DIDR0**
- AVR_DIDR1 **DIDR1**
- AVR_TCCR1A **TCCR1A**
- AVR_TCCR1B **TCCR1B**
- AVR_TCCR1C **TCCR1C**
- uint8_t **RESERVED_0x83**
- uint8_t **TCNT1L**
- uint8_t **TCNT1H**
- uint8_t **ICR1L**
- uint8_t **ICR1H**
- uint8_t **OCR1AL**
- uint8_t **OCR1AH**
- uint8_t **OCR1BL**

- uint8_t **OCR1BH**
- uint8_t **RESERVED_0x8C**
- uint8_t **RESERVED_0x8D**
- uint8_t **RESERVED_0x8E**
- uint8_t **RESERVED_0x8F**
- uint8_t **RESERVED_0x90**
- uint8_t **RESERVED_0x91**
- uint8_t **RESERVED_0x92**
- uint8_t **RESERVED_0x93**
- uint8_t **RESERVED_0x94**
- uint8_t **RESERVED_0x95**
- uint8_t **RESERVED_0x96**
- uint8_t **RESERVED_0x97**
- uint8_t **RESERVED_0x98**
- uint8_t **RESERVED_0x99**
- uint8_t **RESERVED_0x9A**
- uint8_t **RESERVED_0x9B**
- uint8_t **RESERVED_0x9C**
- uint8_t **RESERVED_0x9D**
- uint8_t **RESERVED_0x9E**
- uint8_t **RESERVED_0x9F**
- uint8_t **RESERVED_0xA0**
- uint8_t **RESERVED_0xA1**
- uint8_t **RESERVED_0xA2**
- uint8_t **RESERVED_0xA3**
- uint8_t **RESERVED_0xA4**
- uint8_t **RESERVED_0xA5**
- uint8_t **RESERVED_0xA6**
- uint8_t **RESERVED_0xA7**
- uint8_t **RESERVED_0xA8**
- uint8_t **RESERVED_0xA9**
- uint8_t **RESERVED_0xAA**
- uint8_t **RESERVED_0xAB**
- uint8_t **RESERVED_0xAC**
- uint8_t **RESERVED_0xAD**
- uint8_t **RESERVED_0xAE**
- uint8_t **RESERVED_0xAF**
- AVR_TCCR2A **TCCR2A**
- AVR_TCCR2B **TCCR2B**
- uint8_t **TCNT2**
- uint8_t **OCR2A**
- uint8_t **OCR2B**
- uint8_t **RESERVED_0xB5**
- AVR_ASSR **ASSR**
- uint8_t **RESERVED_0xB7**
- uint8_t **TWBR**
- AVR_TWSR **TWSR**
- AVR_TWAR **TWAR**
- uint8_t **TWDR**
- AVR_TWCR **TWCR**
- AVR_TWAMR **TWAMR**
- uint8_t **RESERVED_0xBE**
- uint8_t **RESERVED_0xBF**
- AVR_UCSR0A **UCSR0A**
- AVR_UCSR0B **UCSR0B**

- AVR_UCSR0C **UCSR0C**
- uint8_t **RESERVED_0xC3**
- uint8_t **UBRR0L**
- uint8_t **UBRR0H**
- uint8_t **UDR0**
- uint8_t **RESERVED_0xC7**
- uint8_t **RESERVED_0xC8**
- uint8_t **RESERVED_0xC9**
- uint8_t **RESERVED_0xCA**
- uint8_t **RESERVED_0xCB**
- uint8_t **RESERVED_0xCC**
- uint8_t **RESERVED_0xCD**
- uint8_t **RESERVED_0xCE**
- uint8_t **RESERVED_0xCF**
- uint8_t **RESERVED_0xD0**
- uint8_t **RESERVED_0xD1**
- uint8_t **RESERVED_0xD2**
- uint8_t **RESERVED_0xD3**
- uint8_t **RESERVED_0xD4**
- uint8_t **RESERVED_0xD5**
- uint8_t **RESERVED_0xD6**
- uint8_t **RESERVED_0xD7**
- uint8_t **RESERVED_0xD8**
- uint8_t **RESERVED_0xD9**
- uint8_t **RESERVED_0xDA**
- uint8_t **RESERVED_0xDB**
- uint8_t **RESERVED_0xDC**
- uint8_t **RESERVED_0xDD**
- uint8_t **RESERVED_0xDE**
- uint8_t **RESERVED_0xDF**
- uint8_t **RESERVED_0xE0**
- uint8_t **RESERVED_0xE1**
- uint8_t **RESERVED_0xE2**
- uint8_t **RESERVED_0xE3**
- uint8_t **RESERVED_0xE4**
- uint8_t **RESERVED_0xE5**
- uint8_t **RESERVED_0xE6**
- uint8_t **RESERVED_0xE7**
- uint8_t **RESERVED_0xE8**
- uint8_t **RESERVED_0xE9**
- uint8_t **RESERVED_0xEA**
- uint8_t **RESERVED_0xEB**
- uint8_t **RESERVED_0xEC**
- uint8_t **RESERVED_0xED**
- uint8_t **RESERVED_0xEE**
- uint8_t **RESERVED_0xEF**
- uint8_t **RESERVED_0xF0**
- uint8_t **RESERVED_0xF1**
- uint8_t **RESERVED_0xF2**
- uint8_t **RESERVED_0xF3**
- uint8_t **RESERVED_0xF4**
- uint8_t **RESERVED_0xF5**
- uint8_t **RESERVED_0xF6**
- uint8_t **RESERVED_0xF7**
- uint8_t **RESERVED_0xF8**

- uint8_t **RESERVED_0xF9**
- uint8_t **RESERVED_0xFA**
- uint8_t **RESERVED_0xFB**
- uint8_t **RESERVED_0xFC**
- uint8_t **RESERVED_0xFD**
- uint8_t **RESERVED_0xFE**
- uint8_t **RESERVED_0xFF**

3.12.1 Detailed Description

The first 256 bytes of the AVR memory space is composed of the core 32 general-purpse registers (R0-R31), and 224 bytes for the remaining I/O registers (used by peripherals).

This data structure maps these 256 bytes to their function. Note that each AVR variant has its own set of peripherals, so this struct definition may change as support for new targets is added. The original mapping is based off the periphals found on the atmega328p.

Definition at line 38 of file avr_registerfile.h.

The documentation for this struct was generated from the following file:

- avr_registerfile.h

3.13 HEX_Record_t Struct Reference

Data type used to represent a single Intel Hex Record.

`#include <intel_hex.h>`

Data Fields

- uint8_t u8ByteCount

  Number of bytes in this record.
- uint8_t u8RecordType

  Record type stored in this record.
- uint16_t u16Address

  16-bit address/offset in this record
- uint8_t u8Data [MAX_HEX_DATA_BYTES]

  Record data bytes.
- uint8_t u8Checksum

  8-bit Checksum for the record
- uint32_t u32Line

  Current line number in the file.

3.13.1 Detailed Description

Data type used to represent a single Intel Hex Record.

Definition at line 57 of file intel_hex.h.

The documentation for this struct was generated from the following file:

- intel_hex.h

3.14 Interactive_Command_t Struct Reference

Struct type used to map debugger command-line inputs to command handlers.

Data Fields

- const char * szCommand
  Command string, as input by the user.
- const char * szDescription
  Command description, printed by "help".
- Interactive_Handler pfHandler
  Pointer to handler function.

3.14.1 Detailed Description

Struct type used to map debugger command-line inputs to command handlers.

Definition at line 50 of file interactive.c.

The documentation for this struct was generated from the following file:

- interactive.c

3.15 Option_t Struct Reference

Local data structure used to define a command-line option.

Data Fields

- const char * szAttribute
  Name of the attribute (i.e.
- char * szParameter
  Parameter string associated with the option.
- bool bStandalone
  Attribute is standalone (no parameter value expected)

3.15.1 Detailed Description

Local data structure used to define a command-line option.

Definition at line 31 of file options.c.

3.15.2 Field Documentation

3.15.2.1 const char* Option_t::szAttribute

Name of the attribute (i.e.

what's parsed from the commandline)

Definition at line 33 of file options.c.

The documentation for this struct was generated from the following file:

- options.c

3.16 TraceBuffer_t Struct Reference

Data Fields

- TraceElement_t **astTraceStep** [CONFIG_TRACEBUFFER_SIZE]
- uint32_t **u32Index**

3.16.1 Detailed Description

Definition at line 46 of file trace_buffer.h.

The documentation for this struct was generated from the following file:

- trace_buffer.h

3.17 TraceElement_t Struct Reference

Data Fields

- uint64_t **u64Counter**
- uint64_t **u64CycleCount**
- uint16_t **u16OpCode**
- uint16_t **u16PC**
- uint16_t **u16SP**
- uint8_t **u8SR**
- AVR_CoreRegisters **stCoreRegs**

3.17.1 Detailed Description

Definition at line 32 of file trace_buffer.h.

The documentation for this struct was generated from the following file:

- trace_buffer.h

Chapter 4

File Documentation

4.1 avr_coreregs.h File Reference

Module containing struct definition for the core AVR registers.

```
#include <stdint.h>
```

Data Structures

- struct AVR_CoreRegisters

  This is a bit of overkill, but there are reasons why the struct is presented as more than just a single array of 32 8-bit uints.

4.1.1 Detailed Description

Module containing struct definition for the core AVR registers.

Definition in file avr_coreregs.h.

4.2 avr_coreregs.h

```
00001 /*************************************************************************
00002  *      (     (                       (     |
00003  *    )\ )  )\ )    (              )\ )   |
00004  *   (()/( (()/(    )\     (   (  (()/(   | -- [ Funkenstein ] ------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -----------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] -------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] ---------------
00008  *   | __| | |__    / _ \   \ V /  |   /   | -- [ Runtime ] ---------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\   |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *************************************************************************/
00021 #ifndef __AVR_COREREG_H__
00022 #define __AVR_COREREG_H__
00023
00024 #include <stdint.h>
00025
00038 typedef struct
00039 {
00040     union
00041     {
00042         uint8_t r[32];
00043         uint16_t r_word[16];
00044         struct
00045         {
```

```
00046             uint16_t r1_0;
00047             uint16_t r3_2;
00048             uint16_t r5_4;
00049             uint16_t r7_6;
00050             uint16_t r9_8;
00051             uint16_t r11_10;
00052             uint16_t r13_12;
00053             uint16_t r15_14;
00054             uint16_t r17_16;
00055             uint16_t r19_18;
00056             uint16_t r21_20;
00057             uint16_t r23_22;
00058             uint16_t r25_24;
00059             uint16_t r27_26;
00060             uint16_t r29_28;
00061             uint16_t r31_30;
00062         };
00063         struct
00064         {
00065             uint8_t r0;
00066             uint8_t r1;
00067             uint8_t r2;
00068             uint8_t r3;
00069             uint8_t r4;
00070             uint8_t r5;
00071             uint8_t r6;
00072             uint8_t r7;
00073             uint8_t r8;
00074             uint8_t r9;
00075             uint8_t r10;
00076             uint8_t r11;
00077             uint8_t r12;
00078             uint8_t r13;
00079             uint8_t r14;
00080             uint8_t r15;
00081             uint8_t r16;
00082             uint8_t r17;
00083             uint8_t r18;
00084             uint8_t r19;
00085             uint8_t r20;
00086             uint8_t r21;
00087             uint8_t r22;
00088             uint8_t r23;
00089             uint8_t r24;
00090             uint8_t r25;
00091             union
00092             {
00093                 uint16_t X;
00094                 struct
00095                 {
00096                     uint8_t r26;
00097                     uint8_t r27;
00098                 };
00099             };
00100             union
00101             {
00102                 uint16_t Y;
00103                 struct
00104                 {
00105                     uint8_t r28;
00106                     uint8_t r29;
00107                 };
00108             };
00109             union
00110             {
00111                 uint16_t Z;
00112                 struct
00113                 {
00114                     uint8_t r30;
00115                     uint8_t r31;
00116                 };
00117             };
00118         };
00119     };
00120 } AVR_CoreRegisters;
00121
00122 #endif
```

4.3 avr_cpu.c File Reference

AVR CPU emulator logic - this module contains the entrypoints required to implement CPU instruction fetch/decode/execute operations, using either lookup tables or direct-decoding logic.

```
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "emu_config.h"
#include "avr_cpu.h"
#include "avr_peripheral.h"
#include "avr_interrupt.h"
#include "avr_io.h"
#include "avr_op_decode.h"
#include "avr_op_size.h"
#include "avr_opcodes.h"
#include "avr_op_cycles.h"
#include "trace_buffer.h"
```

Functions

- static void **CPU_Decode** (uint16_t OP_)
- static void **CPU_Execute** (uint16_t OP_)
- uint16_t CPU_Fetch (void)

  CPU_Fetch Fetch the next opcode for the CPU object.
- static void **CPU_GetOpCycles** (uint16_t OP_)
- static void **CPU_GetOpSize** (uint16_t OP_)
- static void **CPU_PeripheralCycle** (void)
- void CPU_RunCycle (void)

  CPU_RunCycle Run a CPU instruction cycle.
- static void **CPU_BuildDecodeTable** (void)
- static void **CPU_BuildOpcodeTable** (void)
- static void **CPU_BuildSizeTable** (void)
- static void **CPU_BuildCycleTable** (void)
- void CPU_Init (AVR_CPU_Config_t *pstConfig_)

  CPU_Init Initialize the CPU object and its associated data.
- void CPU_AddPeriph (AVRPeripheral *pstPeriph_)

  CPU_AddPeriph Add a new I/O Peripheral to the CPU.
- void CPU_RegisterInterruptCallback (InterruptAck pfIntAck_, uint8_t ucVector_)

  CPU_RegisterInterruptCallback.

Variables

- AVR_CPU **stCPU**
- static AVR_Decoder astDecoders [65536] = { 0 }

  2 levels of jump tables are required for AVR.
- static AVR_Opcode **astOpcodes** [65536] = { 0 }
- static uint8_t **au8OpSizes** [65536] = { 0 }
- static uint8_t **au8OpCycles** [65536] = { 0 }

4.3.1 Detailed Description

AVR CPU emulator logic - this module contains the entrypoints required to implement CPU instruction fetch/decode/execute operations, using either lookup tables or direct-decoding logic.

Definition in file avr_cpu.c.

4.3.2 Function Documentation

4.3.2.1 void CPU_AddPeriph (AVRPeripheral * pstPeriph_)

CPU_AddPeriph Add a new I/O Peripheral to the CPU.

Parameters

*pstPeriph_*	Pointer to an initialized AVR Peripheral object to be associated with this CPU.

Definition at line 262 of file avr_cpu.c.

4.3.2.2 uint16_t CPU_Fetch (void)

CPU_Fetch Fetch the next opcode for the CPU object.

Returns

First word of the next opcode

Definition at line 87 of file avr_cpu.c.

4.3.2.3 void CPU_Init (AVR_CPU_Config_t * pstConfig_)

CPU_Init Initialize the CPU object and its associated data.

Parameters

*pstConfig_*	Pointer to an initialized AVR_CPU_Config_t struct

Definition at line 227 of file avr_cpu.c.

4.3.2.4 void CPU_RegisterInterruptCallback (InterruptAck pfIntAck_, uint8_t ucVector_)

CPU_RegisterInterruptCallback.

Install a function callback to be run whenever a specific interrupt vector is run. This is useful for resetting peripheral registers once a specific type of interrupt has been acknowledged.

Parameters

*pfIntAck_*	Callback function to register
*ucVector_*	Interrupt vector index to install handler at

Definition at line 280 of file avr_cpu.c.

4.3.2.5 void CPU_RunCycle (void)

CPU_RunCycle Run a CPU instruction cycle.

This performs Fetch, Decode, Execute, Clock updates, and Interrupt handling.

Definition at line 124 of file avr_cpu.c.

4.3.3 Variable Documentation

4.3.3.1 AVR_Decoder astDecoders[65536] = { 0 } [static]

2 levels of jump tables are required for AVR.

The first is to implement addressing mode detection (which we then use to seed the appropriate intermediate register pointers in the AVR_CPU struct).

This greatly reduces opcode function complexity, saves lots of code. Second-level is a pure jump-table to opcode function pointers, where the CPU register pointers are used w/AVR_CPU struct data to execute the opcode.

Definition at line 57 of file avr_cpu.c.

4.4 avr_cpu.c

```
00001 /*****************************************************************************
00002  *      (     (                        (     |
00003  *    )\ )  )\ )     (               )\ )   |
00004  *   (()/( (()/(     )\     (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                         | "Yeah, it does Arduino..."
00011  * ----------------------------------------+-------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00023 #include <stdint.h>
00024 #include <stdio.h>
00025 #include <string.h>
00026 #include <stdlib.h>
00027
00028 #include "emu_config.h"
00029
00030 #include "avr_cpu.h"
00031 #include "avr_peripheral.h"
00032 #include "avr_interrupt.h"
00033 #include "avr_io.h"
00034 #include "avr_op_decode.h"
00035 #include "avr_op_size.h"
00036 #include "avr_opcodes.h"
00037 #include "avr_op_cycles.h"
00038
00039 #include "trace_buffer.h"
00040
00041 AVR_CPU stCPU;
00042
00043 #if FEATURE_USE_JUMPTABLES
00044 //-----------------------------------------------------------------------------
00055 //-----------------------------------------------------------------------------
00056
00057 static AVR_Decoder astDecoders[65536] = { 0 };
00058 static AVR_Opcode  astOpcodes[65536] = { 0 };
00059 static uint8_t     au8OpSizes[65536] = { 0 };
00060 static uint8_t     au8OpCycles[65536] = { 0 };
00061
00062 #endif
00063
00064 //-----------------------------------------------------------------------------
00065 static void CPU_Decode( uint16_t OP_ )
00066 {
00067 #if FEATURE_USE_JUMPTABLES
00068     astDecoders[OP_]( OP_);
00069 #else
00070     AVR_Decoder pfOp = AVR_Decoder_Function( OP_ );
00071     pfOP(  OP_ );
00072 #endif
00073 }
00074
00075 //-----------------------------------------------------------------------------
00076 static void CPU_Execute( uint16_t OP_ )
00077 {
00078 #if FEATURE_USE_JUMPTABLES
00079     astOpcodes[OP_]();
00080 #else
00081     AVR_Opcode pfOp = AVR_Opcode_Function(OP_);
00082     pfOP(  OP_ );
00083 #endif
00084 }
00085
00086 //-----------------------------------------------------------------------------
00087 uint16_t CPU_Fetch( void )
00088 {
00089     uint16_t PC = stCPU.u16PC;
00090     if (PC >= 16384)
```

```
00091     {
00092         return 0xFFFF;
00093     }
00094     return stCPU.pu16ROM[ stCPU.u16PC ];
00095 }
00096
00097 //------------------------------------------------------------------------------
00098 static void CPU_GetOpCycles( uint16_t OP_ )
00099 {
00100 #if FEATURE_USE_JUMPTABLES
00101     stCPU.u16ExtraCycles = au8OpCycles[ OP_ ];
00102 #else
00103     stCPU.u16ExtraCycles = AVR_Opcode_Cycles( OP_ );
00104 #endif
00105 }
00106
00107 //------------------------------------------------------------------------------
00108 static void CPU_GetOpSize( uint16_t OP_ )
00109 {
00110 #if FEATURE_USE_JUMPTABLES
00111     stCPU.u16ExtraPC = au8OpSizes[ OP_ ];
00112 #else
00113     stCPU.u16ExtraPC = AVR_Opcode_Size( OP_ );
00114 #endif
00115 }
00116
00117 //------------------------------------------------------------------------------
00118 static void CPU_PeripheralCycle( void )
00119 {
00120     IO_Clock();
00121 }
00122
00123 //------------------------------------------------------------------------------
00124 void CPU_RunCycle( void )
00125 {
00126     uint16_t OP;
00127
00128     if (!stCPU.bAsleep)
00129     {
00130
00131         OP = CPU_Fetch();
00132
00133         // From the first word fetched, figure out how big this opcode is
00134         // (either 16 or 32-bit)
00135         CPU_GetOpSize(  OP );
00136
00137         // Based on the first word fetched, figure out the minimum number of
00138         // CPU cycles required to execute the instruction fetched.
00139         CPU_GetOpCycles(  OP );
00140
00141         // Decode the instruction, load internal registers with appropriate
00142         // values.
00143         CPU_Decode(  OP );
00144
00145         // Execute the instruction that was just decoded
00146         CPU_Execute(  OP );
00147
00148         // Update the PC based on the size of the instruction + whatever
00149         // modifications occurred during the execution cycle.
00150         stCPU.u16PC += stCPU.u16ExtraPC;
00151
00152         // Add CPU clock cycles to the global cycle counter based on
00153         // the minimum instruction time, plus whatever modifiers are applied
00154         // during execution of the instruction.
00155         stCPU.u64CycleCount += stCPU.u16ExtraCycles;
00156
00157         // Cycle-accurate peripheral clocking -- one iteration for each
00158         // peripheral for each CPU cycle of the instruction.
00159         // Note that CPU Interrupts are generated in the peripheral
00160         // phase of the instruction cycle.
00161         while (stCPU.u16ExtraCycles--)
00162         {
00163             CPU_PeripheralCycle();
00164         }
00165
00166         // Increment the "total executed instruction counter"
00167         stCPU.u64InstructionCount++;
00168
00169     }
00170     else
00171     {
00172         // CPU is asleep, just NOP and wait until we hit an interrupt.
00173         stCPU.u64CycleCount++;
00174         CPU_PeripheralCycle();
00175     }
00176
00177     // Check to see if there are any pending interrupts - if so, vector
```

```
00178     // to the appropriate location.  This has no effect if no interrupts
00179     // are pending
00180     AVR_Interrupt();
00181 }
00182
00183
00184 #if FEATURE_USE_JUMPTABLES
00185 //------------------------------------------------------------------------------
00186 static void CPU_BuildDecodeTable(void)
00187 {
00188     uint32_t i;
00189     for (i = 0; i < 65536; i++)
00190     {
00191         astDecoders[i] = AVR_Decoder_Function(i);
00192     }
00193 }
00194
00195 //------------------------------------------------------------------------------
00196 static void CPU_BuildOpcodeTable(void)
00197 {
00198     uint32_t i;
00199     for (i = 0; i < 65536; i++)
00200     {
00201         astOpcodes[i] = AVR_Opcode_Function(i);
00202     }
00203 }
00204
00205 //------------------------------------------------------------------------------
00206 static void CPU_BuildSizeTable(void)
00207 {
00208     uint32_t i;
00209     for (i = 0; i < 65536; i++)
00210     {
00211         au8OpSizes[i] = AVR_Opcode_Size(i);
00212     }
00213 }
00214
00215 //------------------------------------------------------------------------------
00216 static void CPU_BuildCycleTable(void)
00217 {
00218     uint32_t i;
00219     for (i = 0; i < 65536; i++)
00220     {
00221         au8OpCycles[i] = AVR_Opcode_Cycles(i);
00222     }
00223 }
00224 #endif
00225
00226 //------------------------------------------------------------------------------
00227 void CPU_Init( AVR_CPU_Config_t *pstConfig_ )
00228 {
00229     memset( &stCPU, 0, sizeof(stCPU));
00230     pstConfig_->u32RAMSize += 256;
00231
00232     // Dynamically allocate memory for RAM, ROM, and EEPROM buffers
00233     stCPU.pu8EEPROM = (uint8_t*)malloc( pstConfig_->u32EESize );
00234     stCPU.pu16ROM    = (uint16_t*)malloc( pstConfig_->u32ROMSize );
00235     stCPU.pstRAM    = (AVR_RAM_t*)malloc( pstConfig_->u32RAMSize );
00236
00237     stCPU.u32ROMSize = pstConfig_->u32ROMSize;
00238     stCPU.u32RAMSize = pstConfig_->u32RAMSize;
00239     stCPU.u32EEPROMSize = pstConfig_->u32EESize;
00240
00241     memset( stCPU.pu8EEPROM, 0, pstConfig_->u32EESize );
00242     memset( stCPU.pu16ROM, 0, pstConfig_->u32ROMSize );
00243     memset( stCPU.pstRAM, 0, pstConfig_->u32RAMSize );
00244
00245     // Set the base stack pointer to top-of-ram.
00246     uint16_t u16InitialStack = 256 + pstConfig_->u32RAMSize - 1;
00247     stCPU.pstRAM->stRegisters.SPH.r = (uint8_t)(u16InitialStack >> 8);
00248     stCPU.pstRAM->stRegisters.SPL.r = (uint8_t)(u16InitialStack & 0xFF);
00249
00250     // Reset the interrupt priority register
00251     stCPU.u8IntPriority = 255;
00252
00253 #if FEATURE_USE_JUMPTABLES
00254     CPU_BuildCycleTable();
00255     CPU_BuildSizeTable();
00256     CPU_BuildOpcodeTable();
00257     CPU_BuildDecodeTable();
00258 #endif
00259 }
00260
00261 //------------------------------------------------------------------------------
00262 void CPU_AddPeriph( AVRPeripheral *pstPeriph_ )
00263 {
00264     IO_AddClocker(  pstPeriph_ );
```

```
00265
00266     uint8_t i;
00267     for (i = pstPeriph_->u8AddrStart; i <= pstPeriph_->u8AddrEnd; i++)
00268     {
00269         IO_AddReader(  pstPeriph_, i );
00270         IO_AddWriter(  pstPeriph_, i );
00271     }
00272
00273     if (pstPeriph_->pfInit)
00274     {
00275         pstPeriph_->pfInit( pstPeriph_->pvContext );
00276     }
00277 }
00278
00279 //-----------------------------------------------------------------------------
00280 void CPU_RegisterInterruptCallback( InterruptAck pfIntAck_, uint8_t ucVector_
      )
00281 {
00282     if (ucVector_ >= 32)
00283     {
00284         return;
00285     }
00286
00287     stCPU.apfInterruptCallbacks[ ucVector_ ] = pfIntAck_;
00288 }
```

4.5 avr_cpu.h File Reference

AVR CPU Emulator - virtual AVR structure declarations and functions used to drive the emulator (fetch/decode/execute).

```
#include <stdint.h>
#include <stdbool.h>
#include "emu_config.h"
#include "avr_peripheral.h"
#include "avr_periphregs.h"
#include "avr_coreregs.h"
#include "avr_registerfile.h"
#include "avr_io.h"
#include "watchpoint.h"
#include "breakpoint.h"
```

Data Structures

- struct AVR_RAM_t

  union structure mapping the first 256 bytes of IO address space to an aray of bytes used to represent CPU RAM.
- struct AVR_CPU

  This structure effectively represents an entire simulated AVR CPU - all memories, registers (memory-mapped or internal), peripherals and housekeeping information.
- struct AVR_CPU_Config_t

  Struct defining parameters used to initialize the AVR CPU structure on startup.

Functions

- void CPU_Init (AVR_CPU_Config_t *pstConfig_)

  CPU_Init Initialize the CPU object and its associated data.
- uint16_t CPU_Fetch (void)

  CPU_Fetch Fetch the next opcode for the CPU object.
- void CPU_RunCycle (void)

  CPU_RunCycle Run a CPU instruction cycle.
- void CPU_AddPeriph (AVRPeripheral *pstPeriph_)

  CPU_AddPeriph Add a new I/O Peripheral to the CPU.

- void CPU_RegisterInterruptCallback (InterruptAck pfIntAck_, uint8_t ucVector_)
  CPU_RegisterInterruptCallback.

Variables

- AVR_CPU **stCPU**

4.5.1 Detailed Description

AVR CPU Emulator - virtual AVR structure declarations and functions used to drive the emulator (fetch/decode/execute).

Definition in file avr_cpu.h.

4.5.2 Function Documentation

4.5.2.1 void CPU_AddPeriph (AVRPeripheral * *pstPeriph_*)

CPU_AddPeriph Add a new I/O Peripheral to the CPU.

Parameters

*pstPeriph_*	Pointer to an initialized AVR Peripheral object to be associated with this CPU.

Definition at line 262 of file avr_cpu.c.

4.5.2.2 uint16_t CPU_Fetch (void)

CPU_Fetch Fetch the next opcode for the CPU object.

Returns

First word of the next opcode

Definition at line 87 of file avr_cpu.c.

4.5.2.3 void CPU_Init (AVR_CPU_Config_t * *pstConfig_*)

CPU_Init Initialize the CPU object and its associated data.

Parameters

*pstConfig_*	Pointer to an initialized AVR_CPU_Config_t struct

Definition at line 227 of file avr_cpu.c.

4.5.2.4 void CPU_RegisterInterruptCallback (InterruptAck *pfIntAck_*, uint8_t *ucVector_*)

CPU_RegisterInterruptCallback.

Install a function callback to be run whenever a specific interrupt vector is run. This is useful for resetting peripheral registers once a specific type of interrupt has been acknowledged.

Parameters

*pfIntAck_*	Callback function to register
*ucVector_*	Interrupt vector index to install handler at

Definition at line 280 of file avr_cpu.c.

4.5.2.5 void CPU_RunCycle (void)

CPU_RunCycle Run a CPU instruction cycle.

This performs Fetch, Decode, Execute, Clock updates, and Interrupt handling.

Definition at line 124 of file avr_cpu.c.

4.6 avr_cpu.h

```
00001 /*************************************************************************
00002  *      (     (                        (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\      (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *************************************************************************/
00022 #ifndef __AVR_CPU_H__
00023 #define __AVR_CPU_H__
00024
00025 #include <stdint.h>
00026 #include <stdbool.h>
00027
00028 #include "emu_config.h"
00029
00030 #include "avr_peripheral.h"
00031 #include "avr_periphregs.h"
00032 #include "avr_coreregs.h"
00033 #include "avr_registerfile.h"
00034 #include "avr_io.h"
00035
00036 #include "watchpoint.h"
00037 #include "breakpoint.h"
00038
00039 //------------------------------------------------------------------------
00047 typedef struct
00048 {
00049     union
00050     {
00051         AVRRegisterFile stRegisters;
00052         uint8_t au8RAM[ sizeof(AVRRegisterFile) ];
00053     };
00054 } AVR_RAM_t;
00055
00056 //------------------------------------------------------------------------
00063 typedef struct
00064 {
00065     //--------------------------------------------------------------------
00066     // Jump tables for peripheral read/write functions.  This implementaton uses
00067     // a table with function pointer arrays, enabling multiple peripherals to
00068     // monitor reads/writes at particular addresses efficiently.
00069     //--------------------------------------------------------------------
00070     IOReaderList *apstPeriphReadTable[CONFIG_IO_ADDRESS_BYTES];
00071     IOWriterList *apstPeriphWriteTable[CONFIG_IO_ADDRESS_BYTES];
00072     IOClockList  *pstClockList;
00073
00074     //--------------------------------------------------------------------
00075     // List of data watchpoints
00076     struct _WatchPoint *pstWatchPoints;
00077
00078     //--------------------------------------------------------------------
00079     // List of instruction breakpoints
00080     struct _BreakPoint *pstBreakPoints;
00081
00082     //--------------------------------------------------------------------
00083     // Internal CPU Registers (not exposed via IO space)
```

```
00084     uint16_t      u16PC;         // Program counter is not memory mapped, unlike all others
00085
00086     //-------------------------------------------------------------------------
00087     // Emulator variables
00088     uint64_t      u64InstructionCount; // Total Executed instructions
00089     uint64_t      u64CycleCount; // Cycle Counter
00090     uint32_t      u32CoreFreq;   // CPU Frequency (Hz)
00091     uint32_t      u32WDTCount;   // Current watchdog timer count
00092     uint16_t      u16ExtraPC;    // Offset to add to the PC after executing an instruction
00093     uint16_t      u16ExtraCycles;// CPU Cycles to add for the current instruction
00094
00095     bool          bAsleep;       // Whether or not the CPU is sleeping (wake by interrupt)
00096     //-------------------------------------------------------------------------
00097     // Temporary registers used for optimizing opcodes - for various addressing modes
00098     uint16_t     *Rd16;
00099     uint8_t      *Rd; // Destination register (in some cases, also source)
00100
00101     uint16_t     *Rr16;
00102     uint8_t      *Rr; // Source register
00103
00104     uint16_t     K;   // Constant data
00105     union
00106     {
00107         uint32_t     k;   // Constant address
00108         int32_t      k_s; // Signed, constant address
00109     };
00110
00111     uint8_t      A; // IO location address
00112     uint8_t      b; // Bit in a register file (3-bits wide)
00113     uint8_t      s; // BIt in the status register (3-bits wide)
00114     uint8_t      q; // Displacement for direct addressing (6-bits)
00115
00116     //-------------------------------------------------------------------------
00117     // Setting up regions of memory for general-purpose RAM (shared with the
00118     // IO space from 0-0xFF), ROM/FLASH, and EEPROM.
00119     //-------------------------------------------------------------------------
00120     uint16_t      *pu16ROM;
00121     uint8_t       *pu8EEPROM;
00122     AVR_RAM_t     *pstRAM;
00123
00124     uint32_t     u32ROMSize;
00125     uint32_t     u32EEPROMSize;
00126     uint32_t     u32RAMSize;
00127
00128     //-------------------------------------------------------------------------
00129     uint8_t      u8IntPriority;  // Priority of pending interrupts this cycle
00130     uint32_t     u32IntFlags;    // Bitmask for the 32 interrupts
00131
00132     //-------------------------------------------------------------------------
00133     InterruptAck apfInterruptCallbacks[32]; // Interrupt callbacks
00134 } AVR_CPU;
00135
00136
00137 //-------------------------------------------------------------------------
00142 typedef struct
00143 {
00144     uint32_t u32ROMSize;
00145     uint32_t u32RAMSize;
00146     uint32_t u32EESize;
00147 } AVR_CPU_Config_t;
00148
00149 //-------------------------------------------------------------------------
00155 void CPU_Init( AVR_CPU_Config_t *pstConfig_ );
00156
00157 //-------------------------------------------------------------------------
00163 uint16_t CPU_Fetch( void );
00164
00165 //-------------------------------------------------------------------------
00171 void CPU_RunCycle( void );
00172
00173 //-------------------------------------------------------------------------
00180 void CPU_AddPeriph( AVRPeripheral *pstPeriph_ );
00181
00182 //-------------------------------------------------------------------------
00193 void CPU_RegisterInterruptCallback( InterruptAck pfIntAck_, uint8_t ucVector_
      );
00194
00195
00196 extern AVR_CPU stCPU;
00197
00198 #endif
```

4.7 avr_cpu_print.c File Reference

Helper module used to print the contents of a virtual AVR's internal registers and memory.

```
#include "avr_cpu.h"
#include "emu_config.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
```

Macros

- #define **PRINT_FUNC** printf
- #define RAM_DISPLAY_SPAN (16)

  Number of RAM values per line.
- #define ROM_DISPLAY_SPAN (8)

  Number of ROM values per line.

Functions

- void print_core_regs (void)

  print_core_regs
- void print_io_reg (uint8_t u8Addr_)

  print_io_reg
- void print_io_reg_with_name (uint8_t u8Addr_, const char ∗szName_)

  print_io_reg_with_name
- void print_ram (uint16_t u16Start_, uint16_t u16Span_)

  print_ram
- void print_rom (uint16_t u16Start_, uint16_t u16Span_)

  print_rom

4.7.1 Detailed Description

Helper module used to print the contents of a virtual AVR's internal registers and memory.

Definition in file avr_cpu_print.c.

4.7.2 Function Documentation

4.7.2.1 void print_core_regs (void)

print_core_regs

Display the contents of the CPU's core registers to the console

Definition at line 37 of file avr_cpu_print.c.

4.7.2.2 void print_io_reg (uint8_t *u8Addr_*)

print_io_reg

Display a single IO register (addresses 0-255) to the console.

Parameters

*u8Addr_*	Address of the IO register to display

Definition at line 116 of file avr_cpu_print.c.

4.7.2.3 void print_io_reg_with_name (uint8_t *u8Addr_*, const char * *szName_*)

print_io_reg_with_name

Print an IO register to the console, with a "friendly" name attached.

Parameters

*u8Addr_*	Address of the IO register to display
*szName_*	"Friendly name" of the register.

Definition at line 122 of file avr_cpu_print.c.

4.7.2.4 void print_ram (uint16_t *u16Start_*, uint16_t *u16Span_*)

print_ram

Display a block of RAM on the console.

Parameters

*u16Start_*	Start address
*u16Span_*	Number of bytes to display

Definition at line 128 of file avr_cpu_print.c.

4.7.2.5 void print_rom (uint16_t *u16Start_*, uint16_t *u16Span_*)

print_rom

Display a block of ROM to the console

Parameters

*u16Start_*	Start address
*u16Span_*	Number of instruction words (16-bit) to display

Definition at line 185 of file avr_cpu_print.c.

4.8 avr_cpu_print.c

```
00001 /*****************************************************************************
00002  *      (     (                      (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\     (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+-----------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00022 #include "avr_cpu.h"
00023
00024 #include "emu_config.h"
00025
00026 #include <stdio.h>
00027 #include <stdlib.h>
```

```
00028 #include <stdint.h>
00029
00030 //------------------------------------------------------------------------
00031 #define PRINT_FUNC      printf
00032
00033 #define RAM_DISPLAY_SPAN        (16)
00034 #define ROM_DISPLAY_SPAN        (8)
00035
00036 //------------------------------------------------------------------------
00037 void print_core_regs( void )
00038 {
00039     uint8_t i;
00040     for (i = 0; i < 32; i++)
00041     {
00042         PRINT_FUNC( "[R%02d] = 0x%02X\n", i, stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[i] );
00043     }
00044     PRINT_FUNC("[SP]  = 0x%02X%02X\n", (uint8_t)stCPU.pstRAM->stRegisters.SPH.r, (uint8_t)stCPU.pstRAM->
      stRegisters.SPL.r );
00045     PRINT_FUNC("[PC]  = 0x%04X\n", (uint16_t)stCPU.u16PC );
00046     PRINT_FUNC("[SREG]= 0x%02X   [", stCPU.pstRAM->stRegisters.SREG.r );
00047
00048     if (1 == stCPU.pstRAM->stRegisters.SREG.I)
00049     {
00050         PRINT_FUNC("I");
00051     }
00052     else
00053     {
00054         PRINT_FUNC("-");
00055     }
00056     if (1 == stCPU.pstRAM->stRegisters.SREG.T)
00057     {
00058         PRINT_FUNC("T");
00059     }
00060     else
00061     {
00062         PRINT_FUNC("-");
00063     }
00064     if (1 == stCPU.pstRAM->stRegisters.SREG.H)
00065     {
00066         PRINT_FUNC("H");
00067     }
00068     else
00069     {
00070         PRINT_FUNC("-");
00071     }
00072     if (1 == stCPU.pstRAM->stRegisters.SREG.S)
00073     {
00074         PRINT_FUNC("S");
00075     }
00076     else
00077     {
00078         PRINT_FUNC("-");
00079     }
00080     if (1 == stCPU.pstRAM->stRegisters.SREG.V)
00081     {
00082         PRINT_FUNC("V");
00083     }
00084     else
00085     {
00086         PRINT_FUNC("-");
00087     }
00088     if (1 == stCPU.pstRAM->stRegisters.SREG.N)
00089     {
00090         PRINT_FUNC("N");
00091     }
00092     else
00093     {
00094         PRINT_FUNC("-");
00095     }
00096     if (1 == stCPU.pstRAM->stRegisters.SREG.Z)
00097     {
00098         PRINT_FUNC("Z");
00099     }
00100     else
00101     {
00102         PRINT_FUNC("-");
00103     }
00104     if (1 == stCPU.pstRAM->stRegisters.SREG.C)
00105     {
00106         PRINT_FUNC("C");
00107     }
00108     else
00109     {
00110         PRINT_FUNC("-");
00111     }
00112     PRINT_FUNC("]\n");
00113 }
```

```
00114
00115 //-----------------------------------------------------------------------------
00116 void print_io_reg( uint8_t u8Addr_ )
00117 {
00118     PRINT_FUNC( "[IO%02X]= 0x%02X\n", u8Addr_, stCPU.pstRAM->au8RAM[u8Addr_] );
00119 }
00120
00121 //-----------------------------------------------------------------------------
00122 void print_io_reg_with_name( uint8_t u8Addr_, const char *szName_ )
00123 {
00124     PRINT_FUNC( "[%s]= 0x%02X\n", szName_, stCPU.pstRAM->au8RAM[u8Addr_] );
00125 }
00126
00127 //-----------------------------------------------------------------------------
00128 void print_ram( uint16_t u16Start_, uint16_t u16Span_ )
00129 {
00130     uint16_t i, j;
00131
00132     while (u16Span_)
00133     {
00134         // Print the current memory address
00135         PRINT_FUNC( "[0x%04X]", u16Start_ );
00136         if (u16Span_ < RAM_DISPLAY_SPAN)
00137         {
00138             j = u16Span_;
00139         }
00140         else
00141         {
00142             j = RAM_DISPLAY_SPAN;
00143         }
00144
00145         // Print a divider, followed by the ASCII codes for each char
00146         PRINT_FUNC( "|" );
00147         for (i = 0; i < j; i++)
00148         {
00149             uint8_t u8Char = stCPU.pstRAM->au8RAM[u16Start_ + i];
00150             if (u8Char < 32)
00151             {
00152                 u8Char = '.';
00153             }
00154
00155             PRINT_FUNC( " %c", u8Char );
00156         }
00157         i = j;
00158         while (i < RAM_DISPLAY_SPAN)
00159         {
00160             PRINT_FUNC("  ");
00161             i++;
00162         }
00163
00164         // Print a divider, followed by the HEX code for each char
00165         PRINT_FUNC( "|" );
00166         for (i = 0; i < j; i++)
00167         {
00168             PRINT_FUNC( " %02X", stCPU.pstRAM->au8RAM[u16Start_ + i]);
00169         }
00170
00171         if (u16Span_ < RAM_DISPLAY_SPAN)
00172         {
00173             u16Span_ = 0;
00174         }
00175         else
00176         {
00177             u16Span_ -= RAM_DISPLAY_SPAN;
00178         }
00179         u16Start_ += RAM_DISPLAY_SPAN;
00180         PRINT_FUNC( "\n" );
00181     }
00182 }
00183
00184 //-----------------------------------------------------------------------------
00185 void print_rom( uint16_t u16Start_, uint16_t u16Span_ )
00186 {
00187     uint16_t i, j;
00188
00189     while (u16Span_)
00190     {
00191         // Print the current memory address
00192         PRINT_FUNC( "[0x%04X]", u16Start_ );
00193         if (u16Span_ < ROM_DISPLAY_SPAN)
00194         {
00195             j = u16Span_;
00196         }
00197         else
00198         {
00199             j = ROM_DISPLAY_SPAN;
00200         }
```

```
00201
00202         // Print a divider, followed by the ASCII codes for each char
00203         PRINT_FUNC( "|" );
00204         for (i = 0; i < j; i++)
00205         {
00206             uint16_t u16Val = stCPU.pu16ROM[u16Start_ + i];
00207             uint8_t u8High = u16Val >> 8;
00208             uint8_t u8Low = u16Val & 0x00FF;
00209
00210             if (u8High < 32)
00211             {
00212                 u8High = '.';
00213             }
00214             if (u8Low < 32)
00215             {
00216                 u8Low = '.';
00217             }
00218
00219             PRINT_FUNC( " %c%c", u8High, u8Low );
00220         }
00221         i = j;
00222         while (i < ROM_DISPLAY_SPAN)
00223         {
00224             PRINT_FUNC("   ");
00225             i++;
00226         }
00227
00228         // Print a divider, followed by the HEX code for each char
00229         PRINT_FUNC( "|" );
00230         for (i = 0; i < j; i++)
00231         {
00232             PRINT_FUNC( " %04X", stCPU.pu16ROM[u16Start_ + i]);
00233         }
00234
00235         if (u16Span_ < ROM_DISPLAY_SPAN)
00236         {
00237             u16Span_ = 0;
00238         }
00239         else
00240         {
00241             u16Span_ -= ROM_DISPLAY_SPAN;
00242         }
00243         u16Start_ += ROM_DISPLAY_SPAN;
00244         PRINT_FUNC( "\n" );
00245     }
00246 }
```

4.9 avr_cpu_print.h File Reference

Helper module used to print the contents of a virtual AVR's internal registers and memory.

```
#include <stdint.h>
#include "avr_cpu.h"
```

Functions

- void print_core_regs (void)

  print_core_regs
- void print_io_reg (uint8_t u8Addr_)

  print_io_reg
- void print_io_reg_with_name (uint8_t u8Addr_, const char *szName_)

  print_io_reg_with_name
- void print_ram (uint16_t u16Start_, uint16_t u16Span_)

  print_ram
- void print_rom (uint16_t u16Start_, uint16_t u16Span_)

  print_rom

4.9.1 Detailed Description

Helper module used to print the contents of a virtual AVR's internal registers and memory.

Definition in file avr_cpu_print.h.

4.9.2 Function Documentation

4.9.2.1 void print_core_regs (void)

print_core_regs

Display the contents of the CPU's core registers to the console

Definition at line 37 of file avr_cpu_print.c.

4.9.2.2 void print_io_reg (uint8_t *u8Addr_*)

print_io_reg

Display a single IO register (addresses 0-255) to the console.

Parameters

*u8Addr_*	Address of the IO register to display

Definition at line 116 of file avr_cpu_print.c.

4.9.2.3 void print_io_reg_with_name (uint8_t *u8Addr_*, const char * *szName_*)

print_io_reg_with_name

Print an IO register to the console, with a "friendly" name attached.

Parameters

*u8Addr_*	Address of the IO register to display
*szName_*	"Friendly name" of the register.

Definition at line 122 of file avr_cpu_print.c.

4.9.2.4 void print_ram (uint16_t *u16Start_*, uint16_t *u16Span_*)

print_ram

Display a block of RAM on the console.

Parameters

*u16Start_*	Start address
*u16Span_*	Number of bytes to display

Definition at line 128 of file avr_cpu_print.c.

4.9.2.5 void print_rom (uint16_t *u16Start_*, uint16_t *u16Span_*)

print_rom

Display a block of ROM to the console

Parameters

*u16Start_*	Start address
*u16Span_*	Number of instruction words (16-bit) to display

Definition at line 185 of file avr_cpu_print.c.

4.10 avr_cpu_print.h

```
00001 /*****************************************************************************
00002  *      (     (                        (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\     (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\   |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+-----------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00023 #ifndef __AVR_CPU_PRINT_H__
00024 #define __AVR_CPU_PRINT_H__
00025
00026 #include <stdint.h>
00027 #include "avr_cpu.h"
00028
00029 //-----------------------------------------------------------------------------
00035 void print_core_regs( void );
00036
00037 //-----------------------------------------------------------------------------
00045 void print_io_reg( uint8_t u8Addr_ );
00046
00047 //-----------------------------------------------------------------------------
00057 void print_io_reg_with_name( uint8_t u8Addr_, const char *szName_ );
00058
00059 //-----------------------------------------------------------------------------
00068 void print_ram( uint16_t u16Start_, uint16_t u16Span_ );
00069
00070 //-----------------------------------------------------------------------------
00079 void print_rom( uint16_t u16Start_, uint16_t u16Span_ );
00080
00081 #endif
```

4.11 avr_disasm.c File Reference

AVR Disassembler Implementation.

```
#include <stdint.h>
#include <stdio.h>
#include "emu_config.h"
#include "avr_op_decode.h"
#include "avr_opcodes.h"
#include "avr_op_size.h"
#include "avr_cpu.h"
#include "avr_cpu_print.h"
#include "avr_loader.h"
```

Functions

- int8_t **Signed_From_Unsigned_6** (uint8_t u8Signed_)
- uint8_t **Register_From_Rd** (void)
- uint8_t **Register_From_Rr** (void)
- uint8_t **Register_From_Rd16** (void)

- uint8_t **Register_From_Rr16** (void)
- static void **AVR_Disasm_ADD** (void)
- static void **AVR_Disasm_ADC** (void)
- static void **AVR_Disasm_ADIW** (void)
- static void **AVR_Disasm_SUB** (void)
- static void **AVR_Disasm_SUBI** (void)
- static void **AVR_Disasm_SBC** (void)
- static void **AVR_Disasm_SBCI** (void)
- static void **AVR_Disasm_SBIW** (void)
- static void **AVR_Disasm_AND** (void)
- static void **AVR_Disasm_ANDI** (void)
- static void **AVR_Disasm_OR** (void)
- static void **AVR_Disasm_ORI** (void)
- static void **AVR_Disasm_EOR** (void)
- static void **AVR_Disasm_COM** (void)
- static void **AVR_Disasm_NEG** (void)
- static void **AVR_Disasm_SBR** (void)
- static void **AVR_Disasm_CBR** (void)
- static void **AVR_Disasm_INC** (void)
- static void **AVR_Disasm_DEC** (void)
- static void **AVR_Disasm_TST** (void)
- static void **AVR_Disasm_CLR** (void)
- static void **AVR_Disasm_SER** (void)
- static void **AVR_Disasm_MUL** (void)
- static void **AVR_Disasm_MULS** (void)
- static void **AVR_Disasm_MULSU** (void)
- static void **AVR_Disasm_FMUL** (void)
- static void **AVR_Disasm_FMULS** (void)
- static void **AVR_Disasm_FMULSU** (void)
- static void **AVR_Disasm_DES** (void)
- static void **AVR_Disasm_RJMP** (void)
- static void **AVR_Disasm_IJMP** (void)
- static void **AVR_Disasm_EIJMP** (void)
- static void **AVR_Disasm_JMP** (void)
- static void **AVR_Disasm_RCALL** (void)
- static void **AVR_Disasm_ICALL** (void)
- static void **AVR_Disasm_EICALL** (void)
- static void **AVR_Disasm_CALL** (void)
- static void **AVR_Disasm_RET** (void)
- static void **AVR_Disasm_RETI** (void)
- static void **AVR_Disasm_CPSE** (void)
- static void **AVR_Disasm_CP** (void)
- static void **AVR_Disasm_CPC** (void)
- static void **AVR_Disasm_CPI** (void)
- static void **AVR_Disasm_SBRC** (void)
- static void **AVR_Disasm_SBRS** (void)
- static void **AVR_Disasm_SBIC** (void)
- static void **AVR_Disasm_SBIS** (void)
- static void **AVR_Disasm_BRBS** (void)
- static void **AVR_Disasm_BRBC** (void)
- static void **AVR_Disasm_BREQ** (void)
- static void **AVR_Disasm_BRNE** (void)
- static void **AVR_Disasm_BRCS** (void)
- static void **AVR_Disasm_BRCC** (void)
- static void **AVR_Disasm_BRSH** (void)

- static void **AVR_Disasm_BRLO** (void)
- static void **AVR_Disasm_BRMI** (void)
- static void **AVR_Disasm_BRPL** (void)
- static void **AVR_Disasm_BRGE** (void)
- static void **AVR_Disasm_BRLT** (void)
- static void **AVR_Disasm_BRHS** (void)
- static void **AVR_Disasm_BRHC** (void)
- static void **AVR_Disasm_BRTS** (void)
- static void **AVR_Disasm_BRTC** (void)
- static void **AVR_Disasm_BRVS** (void)
- static void **AVR_Disasm_BRVC** (void)
- static void **AVR_Disasm_BRIE** (void)
- static void **AVR_Disasm_BRID** (void)
- static void **AVR_Disasm_MOV** (void)
- static void **AVR_Disasm_MOVW** (void)
- static void **AVR_Disasm_LDI** (void)
- static void **AVR_Disasm_LDS** (void)
- static void **AVR_Disasm_LD_X_Indirect** (void)
- static void **AVR_Disasm_LD_X_Indirect_Postinc** (void)
- static void **AVR_Disasm_LD_X_Indirect_Predec** (void)
- static void **AVR_Disasm_LD_Y_Indirect** (void)
- static void **AVR_Disasm_LD_Y_Indirect_Postinc** (void)
- static void **AVR_Disasm_LD_Y_Indirect_Predec** (void)
- static void **AVR_Disasm_LDD_Y** (void)
- static void **AVR_Disasm_LD_Z_Indirect** (void)
- static void **AVR_Disasm_LD_Z_Indirect_Postinc** (void)
- static void **AVR_Disasm_LD_Z_Indirect_Predec** (void)
- static void **AVR_Disasm_LDD_Z** (void)
- static void **AVR_Disasm_STS** (void)
- static void **AVR_Disasm_ST_X_Indirect** (void)
- static void **AVR_Disasm_ST_X_Indirect_Postinc** (void)
- static void **AVR_Disasm_ST_X_Indirect_Predec** (void)
- static void **AVR_Disasm_ST_Y_Indirect** (void)
- static void **AVR_Disasm_ST_Y_Indirect_Postinc** (void)
- static void **AVR_Disasm_ST_Y_Indirect_Predec** (void)
- static void **AVR_Disasm_STD_Y** (void)
- static void **AVR_Disasm_ST_Z_Indirect** (void)
- static void **AVR_Disasm_ST_Z_Indirect_Postinc** (void)
- static void **AVR_Disasm_ST_Z_Indirect_Predec** (void)
- static void **AVR_Disasm_STD_Z** (void)
- static void **AVR_Disasm_LPM** (void)
- static void **AVR_Disasm_LPM_Z** (void)
- static void **AVR_Disasm_LPM_Z_Postinc** (void)
- static void **AVR_Disasm_ELPM** (void)
- static void **AVR_Disasm_ELPM_Z** (void)
- static void **AVR_Disasm_ELPM_Z_Postinc** (void)
- static void **AVR_Disasm_SPM** (void)
- static void **AVR_Disasm_SPM_Z_Postinc2** (void)
- static void **AVR_Disasm_IN** (void)
- static void **AVR_Disasm_OUT** (void)
- static void **AVR_Disasm_LAC** (void)
- static void **AVR_Disasm_LAS** (void)
- static void **AVR_Disasm_LAT** (void)
- static void **AVR_Disasm_LSL** (void)
- static void **AVR_Disasm_LSR** (void)

- static void **AVR_Disasm_POP** (void)
- static void **AVR_Disasm_PUSH** (void)
- static void **AVR_Disasm_ROL** (void)
- static void **AVR_Disasm_ROR** (void)
- static void **AVR_Disasm_ASR** (void)
- static void **AVR_Disasm_SWAP** (void)
- static void **AVR_Disasm_BSET** (void)
- static void **AVR_Disasm_BCLR** (void)
- static void **AVR_Disasm_SBI** (void)
- static void **AVR_Disasm_CBI** (void)
- static void **AVR_Disasm_BST** (void)
- static void **AVR_Disasm_BLD** (void)
- static void **AVR_Disasm_SEC** (void)
- static void **AVR_Disasm_CLC** (void)
- static void **AVR_Disasm_SEN** (void)
- static void **AVR_Disasm_CLN** (void)
- static void **AVR_Disasm_SEZ** (void)
- static void **AVR_Disasm_CLZ** (void)
- static void **AVR_Disasm_SEI** (void)
- static void **AVR_Disasm_CLI** (void)
- static void **AVR_Disasm_SES** (void)
- static void **AVR_Disasm_CLS** (void)
- static void **AVR_Disasm_SEV** (void)
- static void **AVR_Disasm_CLV** (void)
- static void **AVR_Disasm_SET** (void)
- static void **AVR_Disasm_CLT** (void)
- static void **AVR_Disasm_SEH** (void)
- static void **AVR_Disasm_CLH** (void)
- static void **AVR_Disasm_BREAK** (void)
- static void **AVR_Disasm_NOP** (void)
- static void **AVR_Disasm_SLEEP** (void)
- static void **AVR_Disasm_WDR** (void)
- static void **AVR_Disasm_XCH** (void)
- static void **AVR_Disasm_Unimplemented** ()
- AVR_Opcode AVR_Disasm_Function (uint16_t OP_)

  AVR_Disasm_Function.

4.11.1 Detailed Description

AVR Disassembler Implementation.

Definition in file avr_disasm.c.

4.11.2 Function Documentation

4.11.2.1 AVR_Opcode AVR_Disasm_Function (uint16_t *OP_*)

AVR_Disasm_Function.

Return a function pointer to a disassembly routine corresponding to a given opcode.

Parameters

*OP_*	Opcode to disasemble

Returns

Function pointer that, when called with a valid CPU object and opcode, will produce a valid disassembly statement to standard output.

Definition at line 1635 of file avr_disasm.c.

4.12 avr_disasm.c

```
00001 /*****************************************************************************
00002  *      (      (                       (      |
00003  *    )\ )   )\ )     (               )\ )    |
00004  *   (()/(  (()/(     )\      (    (  (()/(    | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\   )\  /(_))       | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))       | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \      | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \    \ V /  |   /     | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\    \_/   |_|_\     |
00010  *                                           | "Yeah, it does Arduino..."
00011  * ------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #include <stdint.h>
00022 #include <stdio.h>
00023
00024 #include "emu_config.h"
00025
00026 #include "avr_op_decode.h"
00027 #include "avr_opcodes.h"
00028 #include "avr_op_size.h"
00029 #include "avr_cpu.h"
00030 #include "avr_cpu_print.h"
00031 #include "avr_loader.h"
00032
00033 inline int8_t Signed_From_Unsigned_6( uint8_t u8Signed_ )
00034 {
00035     int8_t i8Ret = 0;
00036     if( u8Signed_ & 0x20 )
00037     {
00038         //Sign extend...
00039         i8Ret = (int8_t)(u8Signed_ | 0xC0);
00040     }
00041     else
00042     {
00043         i8Ret = (int8_t)u8Signed_;
00044     }
00045     return i8Ret;
00046 }
00047
00048 //-------------------------------------------------------------------------
00049 inline uint8_t Register_From_Rd( void )
00050 {
00051     return stCPU.Rd - &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0);
00052 }
00053  //-------------------------------------------------------------------------
00054 inline uint8_t Register_From_Rr( void )
00055 {
00056     return stCPU.Rr - &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0);
00057 }
00058
00059 //-------------------------------------------------------------------------
00060 inline uint8_t Register_From_Rd16( void )
00061 {
00062     return (uint8_t*)(stCPU.Rd16) - &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0);
00063 }
00064
00065 //-------------------------------------------------------------------------
00066 inline uint8_t Register_From_Rr16( void )
00067 {
00068     return (uint8_t*)(stCPU.Rr16) - &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0);
00069 }
00070
00071 //-------------------------------------------------------------------------
00072 static void AVR_Disasm_ADD( void )
```

```
00073 {
00074     uint8_t u8Rd = Register_From_Rd();
00075     uint8_t u8Rr = Register_From_Rr();
00076 
00077     //ruler: 0----5----10---15---20---25---30---35---40" );
00078     printf( "add r%d, r%d              \t ; Add: r%d = r%d + r%d\n",
00079                 u8Rd, u8Rr,
00080                 u8Rd, u8Rd, u8Rr );
00081 }
00082 
00083 //-------------------------------------------------------------------------
00084 static void AVR_Disasm_ADC( void )
00085 {
00086     uint8_t u8Rd = Register_From_Rd();
00087     uint8_t u8Rr = Register_From_Rr();
00088 
00089     //ruler: 0----5----10---15---20---25---30---35---40" );
00090     printf( "adc r%d, r%d              \t ; Add with carry: r%d = r%d + r%d + C\n",
00091                 u8Rd, u8Rr,
00092                 u8Rd, u8Rd, u8Rr );
00093 
00094 }
00095 
00096 //-------------------------------------------------------------------------
00097 static void AVR_Disasm_ADIW( void )
00098 {
00099     uint8_t u8Rd = Register_From_Rd16();
00100     uint8_t u8K = stCPU.K;
00101 
00102     //ruler: 0----5----10---15---20---25---30---35---40" );
00103     printf( "adiw r%d:%d, %d           \t ; Add immediate to word: r%d:%d = r%d:%d + %d \n",
00104                 u8Rd + 1, u8Rd, u8K,
00105                 u8Rd + 1, u8Rd, u8Rd + 1, u8Rd, u8K
00106                 );
00107 }
00108 
00109 //-------------------------------------------------------------------------
00110 static void AVR_Disasm_SUB( void )
00111 {
00112     uint8_t u8Rd = Register_From_Rd();
00113     uint8_t u8Rr = Register_From_Rr();
00114 
00115     //ruler: 0----5----10---15---20---25---30---35---40" );
00116     printf( "sub r%d, r%d              \t ; Subtract: r%d = r%d - r%d \n",
00117                 u8Rd, u8Rr,
00118                 u8Rd, u8Rd, u8Rr
00119                 );
00120 }
00121 
00122 //-------------------------------------------------------------------------
00123 static void AVR_Disasm_SUBI( void )
00124 {
00125     uint8_t u8Rd = Register_From_Rd();
00126     uint8_t u8K = stCPU.K;
00127 
00128     //ruler: 0----5----10---15---20---25---30---35---40" );
00129     printf( "subi r%d, %d              \t ; Subtract immediate: r%d = r%d - %d \n",
00130                 u8Rd, u8K,
00131                 u8Rd, u8Rd, u8K
00132                 );
00133 }
00134 
00135 //-------------------------------------------------------------------------
00136 static void AVR_Disasm_SBC( void )
00137 {
00138     uint8_t u8Rd = Register_From_Rd();
00139     uint8_t u8Rr = Register_From_Rr();
00140 
00141     //ruler: 0----5----10---15---20---25---30---35---40" );
00142     printf( "sbc r%d, r%d              \t ; Subtract with carry: r%d = r%d - r%d - C \n",
00143                 u8Rd, u8Rr,
00144                 u8Rd, u8Rd, u8Rr
00145                 );
00146 }
00147 
00148 //-------------------------------------------------------------------------
00149 static void AVR_Disasm_SBCI( void )
00150 {
00151     uint8_t u8Rd = Register_From_Rd();
00152     uint8_t u8K = stCPU.K;
00153 
00154     //ruler: 0----5----10---15---20---25---30---35---40" );
00155     printf( "sbci r%d, %d              \t ; Subtract immediate with carry: r%d = r%d - %d - C\n",
00156                 u8Rd, u8K,
00157                 u8Rd, u8Rd, u8K
00158                 );
00159 }
```

```
00160
00161 //-------------------------------------------------------------------------
00162 static void AVR_Disasm_SBIW( void )
00163 {
00164     uint8_t u8Rd = Register_From_Rd16();
00165     uint8_t u8K = stCPU.K;
00166
00167     //ruler: 0----5----10---15---20---25---30---35---40" );
00168     printf( "sbiw r%d:%d, %d           \t ; Subtract immediate from word: r%d:%d = r%d:%d + %d \n",
00169                 u8Rd + 1, u8Rd, u8K,
00170                 u8Rd + 1, u8Rd, u8Rd + 1, u8Rd, u8K
00171                 );
00172 }
00173
00174 //-------------------------------------------------------------------------
00175 static void AVR_Disasm_AND( void )
00176 {
00177     uint8_t u8Rd = Register_From_Rd();
00178     uint8_t u8Rr = Register_From_Rr();
00179
00180     //ruler: 0----5----10---15---20---25---30---35---40" );
00181     printf( "and r%d, r%d              \t ; Logical AND: r%d = r%d & r%d \n",
00182                 u8Rd, u8Rr,
00183                 u8Rd, u8Rd, u8Rr
00184                 );
00185 }
00186
00187 //-------------------------------------------------------------------------
00188 static void AVR_Disasm_ANDI( void )
00189 {
00190     uint8_t u8Rd = Register_From_Rd();
00191     uint8_t u8K = stCPU.K;
00192
00193     //ruler: 0----5----10---15---20---25---30---35---40" );
00194     printf( "andi r%d, %d             \t ; Logical AND with Immediate: r%d = r%d & %d\n",
00195                 u8Rd, u8K,
00196                 u8Rd, u8Rd, u8K
00197                 );
00198 }
00199
00200 //-------------------------------------------------------------------------
00201 static void AVR_Disasm_OR( void )
00202 {
00203     uint8_t u8Rd = Register_From_Rd();
00204     uint8_t u8Rr = Register_From_Rr();
00205
00206     //ruler: 0----5----10---15---20---25---30---35---40" );
00207     printf( "or r%d, r%d              \t ; Logical OR: r%d = r%d | r%d \n",
00208                 u8Rd, u8Rr,
00209                 u8Rd, u8Rd, u8Rr
00210                 );
00211 }
00212
00213 //-------------------------------------------------------------------------
00214 static void AVR_Disasm_ORI( void )
00215 {
00216     uint8_t u8Rd = Register_From_Rd();
00217     uint8_t u8K = stCPU.K;
00218
00219     //ruler: 0----5----10---15---20---25---30---35---40" );
00220     printf( "ori r%d, %d              \t ; Logical OR with Immediate: r%d = r%d | %d\n",
00221                 u8Rd, u8K,
00222                 u8Rd, u8Rd, u8K
00223                 );
00224 }
00225
00226 //-------------------------------------------------------------------------
00227 static void AVR_Disasm_EOR( void )
00228 {
00229     uint8_t u8Rd = Register_From_Rd();
00230     uint8_t u8Rr = Register_From_Rr();
00231
00232     //ruler: 0----5----10---15---20---25---30---35---40" );
00233     printf( "eor r%d, r%d             \t ; Exclusive OR: r%d = r%d ^ r%d \n",
00234                 u8Rd, u8Rr,
00235                 u8Rd, u8Rd, u8Rr
00236                 );
00237 }
00238
00239 //-------------------------------------------------------------------------
00240 static void AVR_Disasm_COM( void )
00241 {
00242     uint8_t u8Rd = Register_From_Rd();
00243
00244     //ruler: 0----5----10---15---20---25---30---35---40" );
00245     printf( "com r%d                  \t ; One's complement (bitwise inverse): r%d = 0xFF - r%d\n",
00246                 u8Rd,
```

```
00247                 u8Rd, u8Rd
00248                 );
00249 }
00250
00251 //-------------------------------------------------------------------------
00252 static void AVR_Disasm_NEG( void )
00253 {
00254     uint8_t u8Rd = Register_From_Rd();
00255
00256     //ruler: 0----5----10---15---20---25---30---35---40" );
00257     printf( "neg r%d                  \t ; Two's complement (sign swap): r%d = 0x00 - r%d\n",
00258                 u8Rd,
00259                 u8Rd, u8Rd
00260                 );
00261 }
00262
00263 //-------------------------------------------------------------------------
00264 static void AVR_Disasm_SBR( void )
00265 {
00266     uint8_t u8Rd = Register_From_Rd();
00267     uint8_t u8K = stCPU.K;
00268
00269     //ruler: 0----5----10---15---20---25---30---35---40" );
00270     printf( "sbr r%d, %d              \t ; Set Bits in Register: r%d = r%d | %d\n",
00271                 u8Rd, u8K,
00272                 u8Rd, u8Rd, u8K
00273                 );
00274 }
00275
00276 //-------------------------------------------------------------------------
00277 static void AVR_Disasm_CBR( void )
00278 {
00279     uint8_t u8Rd = Register_From_Rd();
00280     uint8_t u8K = stCPU.K;
00281
00282     //ruler: 0----5----10---15---20---25---30---35---40" );
00283     printf( "cbr r%d, %d              \t ; Clear Bits in Register: r%d = r%d & (0xFF - %d)\n",
00284                 u8Rd, u8K,
00285                 u8Rd, u8Rd, u8K
00286                 );
00287 }
00288
00289 //-------------------------------------------------------------------------
00290 static void AVR_Disasm_INC( void )
00291 {
00292     uint8_t u8Rd = Register_From_Rd();
00293
00294     //ruler: 0----5----10---15---20---25---30---35---40" );
00295     printf( "inc r%d                  \t ; Increment Register: r%d = r%d + 1\n",
00296                 u8Rd,
00297                 u8Rd, u8Rd
00298                 );
00299 }
00300
00301 //-------------------------------------------------------------------------
00302 static void AVR_Disasm_DEC( void )
00303 {
00304     uint8_t u8Rd = Register_From_Rd();
00305
00306     //ruler: 0----5----10---15---20---25---30---35---40" );
00307     printf( "dec r%d                  \t ; Decrement Register: r%d = r%d - 1\n",
00308                 u8Rd,
00309                 u8Rd, u8Rd
00310                 );
00311 }
00312
00313 //-------------------------------------------------------------------------
00314 static void AVR_Disasm_TST( void )
00315 {
00316     uint8_t u8Rd = Register_From_Rd();
00317
00318     //ruler: 0----5----10---15---20---25---30---35---40" );
00319     printf( "tst r%d                  \t ; Test Register for Zero or Negative\n",
00320                 u8Rd
00321                 );
00322 }
00323
00324 //-------------------------------------------------------------------------
00325 static void AVR_Disasm_CLR( void )
00326 {
00327     uint8_t u8Rd = Register_From_Rd();
00328
00329     //ruler: 0----5----10---15---20---25---30---35---40" );
00330     printf( "clr r%d                  \t ; Clear Register\n",
00331                 u8Rd
00332                 );
00333 }
```

```
00334
00335 //-------------------------------------------------------------------------
00336 static void AVR_Disasm_SER( void )
00337 {
00338     uint8_t u8Rd = Register_From_Rd();
00339
00340     //ruler: 0----5----10---15---20---25---30---35---40" );
00341     printf( "ser r%d                  \t ; Set All Bits in Register\n",
00342                 u8Rd
00343                 );
00344 }
00345
00346 //-------------------------------------------------------------------------
00347 static void AVR_Disasm_MUL( void )
00348 {
00349     uint8_t u8Rd = Register_From_Rd();
00350     uint8_t u8Rr = Register_From_Rr();
00351
00352     //ruler: 0----5----10---15---20---25---30---35---40" );
00353     printf( "mul r%d, r%d             \t ; Unsigned Multiply: r1:0 = r%d * r%d\n",
00354                 u8Rd, u8Rr,
00355                 u8Rd, u8Rr );
00356 }
00357
00358 //-------------------------------------------------------------------------
00359 static void AVR_Disasm_MULS( void )
00360 {
00361     uint8_t u8Rd = Register_From_Rd();
00362     uint8_t u8Rr = Register_From_Rr();
00363
00364     //ruler: 0----5----10---15---20---25---30---35---40" );
00365     printf( "muls r%d, r%d            \t ; Signed Multiply: r1:0 = r%d * r%d\n",
00366                 u8Rd, u8Rr,
00367                 u8Rd, u8Rr );
00368 }
00369
00370 //-------------------------------------------------------------------------
00371 static void AVR_Disasm_MULSU( void )
00372 {
00373     uint8_t u8Rd = Register_From_Rd();
00374     uint8_t u8Rr = Register_From_Rr();
00375
00376     //ruler: 0----5----10---15---20---25---30---35---40" );
00377     printf( "mulsu r%d, r%d           \t ; Signed * Unsigned Multiply: r1:0 = r%d * r%d\n",
00378                 u8Rd, u8Rr,
00379                 u8Rd, u8Rr );
00380 }
00381
00382 //-------------------------------------------------------------------------
00383 static void AVR_Disasm_FMUL( void )
00384 {
00385     uint8_t u8Rd = Register_From_Rd();
00386     uint8_t u8Rr = Register_From_Rr();
00387
00388     //ruler: 0----5----10---15---20---25---30---35---40" );
00389     printf( "fmul r%d, r%d            \t ; Fractional Multiply: r1:0 = r%d * r%d\n",
00390                 u8Rd, u8Rr,
00391                 u8Rd, u8Rr );
00392 }
00393
00394 //-------------------------------------------------------------------------
00395 static void AVR_Disasm_FMULS( void )
00396 {
00397     uint8_t u8Rd = Register_From_Rd();
00398     uint8_t u8Rr = Register_From_Rr();
00399
00400     //ruler: 0----5----10---15---20---25---30---35---40" );
00401     printf( "fmuls r%d, r%d           \t ; Signed Fractional Multiply: r1:0 = r%d * r%d\n",
00402                 u8Rd, u8Rr,
00403                 u8Rd, u8Rr );
00404
00405 }
00406
00407 //-------------------------------------------------------------------------
00408 static void AVR_Disasm_FMULSU( void )
00409 {
00410     uint8_t u8Rd = Register_From_Rd();
00411     uint8_t u8Rr = Register_From_Rr();
00412
00413     //ruler: 0----5----10---15---20---25---30---35---40" );
00414     printf( "fmulsu r%d, r%d          \t ; Signed * Unsigned Fractional Multiply: r1:0 = r%d * r%d\n",
00415                 u8Rd, u8Rr,
00416                 u8Rd, u8Rr );
00417 }
00418
00419 //-------------------------------------------------------------------------
00420 static void AVR_Disasm_DES( void )
```

```
00421 {
00422     uint8_t u8K = stCPU.K;
00423 
00424     //ruler: 0----5----10---15---20---25---30---35---40" );
00425     printf( "des %d                   \t ; DES Encrypt/Decrypt\n",
00426             u8K );
00427 }
00428 
00429 //-----------------------------------------------------------------------
00430 static void AVR_Disasm_RJMP( void )
00431 {
00432     int16_t i16k = stCPU.k_s;
00433 
00434     //ruler: 0----5----10---15---20---25---30---35---40" );
00435     printf( "rjmp %d                  \t ; Relative Jump: PC = PC + %d + 1\n",
00436                 i16k, i16k );
00437 }
00438 
00439 //-----------------------------------------------------------------------
00440 static void AVR_Disasm_IJMP( void )
00441 {
00442     //ruler: 0----5----10---15---20---25---30---35---40" );
00443     printf( "ijmp                     \t ; Indirect Jump: PC = Z\n");
00444 }
00445 
00446 //-----------------------------------------------------------------------
00447 static void AVR_Disasm_EIJMP( void )
00448 {
00449     //ruler: 0----5----10---15---20---25---30---35---40" );
00450     printf( "eijmp                    \t ; Extended Indirect Jump: PC(15:0) = Z(15:0), PC(21:16) = EIND\n"
      );
00451 }
00452 
00453 //-----------------------------------------------------------------------
00454 static void AVR_Disasm_JMP( void )
00455 {
00456     uint32_t u32k = stCPU.k;
00457 
00458     //ruler: 0----5----10---15---20---25---30---35---40" );
00459     printf( "jmp 0x%X                    \t ; Jump to 0x%X \n",
00460                 u32k, u32k );
00461 }
00462 
00463 //-----------------------------------------------------------------------
00464 static void AVR_Disasm_RCALL( void )
00465 {
00466     int16_t i16k = stCPU.k_s;
00467 
00468     //ruler: 0----5----10---15---20---25---30---35---40" );
00469     printf( "rcall %d                 \t ; Relative call to Subroutine: PC = PC +%d + 1\n",
00470                 i16k, i16k
00471                 );
00472 }
00473 
00474 //-----------------------------------------------------------------------
00475 static void AVR_Disasm_ICALL( void )
00476 {
00477     //ruler: 0----5----10---15---20---25---30---35---40" );
00478     printf( "icall                    \t ; Indirect Jump: PC = Z\n");
00479 }
00480 
00481 //-----------------------------------------------------------------------
00482 static void AVR_Disasm_EICALL( void )
00483 {
00484     //ruler: 0----5----10---15---20---25---30---35---40" );
00485     printf( "eicall                   \t ; Extended Indirect Jump: PC(15:0) = Z(15:0), PC(21:16) = EIND\n"
      );
00486 }
00487 
00488 //-----------------------------------------------------------------------
00489 static void AVR_Disasm_CALL( void )
00490 {
00491     uint32_t u32k = stCPU.k;
00492 
00493     //ruler: 0----5----10---15---20---25---30---35---40" );
00494     printf( "call 0x%X                \t ; Long Call to Subroutine: PC = 0x%X \n",
00495                 u32k, u32k
00496                 );
00497 }
00498 //-----------------------------------------------------------------------
00499 static void AVR_Disasm_RET( void )
00500 {
00501     //ruler: 0----5----10---15---20---25---30---35---40" );
00502     printf( "ret                      \t ; Return from subroutine\n" );
00503 }
00504 
00505 //-----------------------------------------------------------------------
```

```
00506 static void AVR_Disasm_RETI( void )
00507 {
00508     //ruler: 0----5----10---15---20---25---30---35---40" );
00509     printf( "reti                       \t ; Return from interrupt\n" );
00510 }
00511
00512 //-----------------------------------------------------------------------
00513 static void AVR_Disasm_CPSE( void )
00514 {
00515     uint8_t u8Rd = Register_From_Rd();
00516     uint8_t u8Rr = Register_From_Rr();
00517
00518     //ruler: 0----5----10---15---20---25---30---35---40" );
00519     printf( "cpse r%d, r%d              \t ; Compare, Skip Next If r%d = r%d\n",
00520                 u8Rd, u8Rr,
00521                 u8Rd, u8Rr
00522                 );
00523 }
00524
00525 //-----------------------------------------------------------------------
00526 static void AVR_Disasm_CP( void )
00527 {
00528     uint8_t u8Rd = Register_From_Rd();
00529     uint8_t u8Rr = Register_From_Rr();
00530
00531     //ruler: 0----5----10---15---20---25---30---35---40" );
00532     printf( "cp r%d, r%d                \t ; Compare: r%d == r%d\n",
00533                 u8Rd, u8Rr,
00534                 u8Rd, u8Rr
00535                 );
00536 }
00537
00538 //-----------------------------------------------------------------------
00539 static void AVR_Disasm_CPC( void )
00540 {
00541     uint8_t u8Rd = Register_From_Rd();
00542     uint8_t u8Rr = Register_From_Rr();
00543
00544     //ruler: 0----5----10---15---20---25---30---35---40" );
00545     printf( "cpc r%d, r%d               \t ; Compare with carry: r%d == r%d + C\n",
00546                 u8Rd, u8Rr,
00547                 u8Rd, u8Rr
00548                 );
00549 }
00550
00551 //-----------------------------------------------------------------------
00552 static void AVR_Disasm_CPI( void )
00553 {
00554     uint8_t u8Rd = Register_From_Rd();
00555     uint8_t u8K = stCPU.K;
00556
00557     //ruler: 0----5----10---15---20---25---30---35---40" );
00558     printf( "cpi r%d, %d                \t ; Compare with Immediate: r%d == %d\n",
00559                 u8Rd, u8K,
00560                 u8Rd, u8K
00561                 );
00562 }
00563
00564 //-----------------------------------------------------------------------
00565 static void AVR_Disasm_SBRC( void )
00566 {
00567     uint8_t u8Rd = Register_From_Rd();
00568     uint8_t u8b = stCPU.b;
00569
00570     //ruler: 0----5----10---15---20---25---30---35---40" );
00571     printf( "sbrc r%d, %d               \t ; Skip if Bit (%d) in Register (r%d) Cleared \n",
00572                 u8Rd, u8b,
00573                 u8Rd, u8b
00574                 );
00575 }
00576
00577 //-----------------------------------------------------------------------
00578 static void AVR_Disasm_SBRS( void )
00579 {
00580     uint8_t u8Rd = Register_From_Rd();
00581     uint8_t u8b = stCPU.b;
00582
00583     //ruler: 0----5----10---15---20---25---30---35---40" );
00584     printf( "sbrs r%d, %d               \t ; Skip if Bit (%d) in Register (r%d) Set \n",
00585                 u8Rd, u8b,
00586                 u8Rd, u8b
00587                 );
00588
00589 }
00590
00591 //-----------------------------------------------------------------------
00592 static void AVR_Disasm_SBIC( void )
```

```
00593 {
00594     uint8_t u8A = stCPU.A;
00595     uint8_t u8b = stCPU.b;
00596 
00597     //ruler: 0----5----10---15---20---25---30---35---40" );
00598     printf( "sbic %d, %d               \t ; Skip if Bit (%d) in IO Register (r%d) Cleared \n",
00599                 u8A, u8b,
00600                 u8A, u8b
00601                 );
00602 }
00603 
00604 //-------------------------------------------------------------------------
00605 static void AVR_Disasm_SBIS( void )
00606 {
00607     uint8_t u8A = stCPU.A;
00608     uint8_t u8b = stCPU.b;
00609 
00610     //ruler: 0----5----10---15---20---25---30---35---40" );
00611     printf( "sbis %d, %d               \t ; Skip if Bit (%d) in IO Register (r%d) Set \n",
00612                 u8A, u8b,
00613                 u8A, u8b
00614                 );
00615 }
00616 
00617 //-------------------------------------------------------------------------
00618 static void AVR_Disasm_BRBS( void )
00619 {
00620     uint8_t u8s = stCPU.s;
00621     int8_t  s8k = stCPU.k_s;
00622 
00623     //ruler: 0----5----10---15---20---25---30---35---40" );
00624     printf( "brbs %d, %d               \t ; Branch if Bit (%d) in SR set: PC = PC + %d + 1\n",
00625                 u8s, s8k,
00626                 u8s, s8k
00627                 );
00628 }
00629 
00630 //-------------------------------------------------------------------------
00631 static void AVR_Disasm_BRBC( void )
00632 {
00633     uint8_t u8s = stCPU.s;
00634     int8_t  s8k = stCPU.k_s;
00635 
00636     //ruler: 0----5----10---15---20---25---30---35---40" );
00637     printf( "brbc %d, %d               \t ; Branch if Bit (%d) in SR clear: PC = PC + %d + 1\n",
00638                 u8s, s8k,
00639                 u8s, s8k
00640                 );
00641 }
00642 
00643 //-------------------------------------------------------------------------
00644 static void AVR_Disasm_BREQ( void )
00645 {
00646     int8_t  s8k = stCPU.k_s;
00647 
00648     //ruler: 0----5----10---15---20---25---30---35---40" );
00649     printf( "breq %d                   \t ; Branch if zero flag set: PC = PC + %d + 1\n",
00650                 s8k,
00651                 s8k
00652                 );
00653 }
00654 
00655 //-------------------------------------------------------------------------
00656 static void AVR_Disasm_BRNE( void )
00657 {
00658     int8_t  s8k = stCPU.k_s;
00659 
00660     //ruler: 0----5----10---15---20---25---30---35---40" );
00661     printf( "brne %d                   \t ; Branch if zero flag clear: PC = PC + %d + 1\n",
00662                 s8k,
00663                 s8k
00664                 );
00665 }
00666 
00667 //-------------------------------------------------------------------------
00668 static void AVR_Disasm_BRCS( void )
00669 {
00670     int8_t  s8k = stCPU.k_s;
00671 
00672     //ruler: 0----5----10---15---20---25---30---35---40" );
00673     printf( "brcs %d                   \t ; Branch if carry flag set: PC = PC + %d + 1\n",
00674                 s8k,
00675                 s8k
00676                 );
00677 }
00678 
00679 //-------------------------------------------------------------------------
```

```
00680 static void AVR_Disasm_BRCC( void )
00681 {
00682     int8_t  s8k = stCPU.k_s;
00683
00684     //ruler: 0----5----10---15---20---25---30---35---40" );
00685     printf( "brcc %d                  \t ; Branch if carry flag clear: PC = PC + %d + 1\n",
00686                 s8k,
00687                 s8k
00688                 );
00689
00690 }
00691
00692 //-----------------------------------------------------------------------
00693 static void AVR_Disasm_BRSH( void )
00694 {
00695     int8_t  s8k = stCPU.k_s;
00696
00697     //ruler: 0----5----10---15---20---25---30---35---40" );
00698     printf( "brsh %d                  \t ; Branch if same or higher: PC = PC + %d + 1\n",
00699                 s8k,
00700                 s8k
00701                 );
00702 }
00703
00704 //-----------------------------------------------------------------------
00705 static void AVR_Disasm_BRLO( void )
00706 {
00707     int8_t  s8k = stCPU.k_s;
00708
00709     //ruler: 0----5----10---15---20---25---30---35---40" );
00710     printf( "brlo %d                  \t ; Branch if lower: PC = PC + %d + 1\n",
00711                 s8k,
00712                 s8k
00713                 );
00714 }
00715
00716 //-----------------------------------------------------------------------
00717 static void AVR_Disasm_BRMI( void )
00718 {
00719     int8_t  s8k = stCPU.k_s;
00720
00721     //ruler: 0----5----10---15---20---25---30---35---40" );
00722     printf( "brmi %d                  \t ; Branch if minus: PC = PC + %d + 1\n",
00723                 s8k,
00724                 s8k
00725                 );
00726 }
00727
00728 //-----------------------------------------------------------------------
00729 static void AVR_Disasm_BRPL( void )
00730 {
00731     int8_t  s8k = stCPU.k_s;
00732
00733     //ruler: 0----5----10---15---20---25---30---35---40" );
00734     printf( "brpl %d                  \t ; Branch if plus: PC = PC + %d + 1\n",
00735                 s8k,
00736                 s8k
00737                 );
00738 }
00739
00740 //-----------------------------------------------------------------------
00741 static void AVR_Disasm_BRGE( void )
00742 {
00743     int8_t  s8k = stCPU.k_s;
00744
00745     //ruler: 0----5----10---15---20---25---30---35---40" );
00746     printf( "brge %d                  \t ; Branch if greater-or-equal (signed): PC = PC + %d + 1\n",
00747                 s8k,
00748                 s8k
00749                 );
00750 }
00751
00752 //-----------------------------------------------------------------------
00753 static void AVR_Disasm_BRLT( void )
00754 {
00755     int8_t  s8k = stCPU.k_s;
00756
00757     //ruler: 0----5----10---15---20---25---30---35---40" );
00758     printf( "brlt %d                  \t ; Branch if less-than (signed): PC = PC + %d + 1\n",
00759                 s8k,
00760                 s8k
00761                 );
00762 }
00763
00764 //-----------------------------------------------------------------------
00765 static void AVR_Disasm_BRHS( void )
00766 {
```

```
00767     int8_t  s8k = stCPU.k_s;
00768
00769     //ruler: 0----5----10---15---20---25---30---35---40" );
00770     printf( "brlt %d                  \t ; Branch if half-carry set: PC = PC + %d + 1\n",
00771                 s8k,
00772                 s8k
00773                 );
00774 }
00775
00776 //-------------------------------------------------------------------------
00777 static void AVR_Disasm_BRHC( void )
00778 {
00779     int8_t  s8k = stCPU.k_s;
00780
00781     //ruler: 0----5----10---15---20---25---30---35---40" );
00782     printf( "brhc %d                  \t ; Branch if half-carry clear: PC = PC + %d + 1\n",
00783                 s8k,
00784                 s8k
00785                 );
00786
00787 }
00788
00789 //-------------------------------------------------------------------------
00790 static void AVR_Disasm_BRTS( void )
00791 {
00792     int8_t  s8k = stCPU.k_s;
00793
00794     //ruler: 0----5----10---15---20---25---30---35---40" );
00795     printf( "brts %d                  \t ; Branch if T-flag set: PC = PC + %d + 1\n",
00796                 s8k,
00797                 s8k
00798                 );
00799 }
00800
00801 //-------------------------------------------------------------------------
00802 static void AVR_Disasm_BRTC( void )
00803 {
00804     int8_t  s8k = stCPU.k_s;
00805
00806     //ruler: 0----5----10---15---20---25---30---35---40" );
00807     printf( "brtc %d                  \t ; Branch if T-flag clear: PC = PC + %d + 1\n",
00808                 s8k,
00809                 s8k
00810                 );
00811 }
00812
00813 //-------------------------------------------------------------------------
00814 static void AVR_Disasm_BRVS( void )
00815 {
00816     int8_t  s8k = stCPU.k_s;
00817
00818     //ruler: 0----5----10---15---20---25---30---35---40" );
00819     printf( "brvs %d                  \t ; Branch if Overflow set: PC = PC + %d + 1\n",
00820                 s8k,
00821                 s8k
00822                 );
00823 }
00824
00825 //-------------------------------------------------------------------------
00826 static void AVR_Disasm_BRVC( void )
00827 {
00828     int8_t  s8k = stCPU.k_s;
00829
00830     //ruler: 0----5----10---15---20---25---30---35---40" );
00831     printf( "brvc %d                  \t ; Branch if Overflow clear: PC = PC + %d + 1\n",
00832                 s8k,
00833                 s8k
00834                 );
00835 }
00836
00837 //-------------------------------------------------------------------------
00838 static void AVR_Disasm_BRIE( void )
00839 {
00840     int8_t  s8k = stCPU.k_s;
00841
00842     //ruler: 0----5----10---15---20---25---30---35---40" );
00843     printf( "brie %d                  \t ; Branch if Interrupt Enabled: PC = PC + %d + 1\n",
00844                 s8k,
00845                 s8k
00846                 );
00847 }
00848
00849 //-------------------------------------------------------------------------
00850 static void AVR_Disasm_BRID( void )
00851 {
00852     int8_t  s8k = stCPU.k_s;
00853
```

```
00854     //ruler: 0----5----10---15---20---25---30---35---40" );
00855     printf( "brid %d                  \t ; Branch if Interrupt Disabled: PC = PC + %d + 1\n",
00856                 s8k,
00857                 s8k
00858                 );
00859
00860 }
00861
00862 //-----------------------------------------------------------------------------
00863 static void AVR_Disasm_MOV( void )
00864 {
00865     uint8_t u8Rd = Register_From_Rd();
00866     uint8_t u8Rr = Register_From_Rr();
00867
00868     //ruler: 0----5----10---15---20---25---30---35---40" );
00869     printf( "mov r%d, r%d             \t ; Copy Register: r%d = r%d\n",
00870                 u8Rd, u8Rr,
00871                 u8Rd, u8Rr
00872                 );
00873 }
00874
00875 //-----------------------------------------------------------------------------
00876 static void AVR_Disasm_MOVW( void )
00877 {
00878     uint16_t u16Rd = Register_From_Rd16();
00879     uint16_t u16Rr = Register_From_Rr16();
00880
00881     //ruler: 0----5----10---15---20---25---30---35---40" );
00882     printf( "movw r%d:r%d, r%d:r%d    \t ; Copy Register (Word): r%d:r%d = r%d:r%d\n",
00883                 u16Rd+1, u16Rd, u16Rr+1, u16Rr,
00884                 u16Rd+1, u16Rd, u16Rr+1, u16Rr
00885                 );
00886 }
00887
00888 //-----------------------------------------------------------------------------
00889 static void AVR_Disasm_LDI( void )
00890 {
00891     uint8_t u8Rd = Register_From_Rd();
00892     uint8_t u8K = stCPU.K;
00893
00894     //ruler: 0----5----10---15---20---25---30---35---40" );
00895     printf( "ldi r%d, %d              \t ; Load Immediate: r%d = %d\n",
00896                 u8Rd, u8K,
00897                 u8Rd, u8K
00898                 );
00899 }
00900
00901 //-----------------------------------------------------------------------------
00902 static void AVR_Disasm_LDS( void )
00903 {
00904     uint8_t u8Rd = Register_From_Rd();
00905     uint16_t u16k = stCPU.k;
00906
00907     //ruler: 0----5----10---15---20---25---30---35---40" );
00908     printf( "lds r%d, %d              \t ; Load Direct from Data Space: r%d = (%d)\n",
00909                 u8Rd, u16k,
00910                 u8Rd, u16k
00911                 );
00912 }
00913
00914 //-----------------------------------------------------------------------------
00915 static void AVR_Disasm_LD_X_Indirect( void )
00916 {
00917     uint8_t u8Rd = Register_From_Rd();
00918
00919     //ruler: 0----5----10---15---20---25---30---35---40" );
00920     printf( "ld r%d, X                \t ; Load Indirect from Data Space\n",
00921                 u8Rd
00922                 );
00923 }
00924
00925 //-----------------------------------------------------------------------------
00926 static void AVR_Disasm_LD_X_Indirect_Postinc( void )
00927 {
00928     uint8_t u8Rd = Register_From_Rd();
00929
00930     //ruler: 0----5----10---15---20---25---30---35---40" );
00931     printf( "ld r%d, X+               \t ; Load Indirect from Data Space w/Postincrement\n",
00932                 u8Rd
00933                 );
00934 }
00935
00936 //-----------------------------------------------------------------------------
00937 static void AVR_Disasm_LD_X_Indirect_Predec( void )
00938 {
00939     uint8_t u8Rd = Register_From_Rd();
00940
```

```
00941     //ruler: 0----5----10---15---20---25---30---35---40" );
00942     printf( "ld r%d, -X               \t ; Load Indirect from Data Space w/Predecrement\n",
00943                 u8Rd
00944                 );
00945 }
00946
00947 //-----------------------------------------------------------------------
00948 static void AVR_Disasm_LD_Y_Indirect( void )
00949 {
00950     uint8_t u8Rd = Register_From_Rd();
00951
00952     //ruler: 0----5----10---15---20---25---30---35---40" );
00953     printf( "ld r%d, Y                \t ; Load Indirect from Data Space\n",
00954                 u8Rd
00955                 );
00956 }
00957
00958 //-----------------------------------------------------------------------
00959 static void AVR_Disasm_LD_Y_Indirect_Postinc( void )
00960 {
00961     uint8_t u8Rd = Register_From_Rd();
00962
00963     //ruler: 0----5----10---15---20---25---30---35---40" );
00964     printf( "ld r%d, Y+               \t ; Load Indirect from Data Space w/Postincrement\n",
00965                 u8Rd
00966                 );
00967 }
00968
00969 //-----------------------------------------------------------------------
00970 static void AVR_Disasm_LD_Y_Indirect_Predec( void )
00971 {
00972     uint8_t u8Rd = Register_From_Rd();
00973
00974     //ruler: 0----5----10---15---20---25---30---35---40" );
00975     printf( "ld r%d, -Y               \t ; Load Indirect from Data Space w/Predecrement\n",
00976                 u8Rd
00977                 );
00978 }
00979
00980 //-----------------------------------------------------------------------
00981 static void AVR_Disasm_LDD_Y( void )
00982 {
00983     uint8_t u8Rd = Register_From_Rd();
00984     uint8_t u8q = stCPU.q;
00985
00986     //ruler: 0----5----10---15---20---25---30---35---40" );
00987     printf( "ldd r%d, Y+%d            \t ; Load Indirect from Data Space (with Displacement)\n",
00988                 u8Rd, u8q
00989                 );
00990 }
00991
00992 //-----------------------------------------------------------------------
00993 static void AVR_Disasm_LD_Z_Indirect( void )
00994 {
00995     uint8_t u8Rd = Register_From_Rd();
00996
00997     //ruler: 0----5----10---15---20---25---30---35---40" );
00998     printf( "ld r%d, Z                \t ; Load Indirect from Data Space\n",
00999                 u8Rd
01000                 );
01001 }
01002
01003 //-----------------------------------------------------------------------
01004 static void AVR_Disasm_LD_Z_Indirect_Postinc( void )
01005 {
01006     uint8_t u8Rd = Register_From_Rd();
01007
01008     //ruler: 0----5----10---15---20---25---30---35---40" );
01009     printf( "ld r%d, Z+               \t ; Load Indirect from Data Space w/Postincrement\n",
01010                 u8Rd
01011                 );
01012 }
01013
01014 //-----------------------------------------------------------------------
01015 static void AVR_Disasm_LD_Z_Indirect_Predec( void )
01016 {
01017     uint8_t u8Rd = Register_From_Rd();
01018
01019     //ruler: 0----5----10---15---20---25---30---35---40" );
01020     printf( "ld r%d, -Z               \t ; Load Indirect from Data Space w/Predecrement\n",
01021                 u8Rd
01022                 );
01023 }
01024
01025 //-----------------------------------------------------------------------
01026 static void AVR_Disasm_LDD_Z( void )
01027 {
```

```
01028     uint8_t u8Rd = Register_From_Rd();
01029     uint8_t u8q = stCPU.q;
01030
01031     //ruler: 0----5----10---15---20---25---30---35---40" );
01032     printf( "ldd r%d, Z+%d             \t ; Load Indirect from Data Space (with Displacement)\n",
01033                 u8Rd, u8q
01034                 );
01035 }
01036
01037 //-------------------------------------------------------------------------
01038 static void AVR_Disasm_STS( void )
01039 {
01040     uint8_t u8Rd = Register_From_Rd();
01041     uint16_t u16k = stCPU.k;
01042
01043     //ruler: 0----5----10---15---20---25---30---35---40" );
01044     printf( "sts %d, r%d               \t ; Store Direct to Data Space: (%d) = r%d\n",
01045                 u16k, u8Rd,
01046                 u16k, u8Rd
01047                 );
01048 }
01049
01050 //-------------------------------------------------------------------------
01051 static void AVR_Disasm_ST_X_Indirect( void )
01052 {
01053     uint8_t u8Rd = Register_From_Rd();
01054
01055     //ruler: 0----5----10---15---20---25---30---35---40" );
01056     printf( "st X, r%d                 \t ; Store Indirect\n",
01057                 u8Rd
01058                 );
01059 }
01060
01061 //-------------------------------------------------------------------------
01062 static void AVR_Disasm_ST_X_Indirect_Postinc( void )
01063 {
01064     uint8_t u8Rd = Register_From_Rd();
01065
01066     //ruler: 0----5----10---15---20---25---30---35---40" );
01067     printf( "st X+, r%d                \t ; Store Indirect w/Postincrement \n",
01068                 u8Rd
01069                 );
01070 }
01071
01072 //-------------------------------------------------------------------------
01073 static void AVR_Disasm_ST_X_Indirect_Predec( void )
01074 {
01075     uint8_t u8Rd = Register_From_Rd();
01076
01077     //ruler: 0----5----10---15---20---25---30---35---40" );
01078     printf( "st -X, r%d                \t ; Store Indirect w/Predecrement\n",
01079                 u8Rd
01080                 );
01081 }
01082
01083 //-------------------------------------------------------------------------
01084 static void AVR_Disasm_ST_Y_Indirect( void )
01085 {
01086     uint8_t u8Rd = Register_From_Rd();
01087
01088     //ruler: 0----5----10---15---20---25---30---35---40" );
01089     printf( "st Y, r%d                 \t ; Store Indirect\n",
01090                 u8Rd
01091                 );
01092 }
01093
01094 //-------------------------------------------------------------------------
01095 static void AVR_Disasm_ST_Y_Indirect_Postinc( void )
01096 {
01097     uint8_t u8Rd = Register_From_Rd();
01098
01099     //ruler: 0----5----10---15---20---25---30---35---40" );
01100     printf( "st Y+, r%d                \t ; Store Indirect w/Postincrement \n",
01101                 u8Rd
01102                 );
01103 }
01104
01105 //-------------------------------------------------------------------------
01106 static void AVR_Disasm_ST_Y_Indirect_Predec( void )
01107 {
01108     uint8_t u8Rd = Register_From_Rd();
01109
01110     //ruler: 0----5----10---15---20---25---30---35---40" );
01111     printf( "st -Y, r%d                \t ; Store Indirect w/Predecrement\n",
01112                 u8Rd
01113                 );
01114 }
```

```
01115
01116 //------------------------------------------------------------------------
01117 static void AVR_Disasm_STD_Y( void )
01118 {
01119     uint8_t u8Rd = Register_From_Rd();
01120     uint8_t u8q = stCPU.q;
01121
01122     //ruler: 0----5----10---15---20---25---30---35---40" );
01123     printf( "std Y+%d, r%d             \t ; Store Indirect from Data Space (with Displacement)\n",
01124                 u8q, u8Rd
01125                 );
01126 }
01127
01128 //------------------------------------------------------------------------
01129 static void AVR_Disasm_ST_Z_Indirect( void )
01130 {
01131     uint8_t u8Rd = Register_From_Rd();
01132
01133     //ruler: 0----5----10---15---20---25---30---35---40" );
01134     printf( "st Z, r%d                 \t ; Store Indirect\n",
01135                 u8Rd
01136                 );
01137 }
01138
01139 //------------------------------------------------------------------------
01140 static void AVR_Disasm_ST_Z_Indirect_Postinc( void )
01141 {
01142     uint8_t u8Rd = Register_From_Rd();
01143
01144     //ruler: 0----5----10---15---20---25---30---35---40" );
01145     printf( "st Z+, r%d                \t ; Store Indirect w/Postincrement \n",
01146                 u8Rd
01147                 );
01148 }
01149
01150 //------------------------------------------------------------------------
01151 static void AVR_Disasm_ST_Z_Indirect_Predec( void )
01152 {
01153     uint8_t u8Rd = Register_From_Rd();
01154
01155     //ruler: 0----5----10---15---20---25---30---35---40" );
01156     printf( "st -Z, r%d                \t ; Store Indirect w/Predecrement\n",
01157                 u8Rd
01158                 );
01159 }
01160
01161 //------------------------------------------------------------------------
01162 static void AVR_Disasm_STD_Z( void )
01163 {
01164     uint8_t u8Rd = Register_From_Rd();
01165     uint8_t u8q = stCPU.q;
01166
01167     //ruler: 0----5----10---15---20---25---30---35---40" );
01168     printf( "std Z+%d, r%d             \t ; Store Indirect from Data Space (with Displacement)\n",
01169                 u8q, u8Rd
01170                 );
01171 }
01172
01173 //------------------------------------------------------------------------
01174 static void AVR_Disasm_LPM( void )
01175 {
01176     //ruler: 0----5----10---15---20---25---30---35---40" );
01177     printf( "lpm                       \t ; Load Program Memory: r0 = (Z)\n" );
01178 }
01179
01180 //------------------------------------------------------------------------
01181 static void AVR_Disasm_LPM_Z( void )
01182 {
01183     uint8_t u8Rd = Register_From_Rd();
01184
01185     //ruler: 0----5----10---15---20---25---30---35---40" );
01186     printf( "lpm r%d, Z                \t ; Load Program Memory: r%d = (Z)\n",
01187                 u8Rd,
01188                 u8Rd
01189                 );
01190 }
01191
01192 //------------------------------------------------------------------------
01193 static void AVR_Disasm_LPM_Z_Postinc( void )
01194 {
01195     uint8_t u8Rd = Register_From_Rd();
01196
01197     //ruler: 0----5----10---15---20---25---30---35---40" );
01198     printf( "lpm r%d, Z+               \t ; Load Program Memory with Postincrement: r%d = (Z), Z = Z + 1\n"
      ,
01199                 u8Rd,
01200                 u8Rd
```

```
01201                 );
01202 }
01203
01204 //-------------------------------------------------------------------------
01205 static void AVR_Disasm_ELPM( void )
01206 {
01207     //ruler: 0----5----10---15---20---25---30---35---40" );
01208     printf( "elpm                       \t ; (Extended) Load Program Memory: r0 = (Z)\n" );
01209 }
01210
01211 //-------------------------------------------------------------------------
01212 static void AVR_Disasm_ELPM_Z( void )
01213 {
01214     uint8_t u8Rd = Register_From_Rd();
01215
01216     //ruler: 0----5----10---15---20---25---30---35---40" );
01217     printf( "elpm r%d, Z                \t ; (Extended) Load Program Memory: r%d = (Z)\n",
01218                 u8Rd,
01219                 u8Rd
01220                 );
01221 }
01222
01223 //-------------------------------------------------------------------------
01224 static void AVR_Disasm_ELPM_Z_Postinc( void )
01225 {
01226     uint8_t u8Rd = Register_From_Rd();
01227
01228     //ruler: 0----5----10---15---20---25---30---35---40" );
01229     printf( "elpm r%d, Z+               \t ; (Extended) Load Program Memory w/Postincrement: r%d = (Z), Z =
     Z + 1\n",
01230                 u8Rd,
01231                 u8Rd
01232                 );
01233 }
01234
01235 //-------------------------------------------------------------------------
01236 static void AVR_Disasm_SPM( void )
01237 {
01238     //ruler: 0----5----10---15---20---25---30---35---40" );
01239     printf( "spm                        \t ; Store Program Memory\n" );
01240 }
01241
01242 //-------------------------------------------------------------------------
01243 static void AVR_Disasm_SPM_Z_Postinc2( void )
01244 {
01245     //ruler: 0----5----10---15---20---25---30---35---40" );
01246     printf( "spm Z+                     \t ; Store Program Memory Z = Z + 2 \n" );
01247 }
01248
01249 //-------------------------------------------------------------------------
01250 static void AVR_Disasm_IN( void )
01251 {
01252     uint8_t u8Rd = Register_From_Rd();
01253     uint8_t u8A = stCPU.A;
01254
01255     //ruler: 0----5----10---15---20---25---30---35---40" );
01256     printf( "in r%d, %d                 \t ; Load an I/O location to register\n",
01257                 u8Rd,
01258                 u8A
01259                 );
01260 }
01261
01262 //-------------------------------------------------------------------------
01263 static void AVR_Disasm_OUT( void )
01264 {
01265     uint8_t u8Rd = Register_From_Rd();
01266     uint8_t u8A = stCPU.A;
01267
01268     //ruler: 0----5----10---15---20---25---30---35---40" );
01269     printf( "out %d, r%d                \t ; Load an I/O location to register\n",
01270                 u8A,
01271                 u8Rd
01272                 );
01273
01274 }
01275
01276 //-------------------------------------------------------------------------
01277 static void AVR_Disasm_LAC( void )
01278 {
01279     uint8_t u8Rd = Register_From_Rd();
01280
01281     //ruler: 0----5----10---15---20---25---30---35---40" );
01282     printf( "lac Z, r%d                    \t ; Load And Clear\n",
01283                 u8Rd
01284                 );
01285 }
01286
```

```
01287 //-------------------------------------------------------------------------
01288 static void AVR_Disasm_LAS( void )
01289 {
01290     uint8_t u8Rd = Register_From_Rd();
01291 
01292     //ruler: 0----5----10---15---20---25---30---35---40" );
01293     printf( "las Z, r%d                  \t ; Load And Set\n",
01294                 u8Rd
01295                 );
01296 }
01297 
01298 //-------------------------------------------------------------------------
01299 static void AVR_Disasm_LAT( void )
01300 {
01301     uint8_t u8Rd = Register_From_Rd();
01302 
01303     //ruler: 0----5----10---15---20---25---30---35---40" );
01304     printf( "lat Z, r%d                  \t ; Load And Toggle\n",
01305                 u8Rd
01306                 );
01307 }
01308 
01309 //-------------------------------------------------------------------------
01310 static void AVR_Disasm_LSL( void )
01311 {
01312     uint8_t u8Rd = Register_From_Rd();
01313 
01314     //ruler: 0----5----10---15---20---25---30---35---40" );
01315     printf( "lsl r%d                  \t ; Logical shift left r%d by 1 bit\n",
01316                 u8Rd,
01317                 u8Rd
01318                 );
01319 }
01320 
01321 //-------------------------------------------------------------------------
01322 static void AVR_Disasm_LSR( void )
01323 {
01324     uint8_t u8Rd = Register_From_Rd();
01325 
01326     //ruler: 0----5----10---15---20---25---30---35---40" );
01327     printf( "lsr r%d                  \t ; Logical shift right r%d by 1 bit\n",
01328                 u8Rd,
01329                 u8Rd
01330                 );
01331 }
01332 
01333 //-------------------------------------------------------------------------
01334 static void AVR_Disasm_POP( void )
01335 {
01336     uint8_t u8Rd = Register_From_Rd();
01337 
01338     //ruler: 0----5----10---15---20---25---30---35---40" );
01339     printf( "pop r%d                  \t ; Pop byte from stack into r%d\n",
01340                 u8Rd,
01341                 u8Rd
01342                 );
01343 }
01344 
01345 //-------------------------------------------------------------------------
01346 static void AVR_Disasm_PUSH( void )
01347 {
01348     uint8_t u8Rd = Register_From_Rd();
01349 
01350     //ruler: 0----5----10---15---20---25---30---35---40" );
01351     printf( "push r%d                  \t ; Push register r%d to stack\n",
01352                 u8Rd,
01353                 u8Rd
01354                 );
01355 }
01356 
01357 //-------------------------------------------------------------------------
01358 static void AVR_Disasm_ROL( void )
01359 {
01360     uint8_t u8Rd = Register_From_Rd();
01361 
01362     //ruler: 0----5----10---15---20---25---30---35---40" );
01363     printf( "rol r%d                  \t ; Rotate Left through Carry\n",
01364                 u8Rd
01365                 );
01366 }
01367 
01368 //-------------------------------------------------------------------------
01369 static void AVR_Disasm_ROR( void )
01370 {
01371     uint8_t u8Rd = Register_From_Rd();
01372 
01373     //ruler: 0----5----10---15---20---25---30---35---40" );
```

```
01374     printf( "ror r%d                  \t ; Rotate Right through Carry\n",
01375                 u8Rd
01376                 );
01377 }
01378
01379 //-----------------------------------------------------------------------
01380 static void AVR_Disasm_ASR( void )
01381 {
01382     uint8_t u8Rd = Register_From_Rd();
01383
01384     //ruler: 0----5----10---15---20---25---30---35---40" );
01385     printf( "asr r%d                  \t ; Arithmatic Shift Right\n",
01386                 u8Rd
01387                 );
01388 }
01389
01390 //-----------------------------------------------------------------------
01391 static void AVR_Disasm_SWAP( void )
01392 {
01393     uint8_t u8Rd = Register_From_Rd();
01394
01395     //ruler: 0----5----10---15---20---25---30---35---40" );
01396     printf( "swap r%d                 \t ; Swap high/low Nibbles in Register\n",
01397                 u8Rd
01398                 );
01399 }
01400
01401 //-----------------------------------------------------------------------
01402 static void AVR_Disasm_BSET( void )
01403 {
01404     uint8_t u8s = stCPU.s;
01405
01406     //ruler: 0----5----10---15---20---25---30---35---40" );
01407     printf( "bset %d                  \t ; Set bit %d in status register\n",
01408                 u8s,
01409                 u8s
01410                 );
01411 }
01412
01413 //-----------------------------------------------------------------------
01414 static void AVR_Disasm_BCLR( void )
01415 {
01416     uint8_t u8s = stCPU.s;
01417
01418     //ruler: 0----5----10---15---20---25---30---35---40" );
01419     printf( "bclr %d                  \t ; Clear bit %d in status register\n",
01420                 u8s,
01421                 u8s
01422                 );
01423 }
01424
01425 //-----------------------------------------------------------------------
01426 static void AVR_Disasm_SBI( void )
01427 {
01428     uint8_t u8b = stCPU.b;
01429     uint8_t u8A = stCPU.A;
01430
01431     //ruler: 0----5----10---15---20---25---30---35---40" );
01432     printf( "sbi %d, %d               \t ; Set bit in I/O register\n",
01433                 u8A,
01434                 u8b
01435                 );
01436 }
01437
01438 //-----------------------------------------------------------------------
01439 static void AVR_Disasm_CBI( void )
01440 {
01441     uint8_t u8s = stCPU.b;
01442     uint8_t u8A = stCPU.A;
01443
01444     //ruler: 0----5----10---15---20---25---30---35---40" );
01445     printf( "cbi %d, %d               \t ; Clear bit in I/O register\n",
01446                 u8A,
01447                 u8s
01448                 );
01449 }
01450
01451 //-----------------------------------------------------------------------
01452 static void AVR_Disasm_BST( void )
01453 {
01454     uint8_t u8Rd = Register_From_Rd();
01455     uint8_t u8b = stCPU.b;
01456
01457     //ruler: 0----5----10---15---20---25---30---35---40" );
01458     printf( "bst r%d, %d              \t ; Store Bit %d of r%d in the T register\n",
01459                 u8Rd, u8b,
01460                 u8b, u8Rd
```

```
01461                 );
01462 }
01463
01464 //-----------------------------------------------------------------------
01465 static void AVR_Disasm_BLD( void )
01466 {
01467     uint8_t u8Rd = Register_From_Rd();
01468     uint8_t u8b = stCPU.b;
01469
01470     //ruler: 0----5----10---15---20---25---30---35---40" );
01471     printf( "bld r%d, %d                \t ; Load the T register into Bit %d of r%d\n",
01472                 u8Rd, u8b,
01473                 u8b, u8Rd
01474                 );
01475 }
01476
01477 //-----------------------------------------------------------------------
01478 static void AVR_Disasm_SEC( void )
01479 {
01480     //ruler: 0----5----10---15---20---25---30---35---40" );
01481     printf( "sec                        \t ; Set the carry flag in the SR\n" );
01482 }
01483
01484 //-----------------------------------------------------------------------
01485 static void AVR_Disasm_CLC( void )
01486 {
01487     //ruler: 0----5----10---15---20---25---30---35---40" );
01488     printf( "clc                        \t ; Clear the carry flag in the SR\n" );
01489 }
01490
01491 //-----------------------------------------------------------------------
01492 static void AVR_Disasm_SEN( void )
01493 {
01494     //ruler: 0----5----10---15---20---25---30---35---40" );
01495     printf( "sen                        \t ; Set the negative flag in the SR\n" );
01496 }
01497
01498 //-----------------------------------------------------------------------
01499 static void AVR_Disasm_CLN( void )
01500 {
01501     //ruler: 0----5----10---15---20---25---30---35---40" );
01502     printf( "cln                        \t ; Clear the negative flag in the SR\n" );
01503 }
01504
01505 //-----------------------------------------------------------------------
01506 static void AVR_Disasm_SEZ( void )
01507 {
01508     //ruler: 0----5----10---15---20---25---30---35---40" );
01509     printf( "sez                        \t ; Set the zero flag in the SR\n" );
01510 }
01511
01512 //-----------------------------------------------------------------------
01513 static void AVR_Disasm_CLZ( void )
01514 {
01515     //ruler: 0----5----10---15---20---25---30---35---40" );
01516     printf( "clz                        \t ; Clear the zero flag in the SR\n" );
01517 }
01518
01519 //-----------------------------------------------------------------------
01520 static void AVR_Disasm_SEI( void )
01521 {
01522     //ruler: 0----5----10---15---20---25---30---35---40" );
01523     printf( "sei                        \t ; Enable MCU interrupts\n" );
01524 }
01525
01526 //-----------------------------------------------------------------------
01527 static void AVR_Disasm_CLI( void )
01528 {
01529     //ruler: 0----5----10---15---20---25---30---35---40" );
01530     printf( "cli                        \t ; Disable MCU interrupts\n" );
01531 }
01532
01533 //-----------------------------------------------------------------------
01534 static void AVR_Disasm_SES( void )
01535 {
01536     //ruler: 0----5----10---15---20---25---30---35---40" );
01537     printf( "ses                        \t ; Set the sign flag in the SR\n" );
01538 }
01539
01540 //-----------------------------------------------------------------------
01541 static void AVR_Disasm_CLS( void )
01542 {
01543     //ruler: 0----5----10---15---20---25---30---35---40" );
01544     printf( "cls                        \t ; Clear the sign flag in the SR\n" );
01545 }
01546
01547 //-----------------------------------------------------------------------
```

```
01548 static void AVR_Disasm_SEV( void )
01549 {
01550     //ruler: 0----5----10---15---20---25---30---35---40" );
01551     printf( "sev                        \t ; Set the overflow flag in the SR\n" );
01552 }
01553
01554 //-----------------------------------------------------------------------
01555 static void AVR_Disasm_CLV( void )
01556 {
01557     //ruler: 0----5----10---15---20---25---30---35---40" );
01558     printf( "clv                        \t ; Clear the overflow flag in the SR\n" );
01559 }
01560
01561 //-----------------------------------------------------------------------
01562 static void AVR_Disasm_SET( void )
01563 {
01564     //ruler: 0----5----10---15---20---25---30---35---40" );
01565     printf( "set                        \t ; Set the T-flag in the SR\n" );
01566 }
01567
01568 //-----------------------------------------------------------------------
01569 static void AVR_Disasm_CLT( void )
01570 {
01571     //ruler: 0----5----10---15---20---25---30---35---40" );
01572     printf( "clt                        \t ; Clear the T-flag in the SR\n" );
01573 }
01574
01575 //-----------------------------------------------------------------------
01576 static void AVR_Disasm_SEH( void )
01577 {
01578     //ruler: 0----5----10---15---20---25---30---35---40" );
01579     printf( "seh                        \t ; Set half-carry flag in SR\n" );
01580 }
01581
01582 //-----------------------------------------------------------------------
01583 static void AVR_Disasm_CLH( void )
01584 {
01585     //ruler: 0----5----10---15---20---25---30---35---40" );
01586     printf( "clh                        \t ; Clear half-carry flag in SR\n" );
01587 }
01588
01589 //-----------------------------------------------------------------------
01590 static void AVR_Disasm_BREAK( void )
01591 {
01592     //ruler: 0----5----10---15---20---25---30---35---40" );
01593     printf( "break                      \t ; Halt for debugger\n" );
01594 }
01595
01596 //-----------------------------------------------------------------------
01597 static void AVR_Disasm_NOP( void )
01598 {
01599     //ruler: 0----5----10---15---20---25---30---35---40" );
01600     printf( "nop                        \t ; Do nothing\n" );
01601 }
01602
01603 //-----------------------------------------------------------------------
01604 static void AVR_Disasm_SLEEP( void )
01605 {
01606     //ruler: 0----5----10---15---20---25---30---35---40" );
01607     printf( "sleep                      \t ; Put MCU into sleep mode\n" );
01608 }
01609
01610 //-----------------------------------------------------------------------
01611 static void AVR_Disasm_WDR( void )
01612 {
01613     //ruler: 0----5----10---15---20---25---30---35---40" );
01614     printf( "wdr                        \t ; Reset Watchdog Timer\n" );
01615 }
01616
01617 //-----------------------------------------------------------------------
01618 static void AVR_Disasm_XCH( void )
01619 {
01620     uint8_t u8Rd = Register_From_Rd();
01621
01622     //ruler: 0----5----10---15---20---25---30---35---40" );
01623     printf( "xch Z, r%d                 \t ; Exchange registers w/memory\n",
01624                 u8Rd
01625                 );
01626 }
01627
01628 //-----------------------------------------------------------------------
01629 static void AVR_Disasm_Unimplemented()
01630 {
01631     printf( ".db 0x%04X ; Data (not an opcode)\n", stCPU.pu16ROM[ stCPU.u16PC ] );
01632 }
01633
01634 //-----------------------------------------------------------------------
```

```
01635 AVR_Opcode AVR_Disasm_Function( uint16_t OP_ )
01636 {
01637     // Special instructions - "static" encoding
01638     switch (OP_)
01639     {
01640     case 0x0000: return AVR_Disasm_NOP;
01641
01642     case 0x9408: return AVR_Disasm_SEC;
01643     case 0x9409: return AVR_Disasm_IJMP;
01644     case 0x9418: return AVR_Disasm_SEZ;
01645     case 0x9419: return AVR_Disasm_EIJMP;
01646     case 0x9428: return AVR_Disasm_SEN;
01647     case 0x9438: return AVR_Disasm_SEV;
01648     case 0x9448: return AVR_Disasm_SES;
01649     case 0x9458: return AVR_Disasm_SEH;
01650     case 0x9468: return AVR_Disasm_SET;
01651     case 0x9478: return AVR_Disasm_SEI;
01652
01653     case 0x9488: return AVR_Disasm_CLC;
01654     case 0x9498: return AVR_Disasm_CLZ;
01655     case 0x94A8: return AVR_Disasm_CLN;
01656     case 0x94B8: return AVR_Disasm_CLV;
01657     case 0x94C8: return AVR_Disasm_CLS;
01658     case 0x94D8: return AVR_Disasm_CLH;
01659     case 0x94E8: return AVR_Disasm_CLT;
01660     case 0x94F8: return AVR_Disasm_CLI;
01661
01662     case 0x9508: return AVR_Disasm_RET;
01663     case 0x9509: return AVR_Disasm_ICALL;
01664     case 0x9518: return AVR_Disasm_RETI;
01665     case 0x9519: return AVR_Disasm_EICALL;
01666     case 0x9588: return AVR_Disasm_SLEEP;
01667     case 0x9598: return AVR_Disasm_BREAK;
01668     case 0x95A8: return AVR_Disasm_WDR;
01669     case 0x95C8: return AVR_Disasm_LPM;
01670     case 0x95D8: return AVR_Disasm_ELPM;
01671     case 0x95E8: return AVR_Disasm_SPM;
01672     case 0x95F8: return AVR_Disasm_SPM_Z_Postinc2;
01673     }
01674
01675 #if 0
01676     // Note: These disasm handlers are generalized versions of specific mnemonics in the above list.
01677     // For disassembly, it's probably easier to read the output from the more "spcific" mnemonics, so
01678     // those are used.  For emulation, using the generalized functions may be more desirable.
01679     switch( OP_ & 0xFF8F)
01680     {
01681     case 0x9408: return AVR_Disasm_BSET;
01682     case 0x9488: return AVR_Disasm_BCLR;
01683     }
01684 #endif
01685
01686     switch (OP_ & 0xFF88)
01687     {
01688     case 0x0300: return AVR_Disasm_MULSU;
01689     case 0x0308: return AVR_Disasm_FMUL;
01690     case 0x0380: return AVR_Disasm_FMULS;
01691     case 0x0388: return AVR_Disasm_FMULSU;
01692     }
01693
01694     switch (OP_ & 0xFF0F)
01695     {
01696     case 0x940B: return AVR_Disasm_DES;
01697     case 0xEF0F: return AVR_Disasm_SER;
01698     }
01699
01700     switch (OP_ & 0xFF00)
01701     {
01702     case 0x0100: return AVR_Disasm_MOVW;
01703     case 0x9600: return AVR_Disasm_ADIW;
01704     case 0x9700: return AVR_Disasm_SBIW;
01705
01706     case 0x9800: return AVR_Disasm_CBI;
01707     case 0x9900: return AVR_Disasm_SBIC;
01708     case 0x9A00: return AVR_Disasm_SBI;
01709     case 0x9B00: return AVR_Disasm_SBIS;
01710     }
01711
01712     switch (OP_ & 0xFE0F)
01713     {
01714     case 0x8008: return AVR_Disasm_LD_Y_Indirect;
01715     case 0x8000: return AVR_Disasm_LD_Z_Indirect;
01716     case 0x8200: return AVR_Disasm_ST_Z_Indirect;
01717     case 0x8208: return AVR_Disasm_ST_Y_Indirect;
01718
01719     // -- Single 5-bit register...
01720     case 0x9000: return AVR_Disasm_LDS;
01721     case 0x9001: return AVR_Disasm_LD_Z_Indirect_Postinc;
```

```
01722     case 0x9002: return AVR_Disasm_LD_Z_Indirect_Predec;
01723     case 0x9004: return AVR_Disasm_LPM_Z;
01724     case 0x9005: return AVR_Disasm_LPM_Z_Postinc;
01725     case 0x9006: return AVR_Disasm_ELPM_Z;
01726     case 0x9007: return AVR_Disasm_ELPM_Z_Postinc;
01727     case 0x9009: return AVR_Disasm_LD_Y_Indirect_Postinc;
01728     case 0x900A: return AVR_Disasm_LD_Y_Indirect_Predec;
01729     case 0x900C: return AVR_Disasm_LD_X_Indirect;
01730     case 0x900D: return AVR_Disasm_LD_X_Indirect_Postinc;
01731     case 0x900E: return AVR_Disasm_LD_X_Indirect_Predec;
01732     case 0x900F: return AVR_Disasm_POP;
01733
01734     case 0x9200: return AVR_Disasm_STS;
01735     case 0x9201: return AVR_Disasm_ST_Z_Indirect_Postinc;
01736     case 0x9202: return AVR_Disasm_ST_Z_Indirect_Predec;
01737     case 0x9204: return AVR_Disasm_XCH;
01738     case 0x9205: return AVR_Disasm_LAS;
01739     case 0x9206: return AVR_Disasm_LAC;
01740     case 0x9207: return AVR_Disasm_LAT;
01741     case 0x9209: return AVR_Disasm_ST_Y_Indirect_Postinc;
01742     case 0x920A: return AVR_Disasm_ST_Y_Indirect_Predec;
01743     case 0x920C: return AVR_Disasm_ST_X_Indirect;
01744     case 0x920D: return AVR_Disasm_ST_X_Indirect_Postinc;
01745     case 0x920E: return AVR_Disasm_ST_X_Indirect_Predec;
01746     case 0x920F: return AVR_Disasm_PUSH;
01747
01748     // -- One-operand instructions
01749     case 0x9400: return AVR_Disasm_COM;
01750     case 0x9401: return AVR_Disasm_NEG;
01751     case 0x9402: return AVR_Disasm_SWAP;
01752     case 0x9403: return AVR_Disasm_INC;
01753     case 0x9405: return AVR_Disasm_ASR;
01754     case 0x9406: return AVR_Disasm_LSR;
01755     case 0x9407: return AVR_Disasm_ROR;
01756     case 0x940A: return AVR_Disasm_DEC;
01757
01758     }
01759     switch (OP_ & 0xFE0E)
01760     {
01761     case 0x940C: return AVR_Disasm_JMP;
01762     case 0x940E: return AVR_Disasm_CALL;
01763     }
01764
01765     switch (OP_ & 0xFE08)
01766     {
01767
01768     // -- BLD/BST Encoding
01769     case 0xF800: return AVR_Disasm_BLD;
01770     case 0xFA00: return AVR_Disasm_BST;
01771     // -- SBRC/SBRS Encoding
01772     case 0xFC00: return AVR_Disasm_SBRC;
01773     case 0xFE00: return AVR_Disasm_SBRS;
01774     }
01775
01776     switch (OP_ & 0xFC07)
01777     {
01778     // -- Conditional branches
01779     case 0xF000: return AVR_Disasm_BRCS;
01780     // case 0xF000: return AVR_Disasm_BRLO;            // AKA AVR_Disasm_BRCS;
01781     case 0xF001: return AVR_Disasm_BREQ;
01782     case 0xF002: return AVR_Disasm_BRMI;
01783     case 0xF003: return AVR_Disasm_BRVS;
01784     case 0xF004: return AVR_Disasm_BRLT;
01785     case 0xF006: return AVR_Disasm_BRTS;
01786     case 0xF007: return AVR_Disasm_BRIE;
01787     case 0xF400: return AVR_Disasm_BRCC;
01788     // case 0xF400: return AVR_Disasm_BRSH;            // AKA AVR_Disasm_BRCC;
01789     case 0xF401: return AVR_Disasm_BRNE;
01790     case 0xF402: return AVR_Disasm_BRPL;
01791     case 0xF403: return AVR_Disasm_BRVC;
01792     case 0xF404: return AVR_Disasm_BRGE;
01793     case 0xF405: return AVR_Disasm_BRHC;
01794     case 0xF406: return AVR_Disasm_BRTC;
01795     case 0xF407: return AVR_Disasm_BRID;
01796     }
01797
01798     switch (OP_ & 0xFC00)
01799     {
01800     // -- 4-bit register pair
01801     case 0x0200: return AVR_Disasm_MULS;
01802
01803     // -- 5-bit register pairs --
01804     case 0x0400: return AVR_Disasm_CPC;
01805     case 0x0800: return AVR_Disasm_SBC;
01806     case 0x0C00: return AVR_Disasm_ADD;
01807     // case 0x0C00: return AVR_Disasm_LSL; (!! Implemented with: " add rd, rd"
01808     case 0x1000: return AVR_Disasm_CPSE;
```

```
01809     case 0x1300: return AVR_Disasm_ROL;
01810     case 0x1400: return AVR_Disasm_CP;
01811     case 0x1C00: return AVR_Disasm_ADC;
01812     case 0x1800: return AVR_Disasm_SUB;
01813     case 0x2000: return AVR_Disasm_AND;
01814     // case 0x2000: return AVR_Disasm_TST; (!! Implemented with: " and rd, rd"
01815     case 0x2400: return AVR_Disasm_EOR;
01816     case 0x2C00: return AVR_Disasm_MOV;
01817     case 0x2800: return AVR_Disasm_OR;
01818
01819     // -- 5-bit register pairs -- Destination = R1:R0
01820     case 0x9C00: return AVR_Disasm_MUL;
01821     }
01822
01823     switch (OP_ & 0xF800)
01824     {
01825     case  0xB800: return AVR_Disasm_OUT;
01826     case  0xB000: return AVR_Disasm_IN;
01827     }
01828
01829     switch (OP_ & 0xF000)
01830     {
01831     // -- Register immediate --
01832     case 0x3000: return AVR_Disasm_CPI;
01833     case 0x4000: return AVR_Disasm_SBCI;
01834     case 0x5000: return AVR_Disasm_SUBI;
01835     case 0x6000: return AVR_Disasm_ORI;// return AVR_Disasm_SBR;
01836     case 0x7000: return AVR_Disasm_ANDI;
01837
01838     //-- 12-bit immediate
01839     case 0xC000: return AVR_Disasm_RJMP;
01840     case 0xD000: return AVR_Disasm_RCALL;
01841
01842     // -- Register immediate
01843     case 0xE000: return AVR_Disasm_LDI;
01844     }
01845
01846     switch (OP_ & 0xD208)
01847     {
01848     // -- 7-bit signed offset
01849     case 0x8000: return AVR_Disasm_LDD_Z;
01850     case 0x8008: return AVR_Disasm_LDD_Y;
01851     case 0x8200: return AVR_Disasm_STD_Z;
01852     case 0x8208: return AVR_Disasm_STD_Y;
01853     }
01854
01855     return AVR_Disasm_Unimplemented;
01856 }
01857
```

4.13 avr_disasm.h File Reference

AVR Disassembler Implementation.

```
#include "avr_opcodes.h"
```

Functions

- AVR_Opcode AVR_Disasm_Function (uint16_t OP_)

  AVR_Disasm_Function.

4.13.1 Detailed Description

AVR Disassembler Implementation.

Definition in file avr_disasm.h.

4.13.2 Function Documentation

4.13.2.1 AVR_Opcode AVR_Disasm_Function (uint16_t *OP_*)

AVR_Disasm_Function.

Return a function pointer to a disassembly routine corresponding to a given opcode.

Parameters

*OP_*	Opcode to disassemble

Returns

Function pointer that, when called with a valid CPU object and opcode, will produce a valid disassembly statement to standard output.

Definition at line 1635 of file avr_disasm.c.

4.14 avr_disasm.h

```
/*****************************************************************************
 *     (     (                       (    |
 *    )\ )  )\ )    (              )\ )   |
 *   (()/( (()/(    )\     (   (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                        | "Yeah, it does Arduino..."
 * ---------------------------------------+-----------------------------------
 * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 *****************************************************************************/
#ifndef __AVR_DISASM_H__
#define __AVR_DISASM_H__

#include "avr_opcodes.h"

//---------------------------------------------------------------------------
AVR_Opcode AVR_Disasm_Function( uint16_t OP_ );


#endif
```

4.15 avr_interrupt.c File Reference

CPU Interrupt management.

```
#include <stdint.h>
#include "emu_config.h"
#include "avr_cpu.h"
```

Functions

- static void **AVR_NextInterrupt** (void)
- void AVR_InterruptCandidate (uint8_t u8Vector_)

    AVR_InterruptCandidate.
- void AVR_ClearCandidate (uint8_t u8Vector_)

    AVR_ClearCandidate.
- void AVR_Interrupt (void)

    AVR_Interrupt.

4.15.1 Detailed Description

CPU Interrupt management.

Definition in file avr_interrupt.c.

4.15.2 Function Documentation

4.15.2.1 void AVR_ClearCandidate (uint8_t *u8Vector_*)

AVR_ClearCandidate.

Parameters

*u8Vector_*	Vector to clear pending interrupt for.

Definition at line 58 of file avr_interrupt.c.

4.15.2.2 void AVR_Interrupt (void)

AVR_Interrupt.

Entrypoint for CPU interrupts. Stop executing the currently-executing code, push the current PC to the stack, disable interrupts, and resume execution at the new location specified in the vector table.

Definition at line 66 of file avr_interrupt.c.

4.15.2.3 void AVR_InterruptCandidate (uint8_t *u8Vector_*)

AVR_InterruptCandidate.

Given an existing interrupt candidate, determine if the selected interrupt vector is of highier priority. If higher priority, update the candidate.

Parameters

*u8Vector_*	- Candidate interrupt vector.

Definition at line 46 of file avr_interrupt.c.

4.16 avr_interrupt.c

```
00001 /*****************************************************************************
00002  *      (      (                       (      |
00003  *    )\ )   )\ )     (              )\ )    |
00004  *   (()/(  (()/(     )\      (    (  (()/(    | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\   )\   /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ----------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \     | -- [ Virtual ] ------------------
00008  *   | __| | |__    / _ \    \ V /  |   /     | -- [ Runtime ] ------------------
00009  *   |_|   |____| /_/ \_\    \_/   |_|_\     |
00010  *                                          | "Yeah, it does Arduino..."
00011  * -----------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #include <stdint.h>
00022 #include "emu_config.h"
00023 #include "avr_cpu.h"
00024
00025 //-----------------------------------------------------------------------------
00026 static void AVR_NextInterrupt(void)
00027 {
00028     uint32_t i = 0x80000000;
00029     uint32_t j = 31;
00030     while (i)
00031     {
```

```
00032         if ((stCPU.u32IntFlags & i) == i)
00033         {
00034             stCPU.u8IntPriority = j;
00035             return;
00036         }
00037         i >>= 1;
00038         j--;
00039     }
00040
00041     stCPU.u8IntPriority = 255;
00042     stCPU.u32IntFlags = 0;
00043 }
00044
00045 //------------------------------------------------------------------------------
00046 void AVR_InterruptCandidate( uint8_t u8Vector_ )
00047 {
00048     // Interrupts are prioritized by index -- lower == higher priority.
00049     // Candidate is the lowest
00050     if (u8Vector_ < stCPU.u8IntPriority)
00051     {
00052         stCPU.u8IntPriority = u8Vector_;
00053     }
00054     stCPU.u32IntFlags |= (1 << u8Vector_);
00055 }
00056
00057 //------------------------------------------------------------------------------
00058 void AVR_ClearCandidate( uint8_t u8Vector_ )
00059 {
00060     stCPU.u32IntFlags &= ~(1 << u8Vector_ );
00061     AVR_NextInterrupt();
00062 }
00063
00064
00065 //------------------------------------------------------------------------------
00066 void AVR_Interrupt( void )
00067 {
00068     // First - check to see if there's an interrupt pending.
00069     if (stCPU.u8IntPriority == 255 || stCPU.pstRAM->stRegisters.SREG.I == 0)
00070     {
00071         return; // no interrupt pending
00072     }
00073
00074     // Push the current PC to stack.
00075     uint16_t u16SP = (((uint16_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
00076                      (((uint16_t)stCPU.pstRAM->stRegisters.SPL.r));
00077
00078     uint16_t u16StoredPC = stCPU.u16PC;
00079
00080     stCPU.pstRAM->au8RAM[ u16SP ]     = (uint8_t)(u16StoredPC & 0x00FF);
00081     stCPU.pstRAM->au8RAM[ u16SP - 1 ] = (uint8_t)(u16StoredPC >> 8);
00082
00083     // Stack is post-decremented
00084     u16SP -= 2;
00085
00086     // Store the new SP.
00087     stCPU.pstRAM->stRegisters.SPH.r = (u16SP >> 8);
00088     stCPU.pstRAM->stRegisters.SPL.r = (u16SP & 0x00FF);
00089
00090     // Read the new PC from the vector table
00091     uint16_t u16NewPC = (uint16_t)(stCPU.u8IntPriority * 2);
00092
00093     // Set the new PC
00094     stCPU.u16PC = u16NewPC;
00095     stCPU.u16ExtraPC = 0;
00096
00097     // Clear the "I" (global interrupt enabled) register in the SR
00098     stCPU.pstRAM->stRegisters.SREG.I = 0;
00099
00100     // Run the interrupt-acknowledge callback associated with this vector
00101     if (stCPU.u8IntPriority < 32 && stCPU.apfInterruptCallbacks[ stCPU.u8IntPriority ])
00102     {
00103         stCPU.apfInterruptCallbacks[ stCPU.u8IntPriority ]( stCPU.u8IntPriority );
00104     }
00105
00106     // Reset the CPU interrupt priority
00107     stCPU.u32IntFlags &= ~(1 << stCPU.u8IntPriority);
00108     AVR_NextInterrupt();
00109
00110     // Clear any sleep-mode flags currently set
00111     stCPU.bAsleep = false;
00112 }
```

4.17 avr_interrupt.h File Reference

AVR CPU Interrupt management - functionality responsible for arming/ disarming/processing CPU interrupts generated during the course of normal operation.

```
#include <stdint.h>
#include "emu_config.h"
#include "avr_cpu.h"
```

Functions

- void AVR_InterruptCandidate (uint8_t u8Vector_)

  AVR_InterruptCandidate.
- void AVR_ClearCandidate (uint8_t u8Vector_)

  AVR_ClearCandidate.
- void AVR_Interrupt (void)

  AVR_Interrupt.

4.17.1 Detailed Description

AVR CPU Interrupt management - functionality responsible for arming/ disarming/processing CPU interrupts generated during the course of normal operation.

Definition in file avr_interrupt.h.

4.17.2 Function Documentation

4.17.2.1 void AVR_ClearCandidate (uint8_t *u8Vector_*)

AVR_ClearCandidate.

Parameters

*u8Vector_*	Vector to clear pending interrupt for.

Definition at line 58 of file avr_interrupt.c.

4.17.2.2 void AVR_Interrupt (void)

AVR_Interrupt.

Entrypoint for CPU interrupts. Stop executing the currently-executing code, push the current PC to the stack, disable interrupts, and resume execution at the new location specified in the vector table.

Definition at line 66 of file avr_interrupt.c.

4.17.2.3 void AVR_InterruptCandidate (uint8_t *u8Vector_*)

AVR_InterruptCandidate.

Given an existing interrupt candidate, determine if the selected interrupt vector is of highier priority. If higher priority, update the candidate.

Parameters

*u8Vector_*	- Candidate interrupt vector.

Definition at line 46 of file avr_interrupt.c.

4.18 avr_interrupt.h

```
00001 /*****************************************************************************
00002  *      (     (                        (     |
00003  *    )\ )  )\ )     (               )\ )   |
00004  *   (()/( (()/(     )\      (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                         | "Yeah, it does Arduino..."
00011  * ----------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00023 #ifndef __AVR_INTERRUPT_H__
00024 #define __AVR_INTERRUPT_H__
00025
00026 #include <stdint.h>
00027 #include "emu_config.h"
00028 #include "avr_cpu.h"
00029
00030 //-----------------------------------------------------------------------------
00039 void AVR_InterruptCandidate( uint8_t u8Vector_ );
00040
00041 //-----------------------------------------------------------------------------
00047 void AVR_ClearCandidate( uint8_t u8Vector_ );
00048
00049 //-----------------------------------------------------------------------------
00058 void AVR_Interrupt( void );
00059
00060 #endif //__AVR_INTERRUPT_H__
```

4.19 avr_io.c File Reference

Interface to connect I/O register updates to their corresponding peripheral plugins.

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include "emu_config.h"
#include "avr_peripheral.h"
#include "avr_cpu.h"
#include "avr_io.h"
```

Functions

- void IO_AddReader (AVRPeripheral *pstPeriph_, uint8_t addr_)

  IO_AddReader.
- void IO_AddWriter (AVRPeripheral *pstPeriph_, uint8_t addr_)

  IO_AddWriter.
- void IO_AddClocker (AVRPeripheral *pstPeriph_)

  IO_AddClocker.
- void IO_Write (uint8_t addr_, uint8_t value_)

  IO_Write.
- void IO_Read (uint8_t addr_, uint8_t *value_)

IO_Read.

- void IO_Clock (void)

  IO_Clock.

4.19.1 Detailed Description

Interface to connect I/O register updates to their corresponding peripheral plugins.

Definition in file avr_io.c.

4.19.2 Function Documentation

4.19.2.1 void IO_AddClocker (AVRPeripheral * *pstPeriph_*)

IO_AddClocker.

Parameters

*pstPeriph_*	

Definition at line 69 of file avr_io.c.

4.19.2.2 void IO_AddReader (AVRPeripheral * *pstPeriph_*, uint8_t *addr_*)

IO_AddReader.

Parameters

*pstPeriph_*	
*addr_*	

Definition at line 33 of file avr_io.c.

4.19.2.3 void IO_AddWriter (AVRPeripheral * *pstPeriph_*, uint8_t *addr_*)

IO_AddWriter.

Parameters

*pstPeriph_*	
*addr_*	

Definition at line 51 of file avr_io.c.

4.19.2.4 void IO_Clock (void)

IO_Clock.

Definition at line 115 of file avr_io.c.

4.19.2.5 void IO_Read (uint8_t *addr_*, uint8_t * *value_*)

IO_Read.

Parameters

*addr_*	
*value_*	

Definition at line 101 of file avr_io.c.

4.19.2.6 void IO_Write (uint8_t *addr_*, uint8_t *value_*)

IO_Write.

Parameters

*addr_*	
*value_*	

Definition at line 87 of file avr_io.c.

4.20 avr_io.c

```
00001 /*****************************************************************************
00002  *      (     (                       (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\      (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+-----------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00022 #include <stdint.h>
00023 #include <stdio.h>
00024 #include <stdlib.h>
00025
00026 #include "emu_config.h"
00027
00028 #include "avr_peripheral.h"
00029 #include "avr_cpu.h"
00030 #include "avr_io.h"
00031
00032 //-----------------------------------------------------------------------------
00033 void IO_AddReader(  AVRPeripheral *pstPeriph_, uint8_t addr_)
00034 {
00035     IOReaderList *node = NULL;
00036
00037     node = (IOReaderList*)malloc(sizeof(*node));
00038     if (!node)
00039     {
00040         return;
00041     }
00042
00043     node->next = stCPU.apstPeriphReadTable[addr_];
00044     node->pfReader = pstPeriph_->pfRead;
00045     node->pvContext = pstPeriph_->pvContext;
00046
00047     stCPU.apstPeriphReadTable[addr_] = node;
00048 }
00049
00050 //-----------------------------------------------------------------------------
00051 void IO_AddWriter(  AVRPeripheral *pstPeriph_, uint8_t addr_)
00052 {
00053     IOWriterList *node = NULL;
00054
00055     node = (IOWriterList*)malloc(sizeof(*node));
00056     if (!node)
00057     {
00058         return;
00059     }
00060
00061     node->next = stCPU.apstPeriphWriteTable[addr_];
00062     node->pfWriter = pstPeriph_->pfWrite;
00063     node->pvContext = pstPeriph_->pvContext;
00064
```

```
00065     stCPU.apstPeriphWriteTable[addr_] = node;
00066 }
00067
00068 //-----------------------------------------------------------------------------
00069 void IO_AddClocker(  AVRPeripheral *pstPeriph_ )
00070 {
00071     IOClockList *node = NULL;
00072
00073     node = (IOClockList*)malloc(sizeof(*node));
00074     if (!node)
00075     {
00076         return;
00077     }
00078
00079     node->next = stCPU.pstClockList;
00080     node->pfClock = pstPeriph_->pfClock;
00081     node->pvContext = pstPeriph_->pvContext;
00082
00083     stCPU.pstClockList = node;
00084 }
00085
00086 //-----------------------------------------------------------------------------
00087 void IO_Write(  uint8_t addr_, uint8_t value_ )
00088 {
00089     IOWriterList *node = stCPU.apstPeriphWriteTable[addr_];
00090     while (node)
00091     {
00092         if (node->pfWriter)
00093         {
00094             node->pfWriter( node->pvContext, addr_, value_ );
00095         }
00096         node = node->next;
00097     }
00098 }
00099
00100 //-----------------------------------------------------------------------------
00101 void IO_Read(  uint8_t addr_, uint8_t *value_ )
00102 {
00103     IOReaderList *node = stCPU.apstPeriphReadTable[addr_];
00104     while (node)
00105     {
00106         if (node->pfReader)
00107         {
00108             node->pfReader( node->pvContext, addr_, value_ );
00109         }
00110         node = node->next;
00111     }
00112 }
00113
00114 //-----------------------------------------------------------------------------
00115 void IO_Clock( void )
00116 {
00117     IOClockList *node = stCPU.pstClockList;
00118     while (node)
00119     {
00120         if (node->pfClock)
00121         {
00122             node->pfClock( node->pvContext );
00123         }
00124         node = node->next;
00125     }
00126 }
```

4.21 avr_io.h File Reference

Interface to connect I/O register updates to their corresponding peripheral plugins.

```
#include "avr_peripheral.h"
```

Data Structures

- struct _IOReaderList
- struct _IOWriterList
- struct _IOClockList

Typedefs

- typedef struct _IOReaderList **IOReaderList**
- typedef struct _IOWriterList **IOWriterList**
- typedef struct _IOClockList **IOClockList**

Functions

- void IO_AddReader (AVRPeripheral *pstPeriph_, uint8_t addr_)
  IO_AddReader.
- void IO_AddWriter (AVRPeripheral *pstPeriph_, uint8_t addr_)
  IO_AddWriter.
- void IO_AddClocker (AVRPeripheral *pstPeriph_)
  IO_AddClocker.
- void IO_Write (uint8_t addr_, uint8_t value_)
  IO_Write.
- void IO_Read (uint8_t addr_, uint8_t *value_)
  IO_Read.
- void IO_Clock (void)
  IO_Clock.

4.21.1 Detailed Description

Interface to connect I/O register updates to their corresponding peripheral plugins.

Definition in file avr_io.h.

4.21.2 Function Documentation

4.21.2.1 void IO_AddClocker (AVRPeripheral * *pstPeriph_*)

IO_AddClocker.

Parameters

*pstPeriph_*	

Definition at line 69 of file avr_io.c.

4.21.2.2 void IO_AddReader (AVRPeripheral * *pstPeriph_*, uint8_t *addr_*)

IO_AddReader.

Parameters

*pstPeriph_*	
*addr_*	

Definition at line 33 of file avr_io.c.

4.21.2.3 void IO_AddWriter (AVRPeripheral * *pstPeriph_*, uint8_t *addr_*)

IO_AddWriter.

Parameters

*pstPeriph_*	
*addr_*	

Definition at line 51 of file avr_io.c.

4.21.2.4 void IO_Clock (void)

IO_Clock.

Definition at line 115 of file avr_io.c.

4.21.2.5 void IO_Read (uint8_t *addr_*, uint8_t * *value_*)

IO_Read.

Parameters

*addr_*	
*value_*	

Definition at line 101 of file avr_io.c.

4.21.2.6 void IO_Write (uint8_t *addr_*, uint8_t *value_*)

IO_Write.

Parameters

*addr_*	
*value_*	

Definition at line 87 of file avr_io.c.

4.22 avr_io.h

```
00001 /*****************************************************************************
00002  *      (     (                        (     |
00003  *    )\ )  )\ )     (              )\ )    |
00004  *   (()/( (()/(     )\      (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ----------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] ------------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] ------------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+-----------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00022 #ifndef __AVR_IO_H__
00023 #define __AVR_IO_H__
00024
00025 #include "avr_peripheral.h"
00026
00027 //-----------------------------------------------------------------------------
00028 typedef struct _IOReaderList
00029 {
00030     struct _IOReaderList *next;
00031     void *pvContext;
00032     PeriphRead pfReader;
00033 } IOReaderList;
00034
00035 //-----------------------------------------------------------------------------
00036 typedef struct _IOWriterList
00037 {
00038     struct _IOWriterList *next;
00039     void *pvContext;
```

```
00040     PeriphWrite pfWriter;
00041 } IOWriterList;
00042
00043 //-----------------------------------------------------------------------------
00044 typedef struct _IOClockList
00045 {
00046     struct _IOClockList *next;
00047     void *pvContext;
00048     PeriphClock pfClock;
00049 } IOClockList;
00050
00051 //-----------------------------------------------------------------------------
00058 void IO_AddReader(  AVRPeripheral *pstPeriph_, uint8_t addr_);
00059
00060 //-----------------------------------------------------------------------------
00067 void IO_AddWriter(  AVRPeripheral *pstPeriph_, uint8_t addr_);
00068
00069 //-----------------------------------------------------------------------------
00075 void IO_AddClocker(  AVRPeripheral *pstPeriph_ );
00076
00077 //-----------------------------------------------------------------------------
00084 void IO_Write(  uint8_t addr_, uint8_t value_ );
00085
00086 //-----------------------------------------------------------------------------
00093 void IO_Read(  uint8_t addr_, uint8_t *value_ );
00094
00095 //-----------------------------------------------------------------------------
00100 void IO_Clock( void );
00101
00102 #endif
```

4.23 avr_loader.c File Reference

Functions to load intel-formatted programming files into a virtual AVR.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/fcntl.h>
#include "emu_config.h"
#include "avr_cpu.h"
#include "intel_hex.h"
```

Functions

- static void **AVR_Copy_Record** (HEX_Record_t *pstHex_)
- bool AVR_Load_HEX (const char *szFilePath_)

  AVR_Load_HEX Load a hex file, specified by path, into the flash memory of the CPU object.

4.23.1 Detailed Description

Functions to load intel-formatted programming files into a virtual AVR.

Definition in file avr_loader.c.

4.23.2 Function Documentation

4.23.2.1 bool AVR_Load_HEX (const char * *szFilePath_*)

AVR_Load_HEX Load a hex file, specified by path, into the flash memory of the CPU object.

Parameters

*szFilePath_*	Pointer to the hexfile path

Returns

true if the hex file load operation succeeded, false otherwise

Definition at line 48 of file avr_loader.c.

4.24 avr_loader.c

```
00001 /*****************************************************************************
00002  *      (     (                          (     |
00003  *    )\ )  )\ )     (               )\ )    |
00004  *   (()/( (()/(     )\      (   (  (()/(    | -- [ Funkenstein ] ------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))       | -- [ Litle ] -------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] --------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \     | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \    \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\    \_/   |_|_\    |
00010  *                                          | "Yeah, it does Arduino..."
00011  * -----------------------------------------+-----------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #include <stdio.h>
00022 #include <stdlib.h>
00023 #include <string.h>
00024 #include <sys/stat.h>
00025 #include <sys/fcntl.h>
00026
00027 #include "emu_config.h"
00028
00029 #include "avr_cpu.h"
00030 #include "intel_hex.h"
00031
00032 //-----------------------------------------------------------------------------
00033 static void AVR_Copy_Record( HEX_Record_t *pstHex_)
00034 {
00035     uint16_t u16Data;
00036     uint16_t i;
00037     for (i = 0; i < pstHex_->u8ByteCount; i += 2)
00038     {
00039         u16Data = pstHex_->u8Data[i+1];
00040         u16Data <<= 8;
00041         u16Data |= pstHex_->u8Data[i];
00042
00043         stCPU.pu16ROM[(pstHex_->u16Address + i) >> 1] = u16Data;
00044     }
00045 }
00046
00047 //-----------------------------------------------------------------------------
00048 bool AVR_Load_HEX( const char *szFilePath_)
00049 {
00050     HEX_Record_t stRecord;
00051     uint32_t u32Addr = 0;
00052     int fd = -1;
00053
00054     if (!szFilePath_)
00055     {
00056         fprintf(stderr, "No programming file specified\n");
00057         return false;
00058     }
00059
00060     fd = open(szFilePath_, O_RDONLY);
00061
00062     if (-1 == fd)
00063     {
00064         fprintf(stderr, "Unable to open file\n");
00065         return false;
00066     }
00067
00068     bool rc = true;
00069
00070     while (rc)
00071     {
00072         rc = HEX_Read_Record(fd, &stRecord);
00073         if (RECORD_EOF == stRecord.u8RecordType)
00074         {
```

```
00075             break;
00076         }
00077         if (RECORD_DATA == stRecord.u8RecordType)
00078         {
00079             AVR_Copy_Record(&stRecord);
00080         }
00081     }
00082
00083 cleanup:
00084     close(fd);
00085     return rc;
00086 }
```

4.25 avr_loader.h File Reference

Functions to load intel-formatted programming files into a virtual AVR.

```
#include <stdint.h>
#include "avr_cpu.h"
```

Functions

- bool AVR_Load_HEX (const char *szFilePath_)

  AVR_Load_HEX Load a hex file, specified by path, into the flash memory of the CPU object.

4.25.1 Detailed Description

Functions to load intel-formatted programming files into a virtual AVR.

Definition in file avr_loader.h.

4.25.2 Function Documentation

4.25.2.1 bool AVR_Load_HEX (const char * *szFilePath_*)

AVR_Load_HEX Load a hex file, specified by path, into the flash memory of the CPU object.

Parameters

*szFilePath_*	Pointer to the hexfile path

Returns

true if the hex file load operation succeeded, false otherwise

Definition at line 48 of file avr_loader.c.

4.26 avr_loader.h

```
00001 /*****************************************************************************
00002  *      (     (                        (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\      (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\   )\  /(_))     | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ----------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] ------------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] ------------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+-----------------------------------
```

```
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  ************************************************************************/
00021 #ifndef __AVR_LOADER_H__
00022 #define __AVR_LOADER_H__
00023
00024 #include <stdint.h>
00025 #include "avr_cpu.h"
00026
00027 //------------------------------------------------------------------------
00035 bool AVR_Load_HEX( const char *szFilePath_);
00036
00037 #endif
```

4.27 avr_op_cycles.c File Reference

Opcode cycle counting functions.

```
#include <stdint.h>
#include <stdio.h>
#include "emu_config.h"
#include "avr_op_decode.h"
#include "avr_opcodes.h"
#include "avr_op_size.h"
#include "avr_cpu.h"
#include "avr_cpu_print.h"
#include "avr_loader.h"
```

Functions

- static uint8_t **AVR_Opcode_Cycles_ADD** ()
- static uint8_t **AVR_Opcode_Cycles_ADC** ()
- static uint8_t **AVR_Opcode_Cycles_ADIW** ()
- static uint8_t **AVR_Opcode_Cycles_SUB** ()
- static uint8_t **AVR_Opcode_Cycles_SUBI** ()
- static uint8_t **AVR_Opcode_Cycles_SBC** ()
- static uint8_t **AVR_Opcode_Cycles_SBCI** ()
- static uint8_t **AVR_Opcode_Cycles_SBIW** ()
- static uint8_t **AVR_Opcode_Cycles_AND** ()
- static uint8_t **AVR_Opcode_Cycles_ANDI** ()
- static uint8_t **AVR_Opcode_Cycles_OR** ()
- static uint8_t **AVR_Opcode_Cycles_ORI** ()
- static uint8_t **AVR_Opcode_Cycles_EOR** ()
- static uint8_t **AVR_Opcode_Cycles_COM** ()
- static uint8_t **AVR_Opcode_Cycles_NEG** ()
- static uint8_t **AVR_Opcode_Cycles_SBR** ()
- static uint8_t **AVR_Opcode_Cycles_CBR** ()
- static uint8_t **AVR_Opcode_Cycles_INC** ()
- static uint8_t **AVR_Opcode_Cycles_DEC** ()
- static uint8_t **AVR_Opcode_Cycles_TST** ()
- static uint8_t **AVR_Opcode_Cycles_CLR** ()
- static uint8_t **AVR_Opcode_Cycles_SER** ()
- static uint8_t **AVR_Opcode_Cycles_MUL** ()
- static uint8_t **AVR_Opcode_Cycles_MULS** ()
- static uint8_t **AVR_Opcode_Cycles_MULSU** ()
- static uint8_t **AVR_Opcode_Cycles_FMUL** ()
- static uint8_t **AVR_Opcode_Cycles_FMULS** ()

- static uint8_t **AVR_Opcode_Cycles_FMULSU** ()
- static uint8_t **AVR_Opcode_Cycles_DES** ()
- static uint8_t **AVR_Opcode_Cycles_RJMP** ()
- static uint8_t **AVR_Opcode_Cycles_IJMP** ()
- static uint8_t **AVR_Opcode_Cycles_EIJMP** ()
- static uint8_t **AVR_Opcode_Cycles_JMP** ()
- static uint8_t AVR_Opcode_Cycles_RCALL ()
- static uint8_t AVR_Opcode_Cycles_ICALL ()
- static uint8_t **AVR_Opcode_Cycles_EICALL** ()
- static uint8_t AVR_Opcode_Cycles_CALL ()
- static uint8_t AVR_Opcode_Cycles_RET ()
- static uint8_t AVR_Opcode_Cycles_RETI ()
- static uint8_t **AVR_Opcode_Cycles_CPSE** ()
- static uint8_t **AVR_Opcode_Cycles_CP** ()
- static uint8_t **AVR_Opcode_Cycles_CPC** ()
- static uint8_t **AVR_Opcode_Cycles_CPI** ()
- static uint8_t **AVR_Opcode_Cycles_SBRC** ()
- static uint8_t **AVR_Opcode_Cycles_SBRS** ()
- static uint8_t **AVR_Opcode_Cycles_SBIC** ()
- static uint8_t **AVR_Opcode_Cycles_SBIS** ()
- static uint8_t **AVR_Opcode_Cycles_BRBS** ()
- static uint8_t **AVR_Opcode_Cycles_BRBC** ()
- static uint8_t **AVR_Opcode_Cycles_BREQ** ()
- static uint8_t **AVR_Opcode_Cycles_BRNE** ()
- static uint8_t **AVR_Opcode_Cycles_BRCS** ()
- static uint8_t **AVR_Opcode_Cycles_BRCC** ()
- static uint8_t **AVR_Opcode_Cycles_BRSH** ()
- static uint8_t **AVR_Opcode_Cycles_BRLO** ()
- static uint8_t **AVR_Opcode_Cycles_BRMI** ()
- static uint8_t **AVR_Opcode_Cycles_BRPL** ()
- static uint8_t **AVR_Opcode_Cycles_BRGE** ()
- static uint8_t **AVR_Opcode_Cycles_BRLT** ()
- static uint8_t **AVR_Opcode_Cycles_BRHS** ()
- static uint8_t **AVR_Opcode_Cycles_BRHC** ()
- static uint8_t **AVR_Opcode_Cycles_BRTS** ()
- static uint8_t **AVR_Opcode_Cycles_BRTC** ()
- static uint8_t **AVR_Opcode_Cycles_BRVS** ()
- static uint8_t **AVR_Opcode_Cycles_BRVC** ()
- static uint8_t **AVR_Opcode_Cycles_BRIE** ()
- static uint8_t **AVR_Opcode_Cycles_BRID** ()
- static uint8_t **AVR_Opcode_Cycles_MOV** ()
- static uint8_t **AVR_Opcode_Cycles_MOVW** ()
- static uint8_t **AVR_Opcode_Cycles_LDI** ()
- static uint8_t **AVR_Opcode_Cycles_LDS** ()
- static uint8_t **AVR_Opcode_Cycles_LD_X_Indirect** ()
- static uint8_t **AVR_Opcode_Cycles_LD_X_Indirect_Postinc** ()
- static uint8_t **AVR_Opcode_Cycles_LD_X_Indirect_Predec** ()
- static uint8_t **AVR_Opcode_Cycles_LD_Y_Indirect** ()
- static uint8_t **AVR_Opcode_Cycles_LD_Y_Indirect_Postinc** ()
- static uint8_t **AVR_Opcode_Cycles_LD_Y_Indirect_Predec** ()
- static uint8_t **AVR_Opcode_Cycles_LDD_Y** ()
- static uint8_t **AVR_Opcode_Cycles_LD_Z_Indirect** ()
- static uint8_t AVR_Opcode_Cycles_LD_Z_Indirect_Postinc ()
- static uint8_t AVR_Opcode_Cycles_LD_Z_Indirect_Predec ()
- static uint8_t **AVR_Opcode_Cycles_LDD_Z** ()

- static uint8_t **AVR_Opcode_Cycles_STS** ()
- static uint8_t AVR_Opcode_Cycles_ST_X_Indirect ()
- static uint8_t AVR_Opcode_Cycles_ST_X_Indirect_Postinc ()
- static uint8_t AVR_Opcode_Cycles_ST_X_Indirect_Predec ()
- static uint8_t AVR_Opcode_Cycles_ST_Y_Indirect ()
- static uint8_t AVR_Opcode_Cycles_ST_Y_Indirect_Postinc ()
- static uint8_t AVR_Opcode_Cycles_ST_Y_Indirect_Predec ()
- static uint8_t AVR_Opcode_Cycles_STD_Y ()
- static uint8_t AVR_Opcode_Cycles_ST_Z_Indirect ()
- static uint8_t AVR_Opcode_Cycles_ST_Z_Indirect_Postinc ()
- static uint8_t AVR_Opcode_Cycles_ST_Z_Indirect_Predec ()
- static uint8_t AVR_Opcode_Cycles_STD_Z ()
- static uint8_t **AVR_Opcode_Cycles_LPM** ()
- static uint8_t **AVR_Opcode_Cycles_LPM_Z** ()
- static uint8_t **AVR_Opcode_Cycles_LPM_Z_Postinc** ()
- static uint8_t **AVR_Opcode_Cycles_ELPM** ()
- static uint8_t **AVR_Opcode_Cycles_ELPM_Z** ()
- static uint8_t **AVR_Opcode_Cycles_ELPM_Z_Postinc** ()
- static uint8_t AVR_Opcode_Cycles_SPM ()
- static uint8_t AVR_Opcode_Cycles_SPM_Z_Postinc2 ()
- static uint8_t **AVR_Opcode_Cycles_IN** ()
- static uint8_t **AVR_Opcode_Cycles_OUT** ()
- static uint8_t **AVR_Opcode_Cycles_LAC** ()
- static uint8_t **AVR_Opcode_Cycles_LAS** ()
- static uint8_t **AVR_Opcode_Cycles_LAT** ()
- static uint8_t **AVR_Opcode_Cycles_LSL** ()
- static uint8_t **AVR_Opcode_Cycles_LSR** ()
- static uint8_t **AVR_Opcode_Cycles_POP** ()
- static uint8_t **AVR_Opcode_Cycles_PUSH** ()
- static uint8_t **AVR_Opcode_Cycles_ROL** ()
- static uint8_t **AVR_Opcode_Cycles_ROR** ()
- static uint8_t **AVR_Opcode_Cycles_ASR** ()
- static uint8_t **AVR_Opcode_Cycles_SWAP** ()
- static uint8_t **AVR_Opcode_Cycles_BSET** ()
- static uint8_t **AVR_Opcode_Cycles_BCLR** ()
- static uint8_t AVR_Opcode_Cycles_SBI ()
- static uint8_t AVR_Opcode_Cycles_CBI ()
- static uint8_t **AVR_Opcode_Cycles_BST** ()
- static uint8_t **AVR_Opcode_Cycles_BLD** ()
- static uint8_t **AVR_Opcode_Cycles_SEC** ()
- static uint8_t **AVR_Opcode_Cycles_CLC** ()
- static uint8_t **AVR_Opcode_Cycles_SEN** ()
- static uint8_t **AVR_Opcode_Cycles_CLN** ()
- static uint8_t **AVR_Opcode_Cycles_SEZ** ()
- static uint8_t **AVR_Opcode_Cycles_CLZ** ()
- static uint8_t **AVR_Opcode_Cycles_SEI** ()
- static uint8_t **AVR_Opcode_Cycles_CLI** ()
- static uint8_t **AVR_Opcode_Cycles_SES** ()
- static uint8_t **AVR_Opcode_Cycles_CLS** ()
- static uint8_t **AVR_Opcode_Cycles_SEV** ()
- static uint8_t **AVR_Opcode_Cycles_CLV** ()
- static uint8_t **AVR_Opcode_Cycles_SET** ()
- static uint8_t **AVR_Opcode_Cycles_CLT** ()
- static uint8_t **AVR_Opcode_Cycles_SEH** ()
- static uint8_t **AVR_Opcode_Cycles_CLH** ()

- static uint8_t **AVR_Opcode_Cycles_BREAK** ()
- static uint8_t **AVR_Opcode_Cycles_NOP** ()
- static uint8_t **AVR_Opcode_Cycles_SLEEP** ()
- static uint8_t **AVR_Opcode_Cycles_WDR** ()
- static uint8_t **AVR_Opcode_Cycles_XCH** ()
- static uint8_t **AVR_Opcode_Cycles_Unimplemented** ()
- uint8_t AVR_Opcode_Cycles (uint16_t OP_)

  AVR_Opocde_Cycles.

4.27.1 Detailed Description

Opcode cycle counting functions.

Definition in file avr_op_cycles.c.

4.27.2 Function Documentation

4.27.2.1 uint8_t AVR_Opcode_Cycles (uint16_t *OP_*)

AVR_Opocde_Cycles.

Parameters

*OP_*	Opcode to compute the minimum cycles to execute for

Returns

The minimum number of cycles it will take to execute an opcode

Definition at line 892 of file avr_op_cycles.c.

4.27.2.2 static uint8_t AVR_Opcode_Cycles_CALL ( ) [static]

! ToDo – 5 cycles on devices w/22-bit PC

Definition at line 250 of file avr_op_cycles.c.

4.27.2.3 static uint8_t AVR_Opcode_Cycles_CBI ( ) [static]

! ToDo - take into account XMEGA/tinyAVR timing

Definition at line 742 of file avr_op_cycles.c.

4.27.2.4 static uint8_t AVR_Opcode_Cycles_ICALL ( ) [static]

! ToDo – n cycles on devices w/22-bit PC

Definition at line 238 of file avr_op_cycles.c.

4.27.2.5 static uint8_t AVR_Opcode_Cycles_LD_Z_Indirect_Postinc ( ) [static]

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 508 of file avr_op_cycles.c.

4.27.2.6 static uint8_t AVR_Opcode_Cycles_LD_Z_Indirect_Predec ( ) [static]

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 514 of file avr_op_cycles.c.

4.27.2.7 static uint8_t AVR_Opcode_Cycles_RCALL ( ) [static]

! ToDo – n cycles on devices w/22-bit PC

Definition at line 232 of file avr_op_cycles.c.

4.27.2.8 static uint8_t AVR_Opcode_Cycles_RET ( ) [static]

! ToDo – 5 cycles on devices w/22-bit PC

Definition at line 256 of file avr_op_cycles.c.

4.27.2.9 static uint8_t AVR_Opcode_Cycles_RETI ( ) [static]

! ToDo – 5 cycles on devices w/22-bit PC

Definition at line 262 of file avr_op_cycles.c.

4.27.2.10 static uint8_t AVR_Opcode_Cycles_SBI ( ) [static]

! ToDo - take into account XMEGA/tinyAVR timing

Definition at line 736 of file avr_op_cycles.c.

4.27.2.11 static uint8_t AVR_Opcode_Cycles_SPM ( ) [static]

!ToDo - Datasheet says "Depends on the operation"...

Definition at line 634 of file avr_op_cycles.c.

4.27.2.12 static uint8_t AVR_Opcode_Cycles_SPM_Z_Postinc2 ( ) [static]

!ToDo - Datasheet says "Depends on the operation"...

Definition at line 640 of file avr_op_cycles.c.

4.27.2.13 static uint8_t AVR_Opcode_Cycles_ST_X_Indirect ( ) [static]

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 532 of file avr_op_cycles.c.

4.27.2.14 static uint8_t AVR_Opcode_Cycles_ST_X_Indirect_Postinc ( ) [static]

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 538 of file avr_op_cycles.c.

4.27.2.15 static uint8_t AVR_Opcode_Cycles_ST_X_Indirect_Predec ( ) [static]

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 544 of file avr_op_cycles.c.

4.27.2.16 static uint8_t AVR_Opcode_Cycles_ST_Y_Indirect ( ) [static]

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 550 of file avr_op_cycles.c.

4.27.2.17 static uint8_t AVR_Opcode_Cycles_ST_Y_Indirect_Postinc ( ) [static]

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 556 of file avr_op_cycles.c.

4.27.2.18 static uint8_t AVR_Opcode_Cycles_ST_Y_Indirect_Predec ( ) [static]

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 562 of file avr_op_cycles.c.

4.27.2.19 static uint8_t AVR_Opcode_Cycles_ST_Z_Indirect ( ) [static]

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 574 of file avr_op_cycles.c.

4.27.2.20 static uint8_t AVR_Opcode_Cycles_ST_Z_Indirect_Postinc ( ) [static]

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 580 of file avr_op_cycles.c.

4.27.2.21 static uint8_t AVR_Opcode_Cycles_ST_Z_Indirect_Predec ( ) [static]

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 586 of file avr_op_cycles.c.

4.27.2.22 static uint8_t AVR_Opcode_Cycles_STD_Y ( ) [static]

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 568 of file avr_op_cycles.c.

4.27.2.23 static uint8_t AVR_Opcode_Cycles_STD_Z ( ) [static]

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 592 of file avr_op_cycles.c.

4.28 avr_op_cycles.c

```
00001 /*****************************************************************************
00002  *      (     (                          (      |
00003  *    )\ )  )\ )     (               )\ )    |
00004  *   (()/( (()/(    )\      (   (  (()/(    | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                         | "Yeah, it does Arduino..."
00011  * ----------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #include <stdint.h>
00022 #include <stdio.h>
00023
00024 #include "emu_config.h"
00025
00026 #include "avr_op_decode.h"
00027 #include "avr_opcodes.h"
00028 #include "avr_op_size.h"
00029 #include "avr_cpu.h"
00030 #include "avr_cpu_print.h"
00031 #include "avr_loader.h"
00032
00033 //-----------------------------------------------------------------------------
00034 static uint8_t AVR_Opcode_Cycles_ADD()
00035 {
00036     return 1;
00037 }
00038
00039 //-----------------------------------------------------------------------------
00040 static uint8_t AVR_Opcode_Cycles_ADC()
00041 {
00042     return 1;
00043 }
00044
00045 //-----------------------------------------------------------------------------
00046 static uint8_t AVR_Opcode_Cycles_ADIW()
00047 {
00048     return 2;
00049 }
00050
00051 //-----------------------------------------------------------------------------
00052 static uint8_t AVR_Opcode_Cycles_SUB()
00053 {
00054     return 1;
00055 }
00056
00057 //-----------------------------------------------------------------------------
00058 static uint8_t AVR_Opcode_Cycles_SUBI()
00059 {
00060     return 1;
00061 }
00062
00063 //-----------------------------------------------------------------------------
00064 static uint8_t AVR_Opcode_Cycles_SBC()
00065 {
00066     return 1;
00067 }
00068
00069 //-----------------------------------------------------------------------------
00070 static uint8_t AVR_Opcode_Cycles_SBCI()
00071 {
00072     return 1;
00073 }
00074
00075 //-----------------------------------------------------------------------------
00076 static uint8_t AVR_Opcode_Cycles_SBIW()
00077 {
00078     return 2;
00079 }
00080
00081 //-----------------------------------------------------------------------------
00082 static uint8_t AVR_Opcode_Cycles_AND()
00083 {
00084     return 1;
00085 }
00086
00087 //-----------------------------------------------------------------------------
00088 static uint8_t AVR_Opcode_Cycles_ANDI()
00089 {
00090     return 1;
```

```
00091 }
00092
00093 //-------------------------------------------------------------------------
00094 static uint8_t AVR_Opcode_Cycles_OR()
00095 {
00096     return 1;
00097 }
00098
00099 //-------------------------------------------------------------------------
00100 static uint8_t AVR_Opcode_Cycles_ORI()
00101 {
00102     return 1;
00103 }
00104
00105 //-------------------------------------------------------------------------
00106 static uint8_t AVR_Opcode_Cycles_EOR()
00107 {
00108     return 1;
00109 }
00110
00111 //-------------------------------------------------------------------------
00112 static uint8_t AVR_Opcode_Cycles_COM()
00113 {
00114     return 1;
00115 }
00116
00117 //-------------------------------------------------------------------------
00118 static uint8_t AVR_Opcode_Cycles_NEG()
00119 {
00120     return 1;
00121 }
00122
00123 //-------------------------------------------------------------------------
00124 static uint8_t AVR_Opcode_Cycles_SBR()
00125 {
00126     return 1;
00127 }
00128
00129 //-------------------------------------------------------------------------
00130 static uint8_t AVR_Opcode_Cycles_CBR()
00131 {
00132     return 1;
00133 }
00134
00135 //-------------------------------------------------------------------------
00136 static uint8_t AVR_Opcode_Cycles_INC()
00137 {
00138     return 1;
00139 }
00140
00141 //-------------------------------------------------------------------------
00142 static uint8_t AVR_Opcode_Cycles_DEC()
00143 {
00144     return 1;
00145 }
00146
00147 //-------------------------------------------------------------------------
00148 static uint8_t AVR_Opcode_Cycles_TST()
00149 {
00150     return 1;
00151 }
00152
00153 //-------------------------------------------------------------------------
00154 static uint8_t AVR_Opcode_Cycles_CLR()
00155 {
00156     return 1;
00157 }
00158
00159 //-------------------------------------------------------------------------
00160 static uint8_t AVR_Opcode_Cycles_SER()
00161 {
00162     return 1;
00163 }
00164
00165 //-------------------------------------------------------------------------
00166 static uint8_t AVR_Opcode_Cycles_MUL()
00167 {
00168     return 2;
00169 }
00170
00171 //-------------------------------------------------------------------------
00172 static uint8_t AVR_Opcode_Cycles_MULS()
00173 {
00174     return 2;
00175 }
00176
00177 //-------------------------------------------------------------------------
```

```
00178 static uint8_t AVR_Opcode_Cycles_MULSU()
00179 {
00180     return 2;
00181 }
00182
00183 //-----------------------------------------------------------------------------
00184 static uint8_t AVR_Opcode_Cycles_FMUL()
00185 {
00186     return 2;
00187 }
00188
00189 //-----------------------------------------------------------------------------
00190 static uint8_t AVR_Opcode_Cycles_FMULS()
00191 {
00192     return 2;
00193 }
00194
00195 //-----------------------------------------------------------------------------
00196 static uint8_t AVR_Opcode_Cycles_FMULSU()
00197 {
00198     return 2;
00199 }
00200
00201 //-----------------------------------------------------------------------------
00202 static uint8_t AVR_Opcode_Cycles_DES()
00203 {
00204     return 1;
00205 }
00206
00207 //-----------------------------------------------------------------------------
00208 static uint8_t AVR_Opcode_Cycles_RJMP()
00209 {
00210     return 2;
00211 }
00212
00213 //-----------------------------------------------------------------------------
00214 static uint8_t AVR_Opcode_Cycles_IJMP()
00215 {
00216     return 2;
00217 }
00218
00219 //-----------------------------------------------------------------------------
00220 static uint8_t AVR_Opcode_Cycles_EIJMP()
00221 {
00222     return 2;
00223 }
00224
00225 //-----------------------------------------------------------------------------
00226 static uint8_t AVR_Opcode_Cycles_JMP()
00227 {
00228     return 2;
00229 }
00230
00231 //-----------------------------------------------------------------------------
00232 static uint8_t AVR_Opcode_Cycles_RCALL()
00233 {
00234     return 3;
00235 }
00236
00237 //-----------------------------------------------------------------------------
00238 static uint8_t AVR_Opcode_Cycles_ICALL()
00239 {
00240     return 3;
00241 }
00242
00243 //-----------------------------------------------------------------------------
00244 static uint8_t AVR_Opcode_Cycles_EICALL()
00245 {
00246     return 4;
00247 }
00248
00249 //-----------------------------------------------------------------------------
00250 static uint8_t AVR_Opcode_Cycles_CALL()
00251 {
00252     return 4;
00253 }
00254
00255 //-----------------------------------------------------------------------------
00256 static uint8_t AVR_Opcode_Cycles_RET()
00257 {
00258     return 4;
00259 }
00260
00261 //-----------------------------------------------------------------------------
00262 static uint8_t AVR_Opcode_Cycles_RETI()
00263 {
00264     return 4;
```

```
00265 }
00266
00267 //-------------------------------------------------------------------------
00268 static uint8_t AVR_Opcode_Cycles_CPSE()
00269 {
00270     return 1;
00271 }
00272
00273 //-------------------------------------------------------------------------
00274 static uint8_t AVR_Opcode_Cycles_CP()
00275 {
00276     return 1;
00277 }
00278
00279 //-------------------------------------------------------------------------
00280 static uint8_t AVR_Opcode_Cycles_CPC()
00281 {
00282     return 1;
00283 }
00284
00285 //-------------------------------------------------------------------------
00286 static uint8_t AVR_Opcode_Cycles_CPI()
00287 {
00288     return 1;
00289 }
00290
00291 //-------------------------------------------------------------------------
00292 static uint8_t AVR_Opcode_Cycles_SBRC()
00293 {
00294     return 1;
00295 }
00296
00297 //-------------------------------------------------------------------------
00298 static uint8_t AVR_Opcode_Cycles_SBRS()
00299 {
00300     return 1;
00301 }
00302
00303 //-------------------------------------------------------------------------
00304 static uint8_t AVR_Opcode_Cycles_SBIC()
00305 {
00306     return 1;
00307 }
00308
00309 //-------------------------------------------------------------------------
00310 static uint8_t AVR_Opcode_Cycles_SBIS()
00311 {
00312     return 1;
00313 }
00314
00315 //-------------------------------------------------------------------------
00316 static uint8_t AVR_Opcode_Cycles_BRBS()
00317 {
00318     return 1;
00319 }
00320
00321 //-------------------------------------------------------------------------
00322 static uint8_t AVR_Opcode_Cycles_BRBC()
00323 {
00324     return 1;
00325 }
00326
00327 //-------------------------------------------------------------------------
00328 static uint8_t AVR_Opcode_Cycles_BREQ()
00329 {
00330     return 1;
00331 }
00332
00333 //-------------------------------------------------------------------------
00334 static uint8_t AVR_Opcode_Cycles_BRNE()
00335 {
00336     return 1;
00337 }
00338
00339 //-------------------------------------------------------------------------
00340 static uint8_t AVR_Opcode_Cycles_BRCS()
00341 {
00342     return 1;
00343 }
00344
00345 //-------------------------------------------------------------------------
00346 static uint8_t AVR_Opcode_Cycles_BRCC()
00347 {
00348     return 1;
00349 }
00350
00351 //-------------------------------------------------------------------------
```

```
00352 static uint8_t AVR_Opcode_Cycles_BRSH()
00353 {
00354     return 1;
00355 }
00356
00357 //-------------------------------------------------------------------------
00358 static uint8_t AVR_Opcode_Cycles_BRLO()
00359 {
00360     return 1;
00361 }
00362
00363 //-------------------------------------------------------------------------
00364 static uint8_t AVR_Opcode_Cycles_BRMI()
00365 {
00366     return 1;
00367 }
00368
00369 //-------------------------------------------------------------------------
00370 static uint8_t AVR_Opcode_Cycles_BRPL()
00371 {
00372     return 1;
00373 }
00374
00375 //-------------------------------------------------------------------------
00376 static uint8_t AVR_Opcode_Cycles_BRGE()
00377 {
00378     return 1;
00379 }
00380
00381 //-------------------------------------------------------------------------
00382 static uint8_t AVR_Opcode_Cycles_BRLT()
00383 {
00384     return 1;
00385 }
00386
00387 //-------------------------------------------------------------------------
00388 static uint8_t AVR_Opcode_Cycles_BRHS()
00389 {
00390     return 1;
00391 }
00392
00393 //-------------------------------------------------------------------------
00394 static uint8_t AVR_Opcode_Cycles_BRHC()
00395 {
00396     return 1;
00397 }
00398
00399 //-------------------------------------------------------------------------
00400 static uint8_t AVR_Opcode_Cycles_BRTS()
00401 {
00402     return 1;
00403 }
00404
00405 //-------------------------------------------------------------------------
00406 static uint8_t AVR_Opcode_Cycles_BRTC()
00407 {
00408     return 1;
00409 }
00410
00411 //-------------------------------------------------------------------------
00412 static uint8_t AVR_Opcode_Cycles_BRVS()
00413 {
00414     return 1;
00415 }
00416
00417 //-------------------------------------------------------------------------
00418 static uint8_t AVR_Opcode_Cycles_BRVC()
00419 {
00420     return 1;
00421 }
00422
00423 //-------------------------------------------------------------------------
00424 static uint8_t AVR_Opcode_Cycles_BRIE()
00425 {
00426     return 1;
00427 }
00428
00429 //-------------------------------------------------------------------------
00430 static uint8_t AVR_Opcode_Cycles_BRID()
00431 {
00432     return 1;
00433 }
00434
00435 //-------------------------------------------------------------------------
00436 static uint8_t AVR_Opcode_Cycles_MOV()
00437 {
00438     return 1;
```

```
00439 }
00440
00441 //-----------------------------------------------------------------------
00442 static uint8_t AVR_Opcode_Cycles_MOVW()
00443 {
00444     return 1;
00445 }
00446
00447 //-----------------------------------------------------------------------
00448 static uint8_t AVR_Opcode_Cycles_LDI()
00449 {
00450     return 1;
00451 }
00452
00453 //-----------------------------------------------------------------------
00454 static uint8_t AVR_Opcode_Cycles_LDS()
00455 {
00456     return 2;
00457 }
00458
00459 //-----------------------------------------------------------------------
00460 static uint8_t AVR_Opcode_Cycles_LD_X_Indirect()
00461 {
00462     return 1;
00463 }
00464
00465 //-----------------------------------------------------------------------
00466 static uint8_t AVR_Opcode_Cycles_LD_X_Indirect_Postinc()
00467 {
00468     return 2;
00469 }
00470
00471 //-----------------------------------------------------------------------
00472 static uint8_t AVR_Opcode_Cycles_LD_X_Indirect_Predec()
00473 {
00474     return 3;
00475 }
00476
00477 //-----------------------------------------------------------------------
00478 static uint8_t AVR_Opcode_Cycles_LD_Y_Indirect()
00479 {
00480     return 1;
00481 }
00482
00483 //-----------------------------------------------------------------------
00484 static uint8_t AVR_Opcode_Cycles_LD_Y_Indirect_Postinc()
00485 {
00486     return 2;
00487 }
00488
00489 //-----------------------------------------------------------------------
00490 static uint8_t AVR_Opcode_Cycles_LD_Y_Indirect_Predec()
00491 {
00492     return 3;
00493 }
00494
00495 //-----------------------------------------------------------------------
00496 static uint8_t AVR_Opcode_Cycles_LDD_Y()
00497 {
00498     return 2;
00499 }
00500
00501 //-----------------------------------------------------------------------
00502 static uint8_t AVR_Opcode_Cycles_LD_Z_Indirect()
00503 {
00504     return 1;
00505 }
00506
00507 //-----------------------------------------------------------------------
00508 static uint8_t AVR_Opcode_Cycles_LD_Z_Indirect_Postinc()
00509 {
00510     return 2;
00511 }
00512
00513 //-----------------------------------------------------------------------
00514 static uint8_t AVR_Opcode_Cycles_LD_Z_Indirect_Predec()
00515 {
00516     return 3;
00517 }
00518
00519 //-----------------------------------------------------------------------
00520 static uint8_t AVR_Opcode_Cycles_LDD_Z()
00521 {
00522     return 2;
00523 }
00524
00525 //-----------------------------------------------------------------------
```

```
00526 static uint8_t AVR_Opcode_Cycles_STS()
00527 {
00528     return 2;
00529 }
00530
00531 //-------------------------------------------------------------------------
00532 static uint8_t AVR_Opcode_Cycles_ST_X_Indirect()
00533 {
00534     return 2;
00535 }
00536
00537 //-------------------------------------------------------------------------
00538 static uint8_t AVR_Opcode_Cycles_ST_X_Indirect_Postinc()
00539 {
00540     return 2;
00541 }
00542
00543 //-------------------------------------------------------------------------
00544 static uint8_t AVR_Opcode_Cycles_ST_X_Indirect_Predec()
00545 {
00546     return 2;
00547 }
00548
00549 //-------------------------------------------------------------------------
00550 static uint8_t AVR_Opcode_Cycles_ST_Y_Indirect()
00551 {
00552     return 2;
00553 }
00554
00555 //-------------------------------------------------------------------------
00556 static uint8_t AVR_Opcode_Cycles_ST_Y_Indirect_Postinc()
00557 {
00558     return 2;
00559 }
00560
00561 //-------------------------------------------------------------------------
00562 static uint8_t AVR_Opcode_Cycles_ST_Y_Indirect_Predec()
00563 {
00564     return 2;
00565 }
00566
00567 //-------------------------------------------------------------------------
00568 static uint8_t AVR_Opcode_Cycles_STD_Y()
00569 {
00570     return 2;
00571 }
00572
00573 //-------------------------------------------------------------------------
00574 static uint8_t AVR_Opcode_Cycles_ST_Z_Indirect()
00575 {
00576     return 2;
00577 }
00578
00579 //-------------------------------------------------------------------------
00580 static uint8_t AVR_Opcode_Cycles_ST_Z_Indirect_Postinc()
00581 {
00582     return 2;
00583 }
00584
00585 //-------------------------------------------------------------------------
00586 static uint8_t AVR_Opcode_Cycles_ST_Z_Indirect_Predec()
00587 {
00588     return 2;
00589 }
00590
00591 //-------------------------------------------------------------------------
00592 static uint8_t AVR_Opcode_Cycles_STD_Z()
00593 {
00594     return 2;
00595 }
00596
00597 //-------------------------------------------------------------------------
00598 static uint8_t AVR_Opcode_Cycles_LPM()
00599 {
00600     return 3;
00601 }
00602
00603 //-------------------------------------------------------------------------
00604 static uint8_t AVR_Opcode_Cycles_LPM_Z()
00605 {
00606     return 3;
00607 }
00608
00609 //-------------------------------------------------------------------------
00610 static uint8_t AVR_Opcode_Cycles_LPM_Z_Postinc()
00611 {
00612     return 3;
```

```
00613 }
00614
00615 //-----------------------------------------------------------------------------
00616 static uint8_t AVR_Opcode_Cycles_ELPM()
00617 {
00618     return 3;
00619 }
00620
00621 //-----------------------------------------------------------------------------
00622 static uint8_t AVR_Opcode_Cycles_ELPM_Z()
00623 {
00624     return 3;
00625 }
00626
00627 //-----------------------------------------------------------------------------
00628 static uint8_t AVR_Opcode_Cycles_ELPM_Z_Postinc()
00629 {
00630     return 3;
00631 }
00632
00633 //-----------------------------------------------------------------------------
00634 static uint8_t AVR_Opcode_Cycles_SPM()
00635 {
00636     return 2;
00637 }
00638
00639 //-----------------------------------------------------------------------------
00640 static uint8_t AVR_Opcode_Cycles_SPM_Z_Postinc2()
00641 {
00642     return 2;
00643 }
00644
00645 //-----------------------------------------------------------------------------
00646 static uint8_t AVR_Opcode_Cycles_IN()
00647 {
00648     return 1;
00649 }
00650
00651 //-----------------------------------------------------------------------------
00652 static uint8_t AVR_Opcode_Cycles_OUT()
00653 {
00654     return 1;
00655 }
00656
00657 //-----------------------------------------------------------------------------
00658 static uint8_t AVR_Opcode_Cycles_LAC()
00659 {
00660     return 1;
00661 }
00662
00663 //-----------------------------------------------------------------------------
00664 static uint8_t AVR_Opcode_Cycles_LAS()
00665 {
00666     return 1;
00667 }
00668
00669 //-----------------------------------------------------------------------------
00670 static uint8_t AVR_Opcode_Cycles_LAT()
00671 {
00672     return 1;
00673 }
00674
00675 //-----------------------------------------------------------------------------
00676 static uint8_t AVR_Opcode_Cycles_LSL()
00677 {
00678     return 1;
00679 }
00680
00681 //-----------------------------------------------------------------------------
00682 static uint8_t AVR_Opcode_Cycles_LSR()
00683 {
00684     return 1;
00685 }
00686
00687 //-----------------------------------------------------------------------------
00688 static uint8_t AVR_Opcode_Cycles_POP()
00689 {
00690     return 2;
00691 }
00692
00693 //-----------------------------------------------------------------------------
00694 static uint8_t AVR_Opcode_Cycles_PUSH()
00695 {
00696     return 2;
00697 }
00698
00699 //-----------------------------------------------------------------------------
```

```
00700 static uint8_t AVR_Opcode_Cycles_ROL()
00701 {
00702     return 1;
00703 }
00704
00705 //-------------------------------------------------------------------------
00706 static uint8_t AVR_Opcode_Cycles_ROR()
00707 {
00708     return 1;
00709 }
00710
00711 //-------------------------------------------------------------------------
00712 static uint8_t AVR_Opcode_Cycles_ASR()
00713 {
00714     return 1;
00715 }
00716
00717 //-------------------------------------------------------------------------
00718 static uint8_t AVR_Opcode_Cycles_SWAP()
00719 {
00720     return 1;
00721 }
00722
00723 //-------------------------------------------------------------------------
00724 static uint8_t AVR_Opcode_Cycles_BSET()
00725 {
00726     return 1;
00727 }
00728
00729 //-------------------------------------------------------------------------
00730 static uint8_t AVR_Opcode_Cycles_BCLR()
00731 {
00732     return 1;
00733 }
00734
00735 //-------------------------------------------------------------------------
00736 static uint8_t AVR_Opcode_Cycles_SBI()
00737 {
00738     return 2;
00739 }
00740
00741 //-------------------------------------------------------------------------
00742 static uint8_t AVR_Opcode_Cycles_CBI()
00743 {
00744     return 2;
00745 }
00746
00747 //-------------------------------------------------------------------------
00748 static uint8_t AVR_Opcode_Cycles_BST()
00749 {
00750     return 1;
00751 }
00752
00753 //-------------------------------------------------------------------------
00754 static uint8_t AVR_Opcode_Cycles_BLD()
00755 {
00756     return 1;
00757 }
00758
00759 //-------------------------------------------------------------------------
00760 static uint8_t AVR_Opcode_Cycles_SEC()
00761 {
00762     return 1;
00763 }
00764
00765 //-------------------------------------------------------------------------
00766 static uint8_t AVR_Opcode_Cycles_CLC()
00767 {
00768     return 1;
00769 }
00770
00771 //-------------------------------------------------------------------------
00772 static uint8_t AVR_Opcode_Cycles_SEN()
00773 {
00774     return 1;
00775 }
00776
00777 //-------------------------------------------------------------------------
00778 static uint8_t AVR_Opcode_Cycles_CLN()
00779 {
00780     return 1;
00781 }
00782
00783 //-------------------------------------------------------------------------
00784 static uint8_t AVR_Opcode_Cycles_SEZ()
00785 {
00786     return 1;
```

```
00787 }
00788
00789 //-------------------------------------------------------------------------
00790 static uint8_t AVR_Opcode_Cycles_CLZ()
00791 {
00792     return 1;
00793 }
00794
00795 //-------------------------------------------------------------------------
00796 static uint8_t AVR_Opcode_Cycles_SEI()
00797 {
00798     return 1;
00799 }
00800
00801 //-------------------------------------------------------------------------
00802 static uint8_t AVR_Opcode_Cycles_CLI()
00803 {
00804     return 1;
00805 }
00806
00807 //-------------------------------------------------------------------------
00808 static uint8_t AVR_Opcode_Cycles_SES()
00809 {
00810     return 1;
00811 }
00812
00813 //-------------------------------------------------------------------------
00814 static uint8_t AVR_Opcode_Cycles_CLS()
00815 {
00816     return 1;
00817 }
00818
00819 //-------------------------------------------------------------------------
00820 static uint8_t AVR_Opcode_Cycles_SEV()
00821 {
00822     return 1;
00823 }
00824
00825 //-------------------------------------------------------------------------
00826 static uint8_t AVR_Opcode_Cycles_CLV()
00827 {
00828     return 1;
00829 }
00830
00831 //-------------------------------------------------------------------------
00832 static uint8_t AVR_Opcode_Cycles_SET()
00833 {
00834     return 1;
00835 }
00836
00837 //-------------------------------------------------------------------------
00838 static uint8_t AVR_Opcode_Cycles_CLT()
00839 {
00840     return 1;
00841 }
00842
00843 //-------------------------------------------------------------------------
00844 static uint8_t AVR_Opcode_Cycles_SEH()
00845 {
00846     return 1;
00847 }
00848
00849 //-------------------------------------------------------------------------
00850 static uint8_t AVR_Opcode_Cycles_CLH()
00851 {
00852     return 1;
00853 }
00854
00855 //-------------------------------------------------------------------------
00856 static uint8_t AVR_Opcode_Cycles_BREAK()
00857 {
00858     return 1;
00859 }
00860
00861 //-------------------------------------------------------------------------
00862 static uint8_t AVR_Opcode_Cycles_NOP()
00863 {
00864     return 1;
00865 }
00866
00867 //-------------------------------------------------------------------------
00868 static uint8_t AVR_Opcode_Cycles_SLEEP()
00869 {
00870     return 1;
00871 }
00872
00873 //-------------------------------------------------------------------------
```

```
00874 static uint8_t AVR_Opcode_Cycles_WDR()
00875 {
00876     return 1;
00877 }
00878
00879 //-----------------------------------------------------------------------------
00880 static uint8_t AVR_Opcode_Cycles_XCH()
00881 {
00882     return 1;
00883 }
00884
00885 //-----------------------------------------------------------------------------
00886 static uint8_t AVR_Opcode_Cycles_Unimplemented()
00887 {
00888     return 1;
00889 }
00890
00891 //-----------------------------------------------------------------------------
00892 uint8_t AVR_Opcode_Cycles( uint16_t OP_ )
00893 {
00894     // Special instructions - "static" encoding
00895     switch (OP_)
00896     {
00897     case 0x0000: return AVR_Opcode_Cycles_NOP();
00898
00899     case 0x9408: return AVR_Opcode_Cycles_SEC();
00900     case 0x9409: return AVR_Opcode_Cycles_IJMP();
00901     case 0x9418: return AVR_Opcode_Cycles_SEZ();
00902     case 0x9419: return AVR_Opcode_Cycles_EIJMP();
00903     case 0x9428: return AVR_Opcode_Cycles_SEN();
00904     case 0x9438: return AVR_Opcode_Cycles_SEV();
00905     case 0x9448: return AVR_Opcode_Cycles_SES();
00906     case 0x9458: return AVR_Opcode_Cycles_SEH();
00907     case 0x9468: return AVR_Opcode_Cycles_SET();
00908     case 0x9478: return AVR_Opcode_Cycles_SEI();
00909
00910     case 0x9488: return AVR_Opcode_Cycles_CLC();
00911     case 0x9498: return AVR_Opcode_Cycles_CLZ();
00912     case 0x94A8: return AVR_Opcode_Cycles_CLN();
00913     case 0x94B8: return AVR_Opcode_Cycles_CLV();
00914     case 0x94C8: return AVR_Opcode_Cycles_CLS();
00915     case 0x94D8: return AVR_Opcode_Cycles_CLH();
00916     case 0x94E8: return AVR_Opcode_Cycles_CLT();
00917     case 0x94F8: return AVR_Opcode_Cycles_CLI();
00918
00919     case 0x9508: return AVR_Opcode_Cycles_RET();
00920     case 0x9509: return AVR_Opcode_Cycles_ICALL();
00921     case 0x9518: return AVR_Opcode_Cycles_RETI();
00922     case 0x9519: return AVR_Opcode_Cycles_EICALL();
00923     case 0x9588: return AVR_Opcode_Cycles_SLEEP();
00924     case 0x9598: return AVR_Opcode_Cycles_BREAK();
00925     case 0x95A8: return AVR_Opcode_Cycles_WDR();
00926     case 0x95C8: return AVR_Opcode_Cycles_LPM();
00927     case 0x95D8: return AVR_Opcode_Cycles_ELPM();
00928     case 0x95E8: return AVR_Opcode_Cycles_SPM();
00929     case 0x95F8: return AVR_Opcode_Cycles_SPM_Z_Postinc2();
00930     }
00931
00932 #if 0
00933     // Note: These disasm handlers are generalized versions of specific mnemonics in the above list.
00934     // For disassembly, it's probably easier to read the output from the more "spcific" mnemonics, so
00935     // those are used.  For emulation, using the generalized functions may be more desirable.
00936     switch( OP_ & 0xFF8F)
00937     {
00938     case 0x9408: return AVR_Opcode_Cycles_BSET();
00939     case 0x9488: return AVR_Opcode_Cycles_BCLR();
00940     }
00941 #endif
00942
00943     switch (OP_ & 0xFF88)
00944     {
00945     case 0x0300: return AVR_Opcode_Cycles_MULSU();
00946     case 0x0308: return AVR_Opcode_Cycles_FMUL();
00947     case 0x0380: return AVR_Opcode_Cycles_FMULS();
00948     case 0x0388: return AVR_Opcode_Cycles_FMULSU();
00949     }
00950
00951     switch (OP_ & 0xFF0F)
00952     {
00953     case 0x940B: return AVR_Opcode_Cycles_DES();
00954     case 0xEF0F: return AVR_Opcode_Cycles_SER();
00955     }
00956
00957     switch (OP_ & 0xFF00)
00958     {
00959     case 0x0100: return AVR_Opcode_Cycles_MOVW();
00960     case 0x9600: return AVR_Opcode_Cycles_ADIW();
```

```
00961     case 0x9700: return AVR_Opcode_Cycles_SBIW();
00962
00963     case 0x9800: return AVR_Opcode_Cycles_CBI();
00964     case 0x9900: return AVR_Opcode_Cycles_SBIC();
00965     case 0x9A00: return AVR_Opcode_Cycles_SBI();
00966     case 0x9B00: return AVR_Opcode_Cycles_SBIS();
00967     }
00968
00969     switch (OP_ & 0xFE0F)
00970     {
00971     case 0x8008: return AVR_Opcode_Cycles_LD_Y_Indirect();
00972     case 0x8000: return AVR_Opcode_Cycles_LD_Z_Indirect();
00973     case 0x8200: return AVR_Opcode_Cycles_ST_Z_Indirect();
00974     case 0x8208: return AVR_Opcode_Cycles_ST_Y_Indirect();
00975
00976     // -- Single 5-bit register...
00977     case 0x9000: return AVR_Opcode_Cycles_LDS();
00978     case 0x9001: return AVR_Opcode_Cycles_LD_Z_Indirect_Postinc();
00979     case 0x9002: return AVR_Opcode_Cycles_LD_Z_Indirect_Predec();
00980     case 0x9004: return AVR_Opcode_Cycles_LPM_Z();
00981     case 0x9005: return AVR_Opcode_Cycles_LPM_Z_Postinc();
00982     case 0x9006: return AVR_Opcode_Cycles_ELPM_Z();
00983     case 0x9007: return AVR_Opcode_Cycles_ELPM_Z_Postinc();
00984     case 0x9009: return AVR_Opcode_Cycles_LD_Y_Indirect_Postinc();
00985     case 0x900A: return AVR_Opcode_Cycles_LD_Y_Indirect_Predec();
00986     case 0x900C: return AVR_Opcode_Cycles_LD_X_Indirect();
00987     case 0x900D: return AVR_Opcode_Cycles_LD_X_Indirect_Postinc();
00988     case 0x900E: return AVR_Opcode_Cycles_LD_X_Indirect_Predec();
00989     case 0x900F: return AVR_Opcode_Cycles_POP();
00990
00991     case 0x9200: return AVR_Opcode_Cycles_STS();
00992     case 0x9201: return AVR_Opcode_Cycles_ST_Z_Indirect_Postinc();
00993     case 0x9202: return AVR_Opcode_Cycles_ST_Z_Indirect_Predec();
00994     case 0x9204: return AVR_Opcode_Cycles_XCH();
00995     case 0x9205: return AVR_Opcode_Cycles_LAS();
00996     case 0x9206: return AVR_Opcode_Cycles_LAC();
00997     case 0x9207: return AVR_Opcode_Cycles_LAT();
00998     case 0x9209: return AVR_Opcode_Cycles_ST_Y_Indirect_Postinc();
00999     case 0x920A: return AVR_Opcode_Cycles_ST_Y_Indirect_Predec();
01000     case 0x920C: return AVR_Opcode_Cycles_ST_X_Indirect();
01001     case 0x920D: return AVR_Opcode_Cycles_ST_X_Indirect_Postinc();
01002     case 0x920E: return AVR_Opcode_Cycles_ST_X_Indirect_Predec();
01003     case 0x920F: return AVR_Opcode_Cycles_PUSH();
01004
01005     // -- One-operand instructions
01006     case 0x9400: return AVR_Opcode_Cycles_COM();
01007     case 0x9401: return AVR_Opcode_Cycles_NEG();
01008     case 0x9402: return AVR_Opcode_Cycles_SWAP();
01009     case 0x9403: return AVR_Opcode_Cycles_INC();
01010     case 0x9405: return AVR_Opcode_Cycles_ASR();
01011     case 0x9406: return AVR_Opcode_Cycles_LSR();
01012     case 0x9407: return AVR_Opcode_Cycles_ROR();
01013     case 0x940A: return AVR_Opcode_Cycles_DEC();
01014
01015     }
01016     switch (OP_ & 0xFE0E)
01017     {
01018     case 0x940C: return AVR_Opcode_Cycles_JMP();
01019     case 0x940E: return AVR_Opcode_Cycles_CALL();
01020     }
01021
01022     switch (OP_ & 0xFE08)
01023     {
01024
01025     // -- BLD/BST Encoding
01026     case 0xF800: return AVR_Opcode_Cycles_BLD();
01027     case 0xFA00: return AVR_Opcode_Cycles_BST();
01028     // -- SBRC/SBRS Encoding
01029     case 0xFC00: return AVR_Opcode_Cycles_SBRC();
01030     case 0xFE00: return AVR_Opcode_Cycles_SBRS();
01031     }
01032
01033     switch (OP_ & 0xFC07)
01034     {
01035     // -- Conditional branches
01036     case 0xF000: return AVR_Opcode_Cycles_BRCS();
01037     // case 0xF000: return AVR_Opcode_Cycles_BRLO();            // AKA AVR_Opcode_Cycles_BRCS();
01038     case 0xF001: return AVR_Opcode_Cycles_BREQ();
01039     case 0xF002: return AVR_Opcode_Cycles_BRMI();
01040     case 0xF003: return AVR_Opcode_Cycles_BRVS();
01041     case 0xF004: return AVR_Opcode_Cycles_BRLT();
01042     case 0xF006: return AVR_Opcode_Cycles_BRTS();
01043     case 0xF007: return AVR_Opcode_Cycles_BRIE();
01044     case 0xF400: return AVR_Opcode_Cycles_BRCC();
01045     // case 0xF400: return AVR_Opcode_Cycles_BRSH();            // AKA AVR_Opcode_Cycles_BRCC();
01046     case 0xF401: return AVR_Opcode_Cycles_BRNE();
01047     case 0xF402: return AVR_Opcode_Cycles_BRPL();
```

```
01048     case 0xF403: return AVR_Opcode_Cycles_BRVC();
01049     case 0xF404: return AVR_Opcode_Cycles_BRGE();
01050     case 0xF405: return AVR_Opcode_Cycles_BRHC();
01051     case 0xF406: return AVR_Opcode_Cycles_BRTC();
01052     case 0xF407: return AVR_Opcode_Cycles_BRID();
01053     }
01054
01055     switch (OP_ & 0xFC00)
01056     {
01057     // -- 4-bit register pair
01058     case 0x0200: return AVR_Opcode_Cycles_MULS();
01059
01060     // -- 5-bit register pairs --
01061     case 0x0400: return AVR_Opcode_Cycles_CPC();
01062     case 0x0800: return AVR_Opcode_Cycles_SBC();
01063     case 0x0C00: return AVR_Opcode_Cycles_ADD();
01064     // case 0x0C00: return AVR_Opcode_Cycles_LSL(); (!! Implemented with: " add rd, rd"
01065     case 0x1000: return AVR_Opcode_Cycles_CPSE();
01066     case 0x1300: return AVR_Opcode_Cycles_ROL();
01067     case 0x1400: return AVR_Opcode_Cycles_CP();
01068     case 0x1C00: return AVR_Opcode_Cycles_ADC();
01069     case 0x1800: return AVR_Opcode_Cycles_SUB();
01070     case 0x2000: return AVR_Opcode_Cycles_AND();
01071     // case 0x2000: return AVR_Opcode_Cycles_TST(); (!! Implemented with: " and rd, rd"
01072     case 0x2400: return AVR_Opcode_Cycles_EOR();
01073     case 0x2C00: return AVR_Opcode_Cycles_MOV();
01074     case 0x2800: return AVR_Opcode_Cycles_OR();
01075
01076     // -- 5-bit register pairs -- Destination = R1:R0
01077     case 0x9C00: return AVR_Opcode_Cycles_MUL();
01078     }
01079
01080     switch (OP_ & 0xF800)
01081     {
01082     case  0xB800: return AVR_Opcode_Cycles_OUT();
01083     case  0xB000: return AVR_Opcode_Cycles_IN();
01084     }
01085
01086     switch (OP_ & 0xF000)
01087     {
01088     // -- Register immediate --
01089     case 0x3000: return AVR_Opcode_Cycles_CPI();
01090     case 0x4000: return AVR_Opcode_Cycles_SBCI();
01091     case 0x5000: return AVR_Opcode_Cycles_SUBI();
01092     case 0x6000: return AVR_Opcode_Cycles_ORI();
01093     case 0x7000: return AVR_Opcode_Cycles_ANDI();
01094
01095     //-- 12-bit immediate
01096     case 0xC000: return AVR_Opcode_Cycles_RJMP();
01097     case 0xD000: return AVR_Opcode_Cycles_RCALL();
01098
01099     // -- Register immediate
01100     case 0xE000: return AVR_Opcode_Cycles_LDI();
01101     }
01102
01103     switch (OP_ & 0xD208)
01104     {
01105     // -- 7-bit signed offset
01106     case 0x8000: return AVR_Opcode_Cycles_LDD_Z();
01107     case 0x8008: return AVR_Opcode_Cycles_LDD_Y();
01108     case 0x8200: return AVR_Opcode_Cycles_STD_Z();
01109     case 0x8208: return AVR_Opcode_Cycles_STD_Y();
01110     }
01111
01112     return AVR_Opcode_Cycles_Unimplemented();
01113 }
01114
```

4.29 avr_op_cycles.h File Reference

Opcode cycle counting functions.

```
#include <stdint.h>
```

Functions

- uint8_t AVR_Opcode_Cycles (uint16_t OP_)

AVR_Opocde_Cycles.

4.29.1 Detailed Description

Opcode cycle counting functions.

Definition in file avr_op_cycles.h.

4.29.2 Function Documentation

4.29.2.1 uint8_t AVR_Opcode_Cycles (uint16_t *OP_*)

AVR_Opocde_Cycles.

Parameters

*OP_*	Opcode to compute the minimum cycles to execute for

Returns

The minimum number of cycles it will take to execute an opcode

Definition at line 892 of file avr_op_cycles.c.

4.30 avr_op_cycles.h

```
00001 /*****************************************************************************
00002  *      (     (                       (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\     (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ----------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] ------------------
00008  *   | __| | |__    / _ \   \ V /  |   /   | -- [ Runtime ] ------------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\   |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #ifndef __AVR_OP_CYCLES_H__
00022 #define __AVR_OP_CYCLES_H__
00023
00024 #include <stdint.h>
00025
00026 //-----------------------------------------------------------------------------
00032 uint8_t AVR_Opcode_Cycles( uint16_t OP_ );
00033
00034 #endif
```

4.31 avr_op_decode.c File Reference

Module providing logic to decode AVR CPU Opcodes.

```
#include <stdint.h>
#include "emu_config.h"
#include "avr_op_decode.h"
```

Functions

- static void **AVR_Decoder_NOP** (uint16_t OP_)

- static void **AVR_Decoder_Register_Pair_4bit** (uint16_t OP_)
- static void **AVR_Decoder_Register_Pair_3bit** (uint16_t OP_)
- static void **AVR_Decoder_Register_Pair_5bit** (uint16_t OP_)
- static void **AVR_Decoder_Register_Immediate** (uint16_t OP_)
- static void **AVR_Decoder_LDST_YZ_k** (uint16_t OP_)
- static void **AVR_Decoder_LDST** (uint16_t OP_)
- static void **AVR_Decoder_LDS_STS** (uint16_t OP_)
- static void **AVR_Decoder_Register_Single** (uint16_t OP_)
- static void **AVR_Decoder_Register_SC** (uint16_t OP_)
- static void **AVR_Decoder_Misc** (uint16_t OP_)
- static void **AVR_Decoder_Indirect_Jump** (uint16_t OP_)
- static void **AVR_Decoder_DEC_Rd** (uint16_t OP_)
- static void **AVR_Decoder_DES_round_4** (uint16_t OP_)
- static void **AVR_Decoder_JMP_CALL_22** (uint16_t OP_)
- static void **AVR_Decoder_ADIW_SBIW_6** (uint16_t OP_)
- static void **AVR_Decoder_IO_Bit** (uint16_t OP_)
- static void **AVR_Decoder_MUL** (uint16_t OP_)
- static void **AVR_Decoder_IO_In_Out** (uint16_t OP_)
- static void **AVR_Decoder_Relative_Jump** (uint16_t OP_)
- static void **AVR_Decoder_LDI** (uint16_t OP_)
- static void **AVR_Decoder_Conditional_Branch** (uint16_t OP_)
- static void **AVR_Decoder_BLD_BST** (uint16_t OP_)
- static void **AVR_Decoder_SBRC_SBRS** (uint16_t OP_)
- AVR_Decoder AVR_Decoder_Function (uint16_t OP_)

  AVR_Decoder_Function.
- void AVR_Decode (uint16_t OP_)

  AVR_Decode.

4.31.1 Detailed Description

Module providing logic to decode AVR CPU Opcodes. Implemented based on descriptions provided in Atmel document doc0856

Definition in file avr_op_decode.c.

4.31.2 Function Documentation

4.31.2.1 void AVR_Decode (uint16_t *OP_*)

AVR_Decode.

Decode a specified instruction into the internal registers of the CPU object. Opcodes must be decoded before they can be executed.

Parameters

*OP_*	Opcode to decode

Definition at line 400 of file avr_op_decode.c.

4.31.2.2 AVR_Decoder AVR_Decoder_Function (uint16_t *OP_*)

AVR_Decoder_Function.

Returns an "instruction decode" function pointer to the caller for a given opcode.

Parameters

*OP_*	Opcode to return the instruction decode function for

Returns

Pointer to an opcode/instruction decoder routine

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verfied

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

!MOS Verified

Definition at line 251 of file avr_op_decode.c.

4.32 avr_op_decode.c

```
00001 /*************************************************************************
00002  *      (      (                      (      |
00003  *    )\ )   )\ )     (              )\ )    |
00004  *   (()/(  (()/(     )\      (    (  (()/(    | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\   )\   /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ----------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \     | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /     | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\     |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *************************************************************************/
00024 #include <stdint.h>
00025
00026 #include "emu_config.h"
```

```
00027
00028 #include "avr_op_decode.h"
00029
00030 //------------------------------------------------------------------------
00031 static void AVR_Decoder_NOP( uint16_t OP_)
00032 {
00033     // Nothing to do here...
00034 }
00035 //------------------------------------------------------------------------
00036 static void AVR_Decoder_Register_Pair_4bit( uint16_t OP_)
00037 {
00038     uint8_t Rr = (OP_ & 0x000F);
00039     uint8_t Rd = ((OP_ & 0x00F0) >> 4);
00040
00041     stCPU.Rr16 = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r_word[Rr]);
00042     stCPU.Rd16 = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r_word[Rd]);
00043 }
00044 //------------------------------------------------------------------------
00045 static void AVR_Decoder_Register_Pair_3bit( uint16_t OP_)
00046 {
00047     uint8_t Rr = (OP_ & 0x0007) + 16;
00048     uint8_t Rd = ((OP_ & 0x0070) >> 4) + 16;
00049
00050     stCPU.Rr = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rr]);
00051     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00052 }
00053 //------------------------------------------------------------------------
00054 static void AVR_Decoder_Register_Pair_5bit( uint16_t OP_)
00055 {
00056     uint8_t Rr = (OP_ & 0x000F) | ((OP_ & 0x0200) >> 5);
00057     uint8_t Rd = (OP_ & 0x01F0) >> 4;
00058
00059     stCPU.Rr = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rr]);
00060     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00061 }
00062 //------------------------------------------------------------------------
00063 static void AVR_Decoder_Register_Immediate( uint16_t OP_)
00064 {
00065     uint8_t K = (OP_ & 0x000F) | ((OP_ & 0x0F00) >> 4);
00066     uint8_t Rd = ((OP_ & 0x00F0) >> 4) + 16;
00067
00068     stCPU.K = K;
00069     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00070 }
00071 //------------------------------------------------------------------------
00072 static void AVR_Decoder_LDST_YZ_k( uint16_t OP_)
00073 {
00074     uint8_t q = (OP_ & 0x0007) |            // Awkward encoding... see manual for details.
00075                 ((OP_ & 0x0C00) >> (7)) |
00076                 ((OP_ & 0x2000) >> (8));
00077
00078     uint8_t Rd = (OP_ & 0x01F0) >> 4;
00079
00080     stCPU.q = q;
00081     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00082 }
00083 //------------------------------------------------------------------------
00084 static void AVR_Decoder_LDST( uint16_t OP_)
00085 {
00086     uint8_t Rd = (OP_ & 0x01F0) >> 4;
00087
00088     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00089 }
00090 //------------------------------------------------------------------------
00091 static void AVR_Decoder_LDS_STS( uint16_t OP_)
00092 {
00093     uint8_t Rd = (OP_ & 0x01F0) >> 4;
00094
00095     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00096     stCPU.K = stCPU.pu16ROM[ stCPU.u16PC + 1 ];
00097 }
00098 //------------------------------------------------------------------------
00099 static void AVR_Decoder_Register_Single( uint16_t OP_)
00100 {
00101     uint8_t Rd = (OP_ & 0x01F0) >> 4;
00102
00103     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00104 }
00105 //------------------------------------------------------------------------
00106 static void AVR_Decoder_Register_SC( uint16_t OP_)
00107 {
00108     uint8_t b = (OP_ & 0x0070) >> 4;
00109
00110     stCPU.b = b;
00111 }
00112 //------------------------------------------------------------------------
00113 static void AVR_Decoder_Misc( uint16_t OP_)
```

```
00114 {
00115     // Nothing to do here.
00116 }
00117 //-----------------------------------------------------------------------------
00118 static void AVR_Decoder_Indirect_Jump( uint16_t OP_)
00119 {
00120     // Nothing to do here.
00121 }
00122 //-----------------------------------------------------------------------------
00123 static void AVR_Decoder_DEC_Rd( uint16_t OP_)
00124 {
00125     uint8_t Rd = (OP_ & 0x01F0) >> 4;
00126 
00127     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00128 }
00129 //-----------------------------------------------------------------------------
00130 static void AVR_Decoder_DES_round_4( uint16_t OP_)
00131 {
00132     uint8_t K = (OP_ & 0x00F0) >> 4;
00133     stCPU.K = K;
00134 }
00135 //-----------------------------------------------------------------------------
00136 static void AVR_Decoder_JMP_CALL_22( uint16_t OP_)
00137 {
00138     uint16_t op = stCPU.pu16ROM[ stCPU.u16PC + 1 ];
00139     uint32_t k = op;
00140     k |= (((OP_ & 0x0001) | (OP_ & 0x01F0) >> 3) << 16);
00141 
00142     stCPU.k = k;
00143 
00144     // These are 2-cycle instructions.  Clock the CPU here, since we're fetching
00145     // the second word of data for this opcode here.
00146     IO_Clock();
00147 }
00148 //-----------------------------------------------------------------------------
00149 static void AVR_Decoder_ADIW_SBIW_6( uint16_t OP_)
00150 {
00151     uint8_t K = (OP_ & 0x000F) | ((OP_ & 0x00C0) >> 2);
00152     uint8_t Rd16 = (((OP_ & 0x0030) >> 4) * 2) + 24;
00153 
00154     stCPU.K = K;
00155     stCPU.Rd16 = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r_word[Rd16 >> 1]);
00156 }
00157 //-----------------------------------------------------------------------------
00158 static void AVR_Decoder_IO_Bit( uint16_t OP_)
00159 {
00160     uint8_t b = (OP_ & 0x0007);
00161     uint8_t A = (OP_ & 0x00F8) >> 3;
00162 
00163     stCPU.b = b;
00164     stCPU.A = A;
00165 }
00166 //-----------------------------------------------------------------------------
00167 static void AVR_Decoder_MUL( uint16_t OP_)
00168 {
00169     uint8_t Rr = (OP_ & 0x000F) | ((OP_ & 0x0200) >> 5);
00170     uint8_t Rd = (OP_ & 0x01F0) >> 4;
00171 
00172     stCPU.Rr = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rr]);
00173     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00174 }
00175 //-----------------------------------------------------------------------------
00176 static void AVR_Decoder_IO_In_Out( uint16_t OP_)
00177 {
00178     uint8_t A = (OP_ & 0x000F) | ((OP_ & 0x0600) >> 5);
00179     uint8_t Rd = (OP_ & 0x01F0) >> 4;
00180 
00181     stCPU.A = A;
00182     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00183 }
00184 //-----------------------------------------------------------------------------
00185 static void AVR_Decoder_Relative_Jump( uint16_t OP_)
00186 {
00187     // NB: -2K <= k <= 2K
00188     uint16_t k = (OP_ & 0x0FFF);
00189 
00190     // Check for sign bit in 12-bit value...
00191     if (k & 0x0800)
00192     {
00193         stCPU.k_s = (int32_t)((~k & 0x07FF) + 1) * -1;
00194     }
00195     else
00196     {
00197         stCPU.k_s = (int32_t)k;
00198     }
00199 }
00200 //-----------------------------------------------------------------------------
```

```
00201 static void AVR_Decoder_LDI( uint16_t OP_)
00202 {
00203     uint8_t K = (OP_ & 0x000F) | ((OP_ & 0x0F00) >> 4);
00204     uint8_t Rd = ((OP_ & 0x00F0) >> 4) + 16;
00205 
00206     stCPU.K = K;
00207     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00208 }
00209 //-----------------------------------------------------------------------------
00210 static void AVR_Decoder_Conditional_Branch( uint16_t OP_)
00211 {
00212     // NB: -64 <= k <= 63
00213     uint8_t b = (OP_ & 0x0007);
00214     uint8_t k = ((OP_ & 0x03F8) >> 3);
00215 
00216     stCPU.b = b;
00217 
00218     // Check for sign bit in 7-bit value...
00219     if (k & 0x40)
00220     {
00221         // Convert to signed 32-bit integer... probably a cleaner way
00222         // of doing this, but I'm tired.
00223         stCPU.k_s = (int32_t)((~k & 0x3F) + 1) * -1;
00224     }
00225     else
00226     {
00227         stCPU.k_s = (int32_t)k;
00228     }
00229 }
00230 //-----------------------------------------------------------------------------
00231 static void AVR_Decoder_BLD_BST( uint16_t OP_)
00232 {
00233     uint8_t b = (OP_ & 0x0007);
00234     uint8_t Rd = ((OP_ & 0x01F0) >> 4);
00235 
00236     stCPU.b = b;
00237     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00238 }
00239 
00240 //-----------------------------------------------------------------------------
00241 static void AVR_Decoder_SBRC_SBRS( uint16_t OP_)
00242 {
00243     uint8_t b = (OP_ & 0x0007);
00244     uint8_t Rd = ((OP_ & 0x01F0) >> 4);
00245 
00246     stCPU.b = b;
00247     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00248 }
00249 
00250 //-----------------------------------------------------------------------------
00251 AVR_Decoder AVR_Decoder_Function( uint16_t OP_ )
00252 {
00253     if (( OP_ & 0xFF0F) == 0x9408 )
00254     {
00256         // SEx/CLx status register clear/set bit.
00257         return AVR_Decoder_Register_SC;
00258     }
00259     else if (( OP_ & 0xFF0F) == 0x9508 )
00260     {
00262         // Miscellaneous instruction
00263         return AVR_Decoder_Misc;
00264     }
00265     else if (( OP_ & 0xFF0F) == 0x940B )
00266     {
00268         // Des round k
00269         return AVR_Decoder_DES_round_4;
00270     }
00271     else if ( (( OP_ & 0xFF00 ) == 0x0100 ) ||
00272               (( OP_ & 0xFF00 ) == 0x0200 ) )
00273     {
00275         // Register pair 4bit (MOVW, MULS)
00276         return  AVR_Decoder_Register_Pair_4bit;
00277     }
00278     else if (( OP_ & 0xFF00 ) == 0x0300 )
00279     {
00281         // 3-bit register pair (R16->R23) - (FMUL, FMULS, FMULSU, MULSU)
00282         return AVR_Decoder_Register_Pair_3bit;
00283     }
00284     else if (( OP_ & 0xFF00 ) <= 0x2F00 )
00285     {
00286         // Register pair 5bit
00287         return AVR_Decoder_Register_Pair_5bit;
00288     }
00289     else if (( OP_ & 0xFF00) <= 0x7F00 )
00290     {
00292         // Register immediate
00293         return AVR_Decoder_Register_Immediate;
```

```
00294     }
00295     else if (( OP_ & 0xFEEF) == 0x9409 )
00296     {
00298         // Indirect Jump/call
00299         return AVR_Decoder_Indirect_Jump;
00300     }
00301     else if (( OP_ & 0xFE08) == 0x9400 )
00302     {
00304         // 1-operand instructions.
00305         return AVR_Decoder_Register_Single;
00306     }
00307     else if (( OP_ & 0xFE0F) == 0x940A )
00308     {
00310         // Dec Rd
00311         return AVR_Decoder_DEC_Rd;
00312     }
00313     else if (( OP_ & 0xFE0C) == 0x940C )
00314     {
00316         // Jmp/call abs22
00317         return AVR_Decoder_JMP_CALL_22;
00318     }
00319     else if (( OP_ & 0xFE00) == 0x9600 )
00320     {
00322         // ADIW/SBIW Rp
00323         return AVR_Decoder_ADIW_SBIW_6;
00324     }
00325     else if (( OP_ & 0xFC0F) == 0x9000 )
00326     {
00328         // LDS/STS
00329         return AVR_Decoder_LDS_STS;
00330     }
00331     else if (( OP_ & 0xFC00) == 0x9000 )
00332     {
00334         // LD/ST other
00335         return AVR_Decoder_LDST;
00336     }
00337     else if (( OP_ & 0xFC00) == 0x9800 )
00338     {
00340         // IO Space bit operations
00341         return AVR_Decoder_IO_Bit;
00342     }
00343     else if (( OP_ & 0xFC00) == 0x9C00 )
00344     {
00346         // MUL unsigned R1:R0 = Rr x Rd
00347         return AVR_Decoder_MUL;
00348     }
00349     else if (( OP_ & 0xFC00) == 0xF800 )
00350     {
00352         // BLD/BST register bit to STATUS.T
00353         return AVR_Decoder_BLD_BST;
00354     }
00355     else if (( OP_ & 0xFC00) == 0xFC00 )
00356     {
00358         // SBRC/SBRS
00359         return AVR_Decoder_SBRC_SBRS;
00360     }
00361     else if (( OP_ & 0xF800) == 0xF000 )
00362     {
00364         // Conditional branch
00365         return AVR_Decoder_Conditional_Branch;
00366     }
00367     else if (( OP_ & 0xF000) == 0xE000 )
00368     {
00370         // LDI Rh, K
00371         return AVR_Decoder_LDI;
00372     }
00373     else if (( OP_ & 0xF000) == 0xB000 )
00374     {
00376         // IO space IN/OUT operations
00377         return AVR_Decoder_IO_In_Out;
00378     }
00379     else if (( OP_ & 0xE000) == 0xC000 )
00380     {
00382         // RElative Jump/Call
00383         return AVR_Decoder_Relative_Jump;
00384     }
00385     else if (( OP_ & 0xD000) == 0x8000 )
00386     {
00388         // LDD/STD to Z+kY+k
00389         return AVR_Decoder_LDST_YZ_k;
00390     }
00391     else if ( OP_ == 0 )
00392     {
00394         return AVR_Decoder_NOP;
00395     }
00396     return AVR_Decoder_NOP;
00397 }
```

```
00398
00399 //------------------------------------------------------------------------------
00400 void AVR_Decode( uint16_t OP_ )
00401 {
00402     AVR_Decoder myDecoder;
00403     myDecoder = AVR_Decoder_Function(OP_);
00404     myDecoder( OP_);
00405 }
```

4.33 avr_op_decode.h File Reference

Module providing logic to decode AVR CPU Opcodes.

```
#include <stdint.h>
#include "avr_cpu.h"
```

Typedefs

- typedef void(∗ **AVR_Decoder**)(uint16_t OP_)

Functions

- AVR_Decoder AVR_Decoder_Function (uint16_t OP_)

  AVR_Decoder_Function.
- void AVR_Decode (uint16_t OP_)

  AVR_Decode.

4.33.1 Detailed Description

Module providing logic to decode AVR CPU Opcodes.

Definition in file avr_op_decode.h.

4.33.2 Function Documentation

4.33.2.1 void AVR_Decode (uint16_t *OP_*)

AVR_Decode.

Decode a specified instruction into the internal registers of the CPU object. Opcodes must be decoded before they can be executed.

Parameters

*OP_*	Opcode to decode

Definition at line 400 of file avr_op_decode.c.

4.33.2.2 AVR_Decoder AVR_Decoder_Function (uint16_t *OP_*)

AVR_Decoder_Function.

Returns an "instruction decode" function pointer to the caller for a given opcode.

Parameters

*OP_*	Opcode to return the instruction decode function for

Returns

Pointer to an opcode/instruction decoder routine

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verfied

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

! MOS Verified

!MOS Verified

Definition at line 251 of file avr_op_decode.c.

4.34 avr_op_decode.h

```
00001 /*****************************************************************************
00002  *      (      (                       (      |
00003  *    )\ )   )\ )     (              )\ )    |
00004  *   (()/( (()/(     )\      (    (  (()/(    | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\   )\   /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \     | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /     | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\     |
00010  *                                         | "Yeah, it does Arduino..."
00011  * ----------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #ifndef __AVR_OP_DECODE_H__
00022 #define __AVR_OP_DECODE_H__
00023
```

```
00024 #include <stdint.h>
00025 #include "avr_cpu.h"
00026
00027 //------------------------------------------------------------------------------
00028 // Format decoder function jump table
00029 typedef void (*AVR_Decoder)( uint16_t OP_);
00030
00031 //------------------------------------------------------------------------------
00041 AVR_Decoder AVR_Decoder_Function( uint16_t OP_ );
00042
00043 //------------------------------------------------------------------------------
00052 void AVR_Decode( uint16_t OP_ );
00053
00054 #endif
00055
```

4.35 avr_op_size.c File Reference

Module providing opcode sizes.

```
#include <stdint.h>
#include "emu_config.h"
#include "avr_op_size.h"
```

Functions

- static uint8_t **AVR_Opcode_Size_NOP** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_Register_Pair_4bit** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_Register_Pair_3bit** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_Register_Pair_5bit** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_Register_Immediate** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_LDST_YZ_k** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_LDST** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_LDS_STS** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_Register_Single** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_Register_SC** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_Misc** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_Indirect_Jump** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_DEC_Rd** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_DES_round_4** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_JMP_CALL_22** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_ADIW_SBIW_6** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_IO_Bit** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_MUL** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_IO_In_Out** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_Relative_Jump** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_LDI** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_Conditional_Branch** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_BLD_BST** (uint16_t OP_)
- static uint8_t **AVR_Opcode_Size_SBRC_SBRS** (uint16_t OP_)
- uint8_t AVR_Opcode_Size (uint16_t OP_)

  AVR_Opocde_Size.

4.35.1 Detailed Description

Module providing opcode sizes.

Definition in file avr_op_size.c.

4.35.2 Function Documentation

4.35.2.1 uint8_t AVR_Opcode_Size (uint16_t *OP_*)

AVR_Opocde_Size.

Return the number of bytes are in a specific opcode based on a 16-bt first opcode word.

Parameters

*OP_*	Opcode word to determine instruction size for

Returns

The number of words in an instruction

Definition at line 150 of file avr_op_size.c.

4.36 avr_op_size.c

```
00001 /*************************************************************************
00002  *      (     (                       (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\      (   (  (()/(   | -- [ Funkenstein ] ------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] -------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] --------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+-------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *************************************************************************/
00021 #include <stdint.h>
00022
00023 #include "emu_config.h"
00024
00025 #include "avr_op_size.h"
00026
00027 //-------------------------------------------------------------------------
00028 static uint8_t AVR_Opcode_Size_NOP( uint16_t OP_)
00029 {
00030     return 1;
00031 }
00032 //-------------------------------------------------------------------------
00033 static uint8_t AVR_Opcode_Size_Register_Pair_4bit( uint16_t OP_)
00034 {
00035     return 1;
00036 }
00037 //-------------------------------------------------------------------------
00038 static uint8_t AVR_Opcode_Size_Register_Pair_3bit( uint16_t OP_)
00039 {
00040     return 1;
00041 }
00042 //-------------------------------------------------------------------------
00043 static uint8_t AVR_Opcode_Size_Register_Pair_5bit( uint16_t OP_)
00044 {
00045     return 1;
00046 }
00047 //-------------------------------------------------------------------------
00048 static uint8_t AVR_Opcode_Size_Register_Immediate( uint16_t OP_)
00049 {
00050     return 1;
00051 }
00052 //-------------------------------------------------------------------------
00053 static uint8_t AVR_Opcode_Size_LDST_YZ_k( uint16_t OP_)
00054 {
00055     return 1;
00056 }
00057 //-------------------------------------------------------------------------
00058 static uint8_t AVR_Opcode_Size_LDST( uint16_t OP_)
00059 {
00060     return 1;
00061 }
00062 //-------------------------------------------------------------------------
00063 static uint8_t AVR_Opcode_Size_LDS_STS( uint16_t OP_)
```

```
00064 {
00065     return 2;
00066 }
00067 //------------------------------------------------------------------------
00068 static uint8_t AVR_Opcode_Size_Register_Single( uint16_t OP_)
00069 {
00070     return 1;
00071 }
00072 //------------------------------------------------------------------------
00073 static uint8_t AVR_Opcode_Size_Register_SC( uint16_t OP_)
00074 {
00075     return 1;
00076 }
00077 //------------------------------------------------------------------------
00078 static uint8_t AVR_Opcode_Size_Misc( uint16_t OP_)
00079 {
00080     return 1;
00081 }
00082 //------------------------------------------------------------------------
00083 static uint8_t AVR_Opcode_Size_Indirect_Jump( uint16_t OP_)
00084 {
00085     return 1;
00086 }
00087 //------------------------------------------------------------------------
00088 static uint8_t AVR_Opcode_Size_DEC_Rd( uint16_t OP_)
00089 {
00090     return 1;
00091 }
00092 //------------------------------------------------------------------------
00093 static uint8_t AVR_Opcode_Size_DES_round_4( uint16_t OP_)
00094 {
00095     return 1;
00096 }
00097 //------------------------------------------------------------------------
00098 static uint8_t AVR_Opcode_Size_JMP_CALL_22( uint16_t OP_)
00099 {
00100     return 2;
00101 }
00102 //------------------------------------------------------------------------
00103 static uint8_t AVR_Opcode_Size_ADIW_SBIW_6( uint16_t OP_)
00104 {
00105     return 1;
00106 }
00107 //------------------------------------------------------------------------
00108 static uint8_t AVR_Opcode_Size_IO_Bit( uint16_t OP_)
00109 {
00110     return 1;
00111 }
00112 //------------------------------------------------------------------------
00113 static uint8_t AVR_Opcode_Size_MUL( uint16_t OP_)
00114 {
00115     return 1;
00116 }
00117 //------------------------------------------------------------------------
00118 static uint8_t AVR_Opcode_Size_IO_In_Out( uint16_t OP_)
00119 {
00120     return 1;
00121 }
00122 //------------------------------------------------------------------------
00123 static uint8_t AVR_Opcode_Size_Relative_Jump( uint16_t OP_)
00124 {
00125     return 1;
00126 }
00127 //------------------------------------------------------------------------
00128 static uint8_t AVR_Opcode_Size_LDI( uint16_t OP_)
00129 {
00130     return 1;
00131 }
00132 //------------------------------------------------------------------------
00133 static uint8_t AVR_Opcode_Size_Conditional_Branch( uint16_t OP_)
00134 {
00135     return 1;
00136 }
00137 //------------------------------------------------------------------------
00138 static uint8_t AVR_Opcode_Size_BLD_BST( uint16_t OP_)
00139 {
00140     return 1;
00141 }
00142
00143 //------------------------------------------------------------------------
00144 static uint8_t AVR_Opcode_Size_SBRC_SBRS( uint16_t OP_)
00145 {
00146     return 1;
00147 }
00148
00149 //------------------------------------------------------------------------
00150 uint8_t AVR_Opcode_Size( uint16_t OP_ )
```

```
00151 {
00152     if (( OP_ & 0xFF0F) == 0x9408 )
00153     {
00154         // SEx/CLx status register clear/set bit.
00155         return AVR_Opcode_Size_Register_SC( OP_ );
00156     }
00157     else if (( OP_ & 0xFF0F) == 0x9508 )
00158     {
00159         // Miscellaneous instruction
00160         return AVR_Opcode_Size_Misc( OP_ );
00161     }
00162     else if (( OP_ & 0xFF0F) == 0x940B )
00163     {
00164         // Des round k
00165         return AVR_Opcode_Size_DES_round_4( OP_ );
00166     }
00167     else if ( (( OP_ & 0xFF00 ) == 0x0100 ) ||
00168               (( OP_ & 0xFF00 ) == 0x0200 ) )
00169     {
00170         // Register pair 4bit (MOVW, MULS)
00171          return AVR_Opcode_Size_Register_Pair_4bit( OP_ );
00172     }
00173     else if (( OP_ & 0xFF00 ) == 0x0300 )
00174     {
00175         // 3-bit register pair (R16->R23) - (FMUL, FMULS, FMULSU, MULSU)
00176         return AVR_Opcode_Size_Register_Pair_3bit( OP_ );
00177     }
00178     else if (( OP_ & 0xFF00 ) <= 0x4F00 )
00179     {
00180         // Register pair 5bit
00181         return AVR_Opcode_Size_Register_Pair_5bit( OP_ );
00182     }
00183     else if (( OP_ & 0xFF00) <= 0x7F00 )
00184     {
00185         // Register immediate
00186         return AVR_Opcode_Size_Register_Immediate( OP_ );
00187     }
00188     else if (( OP_ & 0xFEEF) == 0x9409 )
00189     {
00190         // Indirect Jump/call
00191         return AVR_Opcode_Size_Indirect_Jump( OP_ );
00192     }
00193     else if (( OP_ & 0xFE08) == 0x9400 )
00194     {
00195         // 1-operand instructions.
00196         return AVR_Opcode_Size_Register_Single( OP_ );
00197     }
00198     else if (( OP_ & 0xFE0F) == 0x940A )
00199     {
00200         // Dec Rd
00201         return AVR_Opcode_Size_DEC_Rd( OP_ );
00202     }
00203     else if (( OP_ & 0xFE0C) == 0x940C )
00204     {
00205         // Jmp/call abs22
00206         return AVR_Opcode_Size_JMP_CALL_22( OP_ );
00207     }
00208     else if (( OP_ & 0xFE00) == 0x9600 )
00209     {
00210         // ADIW/SBIW Rp
00211         return AVR_Opcode_Size_ADIW_SBIW_6( OP_ );
00212     }
00213     else if (( OP_ & 0xFC0F) == 0x9000 )
00214     {
00215         // LDS/STS
00216         return AVR_Opcode_Size_LDS_STS( OP_ );
00217     }
00218     else if (( OP_ & 0xFC00) == 0x9000 )
00219     {
00220         // LD/ST other
00221         return AVR_Opcode_Size_LDST( OP_ );
00222     }
00223     else if (( OP_ & 0xFC00) == 0x9800 )
00224     {
00225         // IO Space bit operations
00226         return AVR_Opcode_Size_IO_Bit( OP_ );
00227     }
00228     else if (( OP_ & 0xFC00) == 0x9C00 )
00229     {
00230         // MUL unsigned R1:R0 = Rr x Rd
00231         return AVR_Opcode_Size_MUL( OP_ );
00232     }
00233     else if (( OP_ & 0xFC00) == 0xF800 )
00234     {
00235         // BLD/BST register bit to STATUS.T
00236         return AVR_Opcode_Size_BLD_BST( OP_ );
00237     }
```

```
00238     else if (( OP_ & 0xFC00) == 0xFC00 )
00239     {
00240         // SBRC/SBRS
00241         return AVR_Opcode_Size_SBRC_SBRS( OP_ );
00242     }
00243     else if (( OP_ & 0xF800) == 0xF000 )
00244     {
00245         // Conditional branch
00246         return AVR_Opcode_Size_Conditional_Branch( OP_ );
00247     }
00248     else if (( OP_ & 0xF000) == 0xE000 )
00249     {
00250         // LDI Rh, K
00251         return AVR_Opcode_Size_LDI( OP_ );
00252     }
00253     else if (( OP_ & 0xF000) == 0xB000 )
00254     {
00255         // IO space IN/OUT operations
00256         return AVR_Opcode_Size_IO_In_Out( OP_ );
00257     }
00258     else if (( OP_ & 0xE000) == 0xC000 )
00259     {
00260         // RElative Jump/Call
00261         return AVR_Opcode_Size_Relative_Jump( OP_ );
00262     }
00263     else if (( OP_ & 0xD000) == 0x8000 )
00264     {
00265         // LDD/STD to Z+kY+k
00266         return AVR_Opcode_Size_LDST_YZ_k( OP_ );
00267     }
00268     else if ( OP_ == 0 )
00269     {
00270         return AVR_Opcode_Size_NOP( OP_ );
00271     }
00272     return AVR_Opcode_Size_NOP( OP_ );
00273 }
```

4.37 avr_op_size.h File Reference

Module providing an interface to lookup the size of an opcode.

```
#include <stdint.h>
```

Functions

- uint8_t AVR_Opcode_Size (uint16_t OP_)
  AVR_Opocde_Size.

4.37.1 Detailed Description

Module providing an interface to lookup the size of an opcode.

Definition in file avr_op_size.h.

4.37.2 Function Documentation

4.37.2.1 uint8_t AVR_Opcode_Size (uint16_t *OP_*)

AVR_Opocde_Size.

Return the number of bytes are in a specific opcode based on a 16-bt first opcode word.

Parameters

*OP_*	Opcode word to determine instruction size for

Returns

The number of words in an instruction

Definition at line 150 of file avr_op_size.c.

4.38 avr_op_size.h

```
00001 /*****************************************************************************
00002  *      (     (                        (     |
00003  *    )\ )  )\ )     (              )\ )    |
00004  *   (()/( (()/(     )\      (   (  (()/(    | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))       | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                         | "Yeah, it does Arduino..."
00011  * ----------------------------------------+-------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #ifndef __AVR_OP_SIZE__
00022 #define __AVR_OP_SIZE__
00023
00024 #include <stdint.h>
00025
00026 //-----------------------------------------------------------------------------
00037 uint8_t AVR_Opcode_Size( uint16_t OP_ );
00038
00039 #endif
```

4.39 avr_opcodes.c File Reference

AVR CPU - Opcode implementation.

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include "avr_cpu_print.h"
#include "emu_config.h"
#include "avr_opcodes.h"
```

Macros

- #define **DEBUG_PRINT**(...)

Functions

- static void **AVR_Abort** (void)
- static void **Data_Write** (uint16_t u16Addr_, uint8_t u8Val_)
- static uint8_t **Data_Read** (uint16_t u16Addr_)
- static void **AVR_Opcode_NOP** (void)
- void **ADD_Half_Carry** (uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
- void **ADD_Full_Carry** (uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
- void **ADD_Overflow_Flag** (uint8_t Rd_, uint8_t Rr_, uint8_t Result_)

- void **Signed_Flag** (void)
- void **R8_Zero_Flag** (uint8_t R_)
- void **R8_CPC_Zero_Flag** (uint8_t R_)
- void **R8_Negative_Flag** (uint8_t R_)
- static void **AVR_Opcode_ADD** (void)
- static void **AVR_Opcode_ADC** (void)
- void **R16_Negative_Flag** (uint16_t Result_)
- void **R16_Zero_Flag** (uint16_t Result_)
- void **ADIW_Overflow_Flag** (uint16_t Rd_, uint16_t Result_)
- void **ADIW_Carry_Flag** (uint16_t Rd_, uint16_t Result_)
- static void **AVR_Opcode_ADIW** (void)
- void **SUB_Overflow_Flag** (uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
- void **SUB_Half_Carry** (uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
- void **SUB_Full_Carry** (uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
- static void **AVR_Opcode_SUB** (void)
- static void **AVR_Opcode_SUBI** (void)
- static void **AVR_Opcode_SBC** (void)
- static void **AVR_Opcode_SBCI** (void)
- void **SBIW_Overflow_Flag** (uint16_t Rd_, uint16_t Result_)
- void **SBIW_Full_Carry** (uint16_t Rd_, uint16_t Result_)
- static void **AVR_Opcode_SBIW** (void)
- static void **AVR_Opcode_AND** (void)
- static void **AVR_Opcode_ANDI** (void)
- static void **AVR_Opcode_OR** (void)
- static void **AVR_Opcode_ORI** (void)
- static void **AVR_Opcode_EOR** (void)
- static void **AVR_Opcode_COM** (void)
- void **NEG_Overflow_Flag** (uint8_t u8Result_)
- void **NEG_Carry_Flag** (uint8_t u8Result_)
- static void **AVR_Opcode_NEG** (void)
- static void **AVR_Opcode_SBR** (void)
- static void **AVR_Opcode_CBR** (void)
- void **INC_Overflow_Flag** (uint8_t u8Result_)
- static void **AVR_Opcode_INC** (void)
- void **DEC_Overflow_Flag** (uint8_t u8Result_)
- static void **AVR_Opcode_DEC** (void)
- static void **AVR_Opcode_SER** (void)
- void **Mul_Carry_Flag** (uint16_t R_)
- void **Mul_Zero_Flag** (uint16_t R_)
- static void **AVR_Opcode_MUL** (void)
- static void **AVR_Opcode_MULS** (void)
- static void **AVR_Opcode_MULSU** (void)
- static void **AVR_Opcode_FMUL** (void)
- static void **AVR_Opcode_FMULS** (void)
- static void **AVR_Opcode_FMULSU** (void)
- static void AVR_Opcode_DES (void)
- static **Unconditional_Jump** (uint16_t u16Addr_)
- static **Relative_Jump** (uint16_t u16Offset_)
- static void **AVR_Opcode_RJMP** (void)
- static void **AVR_Opcode_IJMP** (void)
- static void AVR_Opcode_EIJMP (void)
- static void **AVR_Opcode_JMP** (void)
- static void **AVR_Opcode_RCALL** (void)
- static void **AVR_Opcode_ICALL** (void)
- static void AVR_Opcode_EICALL (void)

- static void **AVR_Opcode_CALL** (void)
- static void **AVR_Opcode_RET** (void)
- static void **AVR_Opcode_RETI** (void)
- static void **AVR_Opcode_CPSE** (void)
- void **CP_Half_Carry** (uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
- void **CP_Full_Carry** (uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
- void **CP_Overflow_Flag** (uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
- static void **AVR_Opcode_CP** (void)
- static void **AVR_Opcode_CPC** (void)
- static void **AVR_Opcode_CPI** (void)
- static void **AVR_Opcode_SBRC** (void)
- static void **AVR_Opcode_SBRS** (void)
- static void **AVR_Opcode_SBIC** (void)
- static void **AVR_Opcode_SBIS** (void)
- static **Conditional_Branch** (void)
- static void **AVR_Opcode_BRBS** (void)
- static void **AVR_Opcode_BRBC** (void)
- static void **AVR_Opcode_BREQ** (void)
- static void **AVR_Opcode_BRNE** (void)
- static void **AVR_Opcode_BRCS** (void)
- static void **AVR_Opcode_BRCC** (void)
- static void **AVR_Opcode_BRSH** (void)
- static void **AVR_Opcode_BRLO** (void)
- static void **AVR_Opcode_BRMI** (void)
- static void **AVR_Opcode_BRPL** (void)
- static void **AVR_Opcode_BRGE** (void)
- static void **AVR_Opcode_BRLT** (void)
- static void **AVR_Opcode_BRHS** (void)
- static void **AVR_Opcode_BRHC** (void)
- static void **AVR_Opcode_BRTS** (void)
- static void **AVR_Opcode_BRTC** (void)
- static void **AVR_Opcode_BRVS** (void)
- static void **AVR_Opcode_BRVC** (void)
- static void **AVR_Opcode_BRIE** (void)
- static void **AVR_Opcode_BRID** (void)
- static void **AVR_Opcode_MOV** (void)
- static void **AVR_Opcode_MOVW** (void)
- static void **AVR_Opcode_LDI** (void)
- static void **AVR_Opcode_LDS** (void)
- static void **AVR_Opcode_LD_X_Indirect** (void)
- static void **AVR_Opcode_LD_X_Indirect_Postinc** (void)
- static void **AVR_Opcode_LD_X_Indirect_Predec** (void)
- static void **AVR_Opcode_LD_Y_Indirect** (void)
- static void **AVR_Opcode_LD_Y_Indirect_Postinc** (void)
- static void **AVR_Opcode_LD_Y_Indirect_Predec** (void)
- static void **AVR_Opcode_LDD_Y** (void)
- static void **AVR_Opcode_LD_Z_Indirect** (void)
- static void **AVR_Opcode_LD_Z_Indirect_Postinc** (void)
- static void **AVR_Opcode_LD_Z_Indirect_Predec** (void)
- static void **AVR_Opcode_LDD_Z** (void)
- static void **AVR_Opcode_STS** (void)
- static void **AVR_Opcode_ST_X_Indirect** (void)
- static void **AVR_Opcode_ST_X_Indirect_Postinc** (void)
- static void **AVR_Opcode_ST_X_Indirect_Predec** (void)
- static void **AVR_Opcode_ST_Y_Indirect** (void)

- static void **AVR_Opcode_ST_Y_Indirect_Postinc** (void)
- static void **AVR_Opcode_ST_Y_Indirect_Predec** (void)
- static void **AVR_Opcode_STD_Y** (void)
- static void **AVR_Opcode_ST_Z_Indirect** (void)
- static void **AVR_Opcode_ST_Z_Indirect_Postinc** (void)
- static void **AVR_Opcode_ST_Z_Indirect_Predec** (void)
- static void **AVR_Opcode_STD_Z** (void)
- static void **AVR_Opcode_LPM** (void)
- static void **AVR_Opcode_LPM_Z** (void)
- static void **AVR_Opcode_LPM_Z_Postinc** (void)
- static void AVR_Opcode_ELPM (void)
- static void **AVR_Opcode_ELPM_Z** (void)
- static void **AVR_Opcode_ELPM_Z_Postinc** (void)
- static void AVR_Opcode_SPM (void)
- static void AVR_Opcode_SPM_Z_Postinc2 (void)
- static void **AVR_Opcode_IN** (void)
- static void **AVR_Opcode_OUT** (void)
- static void **AVR_Opcode_PUSH** (void)
- static void **AVR_Opcode_POP** (void)
- static void **AVR_Opcode_XCH** (void)
- static void **AVR_Opcode_LAS** (void)
- static void **AVR_Opcode_LAC** (void)
- static void **AVR_Opcode_LAT** (void)
- void **LSL_HalfCarry_Flag** (uint8_t R_)
- void **Left_Carry_Flag** (uint8_t R_)
- void **Rotate_Overflow_Flag** ()
- static void **AVR_Opcode_LSL** (void)
- void **Right_Carry_Flag** (uint8_t R_)
- static void **AVR_Opcode_LSR** (void)
- static void **AVR_Opcode_ROL** (void)
- static void **AVR_Opcode_ROR** (void)
- static void **AVR_Opcode_ASR** (void)
- static void **AVR_Opcode_SWAP** (void)
- static void **AVR_Opcode_BSET** (void)
- static void **AVR_Opcode_BCLR** (void)
- static void **AVR_Opcode_SBI** (void)
- static void **AVR_Opcode_CBI** (void)
- static void **AVR_Opcode_BST** (void)
- static void **AVR_Opcode_BLD** (void)
- static void **AVR_Opcode_BREAK** (void)
- static void **AVR_Opcode_SLEEP** (void)
- static void **AVR_Opcode_WDR** (void)
- AVR_Opcode AVR_Opcode_Function (uint16_t OP_)

  AVR_Opcode_Function.
- void AVR_RunOpcode (uint16_t OP_)

  AVR_RunOpcode.

4.39.1 Detailed Description

AVR CPU - Opcode implementation.

Definition in file avr_opcodes.c.

4.39.2 Function Documentation

4.39.2.1 static void AVR_Opcode_DES (void) [static]

ToDo - Implement DES

Definition at line 740 of file avr_opcodes.c.

4.39.2.2 static void AVR_Opcode_EICALL (void) [static]

! ToDo - Implement EIND calling!

Definition at line 843 of file avr_opcodes.c.

4.39.2.3 static void AVR_Opcode_EIJMP (void) [static]

ToDo - implement EIND instructions

Definition at line 778 of file avr_opcodes.c.

4.39.2.4 static void AVR_Opcode_ELPM (void) [static]

! ToDo - Add in RAMPZ register.

Definition at line 1466 of file avr_opcodes.c.

4.39.2.5 AVR_Opcode AVR_Opcode_Function (uint16_t *OP_*)

AVR_Opcode_Function.

Return a function pointer corresponding to the CPU logic for a given opcode.

Parameters

*OP_*	Opcode to return an "opcode execution" function pointer for

Returns

Opcode execution function pointer corresponding to the given opcode.

Definition at line 1838 of file avr_opcodes.c.

4.39.2.6 static void AVR_Opcode_SPM (void) [static]

! Implment later...

Definition at line 1517 of file avr_opcodes.c.

4.39.2.7 static void AVR_Opcode_SPM_Z_Postinc2 (void) [static]

! Implement later...

Definition at line 1523 of file avr_opcodes.c.

4.39.2.8 void AVR_RunOpcode (uint16_t *OP_*)

AVR_RunOpcode.

Execute the instruction corresponding to the provided opcode, on the provided CPU object. Note that the opcode must have just been decoded on the given CPU object before calling this function.

Parameters

*OP_*	Opcode to execute

Definition at line 2039 of file avr_opcodes.c.

4.40 avr_opcodes.c

```
00001 /*****************************************************************************
00002  *      (     (                       (     |
00003  *    )\ )  )\ )    (              )\ )    |
00004  *   (()/( (()/(    )\     (   (  (()/(    | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+-----------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00022 #include <stdint.h>
00023 #include <stdio.h>
00024 #include <stdlib.h>
00025
00026 #include "avr_cpu_print.h"
00027 #include "emu_config.h"
00028 #include "avr_opcodes.h"
00029
00030 //------------------------------------------------------------------------------
00031 #define DEBUG_PRINT(...)
00032
00033 //------------------------------------------------------------------------------
00034 static void AVR_Abort(void)
00035 {
00036     print_core_regs();
00037     exit(-1);
00038 }
00039
00040 //------------------------------------------------------------------------------
00041 static void Data_Write( uint16_t u16Addr_, uint8_t u8Val_ )
00042 {
00043     // Writing to RAM can be a tricky deal, because the address space is shared
00044     // between RAM, the core registers, and a bunch of peripheral I/O registers.
00045     DEBUG_PRINT("Write: 0x%04X=%02X\n", u16Addr_, u8Val_ );
00046     // Check to see if the write operation falls within the peripheral I/O range
00047     if (u16Addr_ >= 32 && u16Addr_ <= 255)
00048     {
00049         // I/O range - check to see if there's a peripheral installed at this address
00050         IOWriterList *pstIOWrite = stCPU.apstPeriphWriteTable[ u16Addr_ ];
00051
00052         // If there is a peripheral or peripherals
00053         if (pstIOWrite)
00054         {
00055             // Iterate through the list of installed peripherals at this address, and
00056             // call their write handler
00057             while (pstIOWrite)
00058             {
00059                 pstIOWrite->pfWriter( pstIOWrite->pvContext, (uint8_t)u16Addr_, u8Val_ );
00060                 pstIOWrite = pstIOWrite->next;
00061             }
00062         }
00063         // Otherwise, there is no peripheral -- just assume we can treat this as normal RAM.
00064         else
00065         {
00066             stCPU.pstRAM->au8RAM[ u16Addr_ ] = u8Val_;
00067         }
00068     }
00069     else if (u16Addr_ >= (stCPU.u32RAMSize + 256))
00070     {
00071         fprintf( stderr, "[Write Abort] RAM Address 0x%04X is out of range!\n", u16Addr_ );
00072         AVR_Abort();
00073     }
00074     // RAM address range - direct write-through.
00075     else
00076     {
00077         stCPU.pstRAM->au8RAM[ u16Addr_ ] = u8Val_;
00078     }
00079 }
00080
00081 //------------------------------------------------------------------------------
```

```
00082 static uint8_t Data_Read( uint16_t u16Addr_)
00083 {
00084     // Writing to RAM can be a tricky deal, because the address space is shared
00085     // between RAM, the core registers, and a bunch of peripheral I/O registers.
00086 
00087     // Check to see if the write operation falls within the peripheral I/O range
00088     DEBUG_PRINT( "Data Read: %04X\n", u16Addr_ );
00089     if (u16Addr_ >= 32 && u16Addr_ <= 255)
00090     {
00091         // I/O range - check to see if there's a peripheral installed at this address
00092         IOReaderList *pstIORead = stCPU.apstPeriphReadTable[ u16Addr_ ];
00093         DEBUG_PRINT( "Peripheral Read: 0x%04X\n", u16Addr_ );
00094         // If there is a peripheral or peripherals
00095         if (pstIORead)
00096         {
00097             DEBUG_PRINT(" Found peripheral\n");
00098             // Iterate through the list of installed peripherals at this address, and
00099             // call their read handler
00100             uint8_t u8Val;
00101             while (pstIORead)
00102             {
00103                 pstIORead->pfReader( pstIORead->pvContext,  (uint8_t)u16Addr_, &u8Val);
00104                 pstIORead = pstIORead->next;
00105             }
00106             return u8Val;
00107         }
00108         // Otherwise, there is no peripheral -- just assume we can treat this as normal RAM.
00109         else
00110         {
00111             DEBUG_PRINT(" No peripheral\n");
00112             return stCPU.pstRAM->au8RAM[ u16Addr_ ];
00113         }
00114     }
00115     else if (u16Addr_ >= (stCPU.u32RAMSize + 256))
00116     {
00117         fprintf( stderr, "[Write Abort] RAM Address 0x%04X is out of range!\n", u16Addr_ );
00118         AVR_Abort();
00119     }
00120     // RAM address range - direct read
00121     else
00122     {
00123         return stCPU.pstRAM->au8RAM[ u16Addr_ ];
00124     }
00125 }
00126 
00127 //------------------------------------------------------------------------------
00128 static void AVR_Opcode_NOP( void )
00129 {
00130     // Nop - do nothing.
00131 }
00132 
00133 //------------------------------------------------------------------------------
00134 inline void ADD_Half_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
00135 {
00136     stCPU.pstRAM->stRegisters.SREG.H =
00137             ( ((Rd_ & Rr_)    & 0x08 )
00138             | ((Rr_ & (~Result_)) & 0x08 )
00139             | (((~Result_) & Rd_) & 0x08) ) != false;
00140 }
00141 
00142 //------------------------------------------------------------------------------
00143 inline void ADD_Full_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
00144 {
00145     stCPU.pstRAM->stRegisters.SREG.C =
00146             ( ((Rd_ & Rr_)    & 0x80 )
00147             | ((Rr_ & (~Result_)) & 0x80 )
00148             | (((~Result_) & Rd_) & 0x80) ) != false;
00149 }
00150 
00151 //------------------------------------------------------------------------------
00152 inline void ADD_Overflow_Flag( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
00153 {
00154     stCPU.pstRAM->stRegisters.SREG.V =
00155              ( ((Rd_ & Rr_ & ~Result_)  & 0x80 )
00156              | ((~Rd_ & ~Rr_ & Result_) & 0x80 ) ) != 0;
00157 }
00158 
00159 //------------------------------------------------------------------------------
00160 inline void Signed_Flag( void )
00161 {
00162     unsigned int N = stCPU.pstRAM->stRegisters.SREG.N;
00163     unsigned int V = stCPU.pstRAM->stRegisters.SREG.V;
00164 
00165     stCPU.pstRAM->stRegisters.SREG.S = N ^ V;
00166 }
00167 
00168 //------------------------------------------------------------------------------
```

```
00169 inline void R8_Zero_Flag( uint8_t R_ )
00170 {
00171     stCPU.pstRAM->stRegisters.SREG.Z = (R_ == 0);
00172 }
00173
00174 //-----------------------------------------------------------------------------
00175 inline void R8_CPC_Zero_Flag( uint8_t R_ )
00176 {
00177     stCPU.pstRAM->stRegisters.SREG.Z = (stCPU.pstRAM->stRegisters.SREG.Z && (R_ == 0));
00178 }
00179
00180 //-----------------------------------------------------------------------------
00181 inline void R8_Negative_Flag( uint8_t R_ )
00182 {
00183     stCPU.pstRAM->stRegisters.SREG.N = ((R_ & 0x80) == 0x80);
00184 }
00185
00186 //-----------------------------------------------------------------------------
00187 static void AVR_Opcode_ADD( void )
00188 {
00189     uint8_t u8Result;
00190     uint8_t u8Rd = *(stCPU.Rd);
00191     uint8_t u8Rr = *(stCPU.Rr);
00192
00193     u8Result = u8Rd + u8Rr;
00194     *(stCPU.Rd) = u8Result;
00195
00196 // ---- Update flags ----
00197     ADD_Half_Carry(  u8Rd, u8Rr, u8Result );
00198     ADD_Full_Carry(  u8Rd, u8Rr, u8Result );
00199     ADD_Overflow_Flag(  u8Rd, u8Rr, u8Result );
00200     R8_Negative_Flag(  u8Result );
00201     R8_Zero_Flag(  u8Result );
00202     Signed_Flag();
00203 }
00204
00205 //-----------------------------------------------------------------------------
00206 static void AVR_Opcode_ADC( void )
00207 {
00208     uint8_t u8Result;
00209     uint8_t u8Rd = *(stCPU.Rd);
00210     uint8_t u8Rr = *(stCPU.Rr);
00211     uint8_t u8Carry = (stCPU.pstRAM->stRegisters.SREG.C);
00212
00213     u8Result = u8Rd + u8Rr + u8Carry;
00214     *(stCPU.Rd) = u8Result;
00215
00216 // ---- Update flags ----
00217     ADD_Half_Carry(  u8Rd, u8Rr, u8Result );
00218     ADD_Full_Carry(  u8Rd, u8Rr, u8Result );
00219     ADD_Overflow_Flag(  u8Rd, u8Rr, u8Result );
00220     R8_Negative_Flag(  u8Result );
00221     R8_Zero_Flag(  u8Result );
00222     Signed_Flag();
00223 }
00224
00225 //-----------------------------------------------------------------------------
00226 inline void R16_Negative_Flag( uint16_t Result_ )
00227 {
00228     stCPU.pstRAM->stRegisters.SREG.N =
00229             ((Result_ & 0x8000) != 0);
00230 }
00231
00232 //-----------------------------------------------------------------------------
00233 inline void R16_Zero_Flag( uint16_t Result_ )
00234 {
00235     stCPU.pstRAM->stRegisters.SREG.Z =
00236             (Result_ == 0);
00237 }
00238
00239 //-----------------------------------------------------------------------------
00240 inline void ADIW_Overflow_Flag( uint16_t Rd_, uint16_t Result_ )
00241 {
00242     stCPU.pstRAM->stRegisters.SREG.V =
00243             (((Rd_ & 0x8000) == 0) && ((Result_ & 0x8000) == 0x8000));
00244 }
00245
00246 //-----------------------------------------------------------------------------
00247 inline void ADIW_Carry_Flag( uint16_t Rd_, uint16_t Result_ )
00248 {
00249     stCPU.pstRAM->stRegisters.SREG.C =
00250             (((Rd_ & 0x8000) == 0x8000) && ((Result_ & 0x8000) == 0));
00251 }
00252
00253 //-----------------------------------------------------------------------------
00254 static void AVR_Opcode_ADIW( void )
00255 {
```

```
00256     uint16_t u16K = (stCPU.K);
00257     uint16_t u16Rd = *(stCPU.Rd16);
00258     uint16_t u16Result;
00259 
00260     u16Result = u16Rd + u16K;
00261     *(stCPU.Rd16) = u16Result;
00262 
00263 // ---- Update Flags ----
00264     ADIW_Carry_Flag(  u16Rd, u16Result);
00265     ADIW_Overflow_Flag(  u16Rd, u16Result );
00266     R16_Negative_Flag(  u16Result );
00267     R16_Zero_Flag(  u16Result );
00268     Signed_Flag();
00269 }
00270 
00271 //-------------------------------------------------------------------------
00272 inline void SUB_Overflow_Flag( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
00273 {
00274     stCPU.pstRAM->stRegisters.SREG.V =
00275             ( ((Rd_ & ~Rr_ & ~Result_) & 0x80 )
00276             | ((~Rd_ & Rr_ & Result_) & 0x80 ) ) != 0;
00277 }
00278 //-------------------------------------------------------------------------
00279 inline void SUB_Half_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
00280 {
00281     stCPU.pstRAM->stRegisters.SREG.H =
00282             ( ((~Rd_ & Rr_) & 0x08 )
00283             | ((Rr_ & Result_) & 0x08 )
00284             | ((Result_ & ~Rd_) & 0x08 ) ) == 0x08;
00285 }
00286 //-------------------------------------------------------------------------
00287 inline void SUB_Full_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
00288 {
00289     stCPU.pstRAM->stRegisters.SREG.C =
00290             ( ((~Rd_ & Rr_) & 0x80 )
00291             | ((Rr_ & Result_) & 0x80 )
00292             | ((Result_ & ~Rd_) & 0x80 ) ) == 0x80;
00293 }
00294 
00295 //-------------------------------------------------------------------------
00296 static void AVR_Opcode_SUB( void )
00297 {
00298     uint8_t u8Rd = *stCPU.Rd;
00299     uint8_t u8Rr = *stCPU.Rr;
00300     uint8_t u8Result = u8Rd - u8Rr;
00301 
00302     *stCPU.Rd = u8Result;
00303 
00304     //--Flags
00305     SUB_Half_Carry( u8Rd, u8Rr, u8Result);
00306     SUB_Full_Carry( u8Rd, u8Rr, u8Result);
00307     SUB_Overflow_Flag( u8Rd, u8Rr, u8Result);
00308     R8_Negative_Flag( u8Result);
00309     R8_Zero_Flag( u8Result);
00310     Signed_Flag();
00311 }
00312 
00313 //-------------------------------------------------------------------------
00314 static void AVR_Opcode_SUBI( void )
00315 {
00316     uint8_t u8Rd = *stCPU.Rd;
00317     uint8_t u8K = (uint8_t)stCPU.K;
00318     uint8_t u8Result = u8Rd - u8K;
00319 
00320     *stCPU.Rd = u8Result;
00321 
00322     //--Flags
00323     SUB_Half_Carry( u8Rd, u8K, u8Result);
00324     SUB_Full_Carry( u8Rd, u8K, u8Result);
00325     SUB_Overflow_Flag( u8Rd, u8K, u8Result);
00326     R8_Negative_Flag( u8Result);
00327     R8_Zero_Flag( u8Result);
00328     Signed_Flag();
00329 }
00330 
00331 //-------------------------------------------------------------------------
00332 static void AVR_Opcode_SBC( void )
00333 {
00334     uint8_t u8Rd = *stCPU.Rd;
00335     uint8_t u8Rr = *stCPU.Rr;
00336     uint8_t u8C = stCPU.pstRAM->stRegisters.SREG.C;
00337     uint8_t u8Result = u8Rd - u8Rr - u8C;
00338 
00339     *stCPU.Rd = u8Result;
00340 
00341     //--Flags
00342     SUB_Half_Carry( u8Rd, u8Rr, u8Result);
```

```
00343     SUB_Full_Carry( u8Rd, u8Rr, u8Result);
00344     SUB_Overflow_Flag( u8Rd, u8Rr, u8Result);
00345     R8_Negative_Flag( u8Result);
00346     if (u8Result)
00347     {
00348         stCPU.pstRAM->stRegisters.SREG.Z = 0;
00349     }
00350     Signed_Flag();
00351 }
00352
00353 //-----------------------------------------------------------------------------
00354 static void AVR_Opcode_SBCI( void )
00355 {
00356     uint8_t u8Rd = *stCPU.Rd;
00357     uint8_t u8K = (uint8_t)stCPU.K;
00358     uint8_t u8C = stCPU.pstRAM->stRegisters.SREG.C;
00359     uint8_t u8Result = u8Rd - u8K - u8C;
00360
00361     *stCPU.Rd = u8Result;
00362
00363     //--Flags
00364     SUB_Half_Carry( u8Rd, u8K, u8Result);
00365     SUB_Full_Carry( u8Rd, u8K, u8Result);
00366     SUB_Overflow_Flag( u8Rd, u8K, u8Result);
00367     R8_Negative_Flag( u8Result);
00368     if (u8Result)
00369     {
00370         stCPU.pstRAM->stRegisters.SREG.Z = 0;
00371     }
00372     Signed_Flag();
00373 }
00374
00375
00376 //-----------------------------------------------------------------------------
00377 inline void SBIW_Overflow_Flag( uint16_t Rd_, uint16_t Result_)
00378 {
00379     stCPU.pstRAM->stRegisters.SREG.V =
00380             ((Rd_ & 0x8000 ) == 0x8000) && ((Result_ & 0x8000) == 0);
00381
00382 }
00383
00384 //-----------------------------------------------------------------------------
00385 inline void SBIW_Full_Carry( uint16_t Rd_, uint16_t Result_)
00386 {
00387     stCPU.pstRAM->stRegisters.SREG.C =
00388             ((Rd_ & 0x8000 ) == 0) && ((Result_ & 0x8000) == 0x8000);
00389 }
00390
00391 //-----------------------------------------------------------------------------
00392 static void AVR_Opcode_SBIW( void )
00393 {
00394     uint16_t u16Rd = *stCPU.Rd16;
00395     uint16_t u16Result;
00396
00397     //fprintf( stderr, "SBIW: RD=[%4X], K=[%2X]\n", u16Rd, stCPU.K );
00398     u16Result = u16Rd - stCPU.K;
00399
00400     *stCPU.Rd16 = u16Result;
00401     //fprintf( stderr, "  Result=[%4X]\n", u16Result );
00402
00403     SBIW_Full_Carry( u16Rd, u16Result);
00404     SBIW_Overflow_Flag( u16Rd, u16Result);
00405     R16_Negative_Flag( u16Result);
00406     R16_Zero_Flag( u16Result);
00407     Signed_Flag();
00408
00409 }
00410
00411 //-----------------------------------------------------------------------------
00412 static void AVR_Opcode_AND( void )
00413 {
00414     uint8_t u8Rd = *stCPU.Rd;
00415     uint8_t u8Rr = *stCPU.Rr;
00416     uint8_t u8Result = u8Rd & u8Rr;
00417
00418     *stCPU.Rd = u8Result;
00419
00420     //--Update Status registers;
00421     stCPU.pstRAM->stRegisters.SREG.V = 0;
00422     R8_Negative_Flag(  u8Result );
00423     R8_Zero_Flag(  u8Result );
00424     Signed_Flag();
00425 }
00426
00427 //-----------------------------------------------------------------------------
00428 static void AVR_Opcode_ANDI( void )
00429 {
```

```
00430     uint8_t u8Rd = *stCPU.Rd;
00431     uint8_t u8Result = u8Rd & (uint8_t)stCPU.K;
00432
00433     *stCPU.Rd = u8Result;
00434
00435     //--Update Status registers;
00436     stCPU.pstRAM->stRegisters.SREG.V = 0;
00437     R8_Negative_Flag(  u8Result );
00438     R8_Zero_Flag(  u8Result );
00439     Signed_Flag();
00440 }
00441
00442 //-------------------------------------------------------------------------
00443 static void AVR_Opcode_OR( void )
00444 {
00445     uint8_t u8Rd = *stCPU.Rd;
00446     uint8_t u8Rr = *stCPU.Rr;
00447     uint8_t u8Result = u8Rd | u8Rr;
00448
00449     *stCPU.Rd = u8Result;
00450
00451     //--Update Status registers;
00452     stCPU.pstRAM->stRegisters.SREG.V = 0;
00453     R8_Negative_Flag(  u8Result );
00454     R8_Zero_Flag(  u8Result );
00455     Signed_Flag();
00456 }
00457
00458 //-------------------------------------------------------------------------
00459 static void AVR_Opcode_ORI( void )
00460 {
00461     uint8_t u8Rd = *stCPU.Rd;
00462     uint8_t u8Result = u8Rd | (uint8_t)stCPU.K;
00463
00464     *stCPU.Rd = u8Result;
00465
00466     //--Update Status registers;
00467     stCPU.pstRAM->stRegisters.SREG.V = 0;
00468     R8_Negative_Flag(  u8Result );
00469     R8_Zero_Flag(  u8Result );
00470     Signed_Flag();
00471 }
00472
00473 //-------------------------------------------------------------------------
00474 static void AVR_Opcode_EOR( void )
00475 {
00476     uint8_t u8Rd = *stCPU.Rd;
00477     uint8_t u8Rr = *stCPU.Rr;
00478     uint8_t u8Result = u8Rd ^ u8Rr;
00479
00480     *stCPU.Rd = u8Result;
00481
00482     //--Update Status registers;
00483     stCPU.pstRAM->stRegisters.SREG.V = 0;
00484     R8_Negative_Flag(  u8Result );
00485     R8_Zero_Flag(  u8Result );
00486     Signed_Flag();
00487 }
00488
00489 //-------------------------------------------------------------------------
00490 static void AVR_Opcode_COM( void )
00491 {
00492     // 1's complement.
00493     uint8_t u8Result = *stCPU.Rd;
00494     u8Result = (0xFF - u8Result);
00495
00496     *stCPU.Rd = u8Result;
00497
00498     //--Update Status registers;
00499     stCPU.pstRAM->stRegisters.SREG.V = 0;
00500     stCPU.pstRAM->stRegisters.SREG.C = 1;
00501     R8_Negative_Flag(  u8Result );
00502     R8_Zero_Flag(  u8Result );
00503     Signed_Flag();
00504 }
00505
00506 //-------------------------------------------------------------------------
00507 inline void NEG_Overflow_Flag( uint8_t u8Result_ )
00508 {
00509     stCPU.pstRAM->stRegisters.SREG.V = (u8Result_ == 0x80);
00510 }
00511
00512 //-------------------------------------------------------------------------
00513 inline void NEG_Carry_Flag( uint8_t u8Result_ )
00514 {
00515     stCPU.pstRAM->stRegisters.SREG.C = (u8Result_ != 0x00);
00516 }
```

```
00517
00518 //-------------------------------------------------------------------------
00519 static void AVR_Opcode_NEG( void )
00520 {
00521     // 2's complement.
00522     uint8_t u8Result = *stCPU.Rd;
00523     u8Result = (0 - u8Result);
00524
00525     *stCPU.Rd = u8Result;
00526
00527     //--Update Status registers;
00528     NEG_Overflow_Flag(  u8Result );
00529     NEG_Carry_Flag(  u8Result );
00530     R8_Negative_Flag(  u8Result );
00531     R8_Zero_Flag(  u8Result );
00532     Signed_Flag();
00533 }
00534
00535 //-------------------------------------------------------------------------
00536 static void AVR_Opcode_SBR( void )
00537 {
00538     // Set Bits in Register
00539     uint8_t u8Result = *stCPU.Rd;
00540     u8Result |= ((uint8_t)stCPU.K);
00541
00542     *stCPU.Rd = u8Result;
00543
00544     //--Update Status registers;
00545     stCPU.pstRAM->stRegisters.SREG.V = 0;
00546     R8_Negative_Flag(  u8Result );
00547     R8_Zero_Flag(  u8Result );
00548     Signed_Flag();
00549 }
00550
00551 //-------------------------------------------------------------------------
00552 static void AVR_Opcode_CBR( void )
00553 {
00554     // Clear Bits in Register
00555     uint8_t u8Result = *stCPU.Rd;
00556     u8Result &= ~((uint8_t)stCPU.K);
00557
00558     *stCPU.Rd = u8Result;
00559
00560     //--Update Status registers;
00561     stCPU.pstRAM->stRegisters.SREG.V = 0;
00562     R8_Negative_Flag(  u8Result );
00563     R8_Zero_Flag(  u8Result );
00564     Signed_Flag();
00565 }
00566
00567 //-------------------------------------------------------------------------
00568 inline void INC_Overflow_Flag( uint8_t u8Result_ )
00569 {
00570     stCPU.pstRAM->stRegisters.SREG.V = (u8Result_ == 0x80);
00571 }
00572
00573 //-------------------------------------------------------------------------
00574 static void AVR_Opcode_INC( void )
00575 {
00576     uint8_t u8Result;
00577     u8Result = *stCPU.Rd + 1;
00578
00579     *stCPU.Rd = u8Result;
00580
00581     //--Update Status registers;
00582     INC_Overflow_Flag(  u8Result );
00583     R8_Negative_Flag(  u8Result );
00584     R8_Zero_Flag(  u8Result );
00585     Signed_Flag();
00586 }
00587 //-------------------------------------------------------------------------
00588 inline void DEC_Overflow_Flag( uint8_t u8Result_ )
00589 {
00590     stCPU.pstRAM->stRegisters.SREG.V = (u8Result_ == 0x7F);
00591 }
00592 //-------------------------------------------------------------------------
00593 static void AVR_Opcode_DEC( void )
00594 {
00595     uint8_t u8Result;
00596     u8Result = *stCPU.Rd - 1;
00597
00598     *stCPU.Rd = u8Result;
00599
00600     //--Update Status registers;
00601     DEC_Overflow_Flag(  u8Result );
00602     R8_Negative_Flag(  u8Result );
00603     R8_Zero_Flag(  u8Result );
```

```
00604     Signed_Flag();
00605 }
00606
00607 //-------------------------------------------------------------------------
00608 static void AVR_Opcode_SER( void )
00609 {
00610     *stCPU.Rd = 0xFF;
00611 }
00612
00613 //-------------------------------------------------------------------------
00614 inline void Mul_Carry_Flag( uint16_t R_ )
00615 {
00616     stCPU.pstRAM->stRegisters.SREG.C = ((R_  & 0x8000) == 0x8000);
00617 }
00618
00619 //-------------------------------------------------------------------------
00620 inline void Mul_Zero_Flag( uint16_t R_ )
00621 {
00622     stCPU.pstRAM->stRegisters.SREG.Z = (R_ == 0);
00623 }
00624
00625 //-------------------------------------------------------------------------
00626 static void AVR_Opcode_MUL( void )
00627 {
00628     uint16_t u16Product;
00629     uint16_t u16R1;
00630     uint16_t u16R2;
00631
00632     u16R1 = *stCPU.Rd;
00633     u16R2 = *stCPU.Rr;
00634
00635     u16Product = u16R1 * u16R2;
00636
00637     stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = u16Product;
00638
00639     //-- Update Flags --
00640     Mul_Zero_Flag( u16Product);
00641     Mul_Carry_Flag( u16Product);
00642 }
00643
00644 //-------------------------------------------------------------------------
00645 static void AVR_Opcode_MULS( void )
00646 {
00647     int16_t s16Product;
00648     int16_t s16R1;
00649     int16_t s16R2;
00650
00651     s16R1 = (int8_t)*stCPU.Rd;
00652     s16R2 = (int8_t)*stCPU.Rr;
00653
00654     s16Product = s16R1 * s16R2;
00655
00656     stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = (uint16_t)s16Product;
00657
00658     //-- Update Flags --
00659     Mul_Zero_Flag( (uint16_t)s16Product);
00660     Mul_Carry_Flag( (uint16_t)s16Product);
00661 }
00662
00663 //-------------------------------------------------------------------------
00664 static void AVR_Opcode_MULSU( void )
00665 {
00666     int16_t s16Product;
00667     int16_t s16R1;
00668     uint16_t u16R2;
00669
00670     s16R1 = (int8_t)*stCPU.Rd;
00671     u16R2 = *stCPU.Rr;
00672
00673     s16Product = s16R1 * u16R2;
00674
00675     stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = (uint16_t)s16Product;
00676
00677     //-- Update Flags --
00678     Mul_Zero_Flag( (uint16_t)s16Product);
00679     Mul_Carry_Flag( (uint16_t)s16Product);
00680 }
00681
00682 //-------------------------------------------------------------------------
00683 static void AVR_Opcode_FMUL( void )
00684 {
00685     uint16_t u16Product;
00686     uint16_t u16R1;
00687     uint16_t u16R2;
00688
00689     u16R1 = *stCPU.Rd;
00690     u16R2 = *stCPU.Rr;
```

```
00691
00692     u16Product = u16R1 * u16R2;
00693
00694     stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = u16Product << 1;
00695
00696     //-- Update Flags --
00697     Mul_Zero_Flag( u16Product);
00698     Mul_Carry_Flag( u16Product);
00699 }
00700
00701 //-------------------------------------------------------------------------
00702 static void AVR_Opcode_FMULS( void )
00703 {
00704     int16_t s16Product;
00705     int16_t s16R1;
00706     int16_t s16R2;
00707
00708     s16R1 = (int8_t)*stCPU.Rd;
00709     s16R2 = (int8_t)*stCPU.Rr;
00710
00711     s16Product = s16R1 * s16R2;
00712
00713     stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = ((uint16_t)s16Product) << 1;
00714
00715     //-- Update Flags --
00716     Mul_Zero_Flag( (uint16_t)s16Product);
00717     Mul_Carry_Flag( (uint16_t)s16Product);
00718 }
00719
00720 //-------------------------------------------------------------------------
00721 static void AVR_Opcode_FMULSU( void )
00722 {
00723     int16_t s16Product;
00724     int16_t s16R1;
00725     uint16_t u16R2;
00726
00727     s16R1 = (int8_t)*stCPU.Rd;
00728     u16R2 = *stCPU.Rr;
00729
00730     s16Product = s16R1 * u16R2;
00731
00732     stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = ((uint16_t)s16Product) << 1;
00733
00734     //-- Update Flags --
00735     Mul_Zero_Flag( (uint16_t)s16Product);
00736     Mul_Carry_Flag( (uint16_t)s16Product);
00737 }
00738
00739 //-------------------------------------------------------------------------
00740 static void AVR_Opcode_DES( void )
00741 {
00743 }
00744
00745 //-------------------------------------------------------------------------
00746 static inline Unconditional_Jump( uint16_t u16Addr_ )
00747 {
00748     stCPU.u16PC = u16Addr_;
00749     stCPU.u16ExtraPC = 0;
00750 }
00751
00752 //-------------------------------------------------------------------------
00753 static inline Relative_Jump( uint16_t u16Offset_ )
00754 {
00755     // u16Offset_ Will always be 1 or 2, based on the size of the next opcode
00756     // in a program
00757
00758     stCPU.u16PC += u16Offset_;
00759     stCPU.u16ExtraPC = 0;
00760     stCPU.u16ExtraCycles += u16Offset_;
00761 }
00762
00763 //-------------------------------------------------------------------------
00764 static void AVR_Opcode_RJMP( void )
00765 {
00766     int32_t s32NewPC = (int32_t)stCPU.u16PC + (int32_t)stCPU.k_s + 1;
00767
00768     Unconditional_Jump(  (uint16_t)s32NewPC );
00769 }
00770
00771 //-------------------------------------------------------------------------
00772 static void AVR_Opcode_IJMP( void )
00773 {
00774     Unconditional_Jump(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z );
00775 }
00776
00777 //-------------------------------------------------------------------------
00778 static void AVR_Opcode_EIJMP( void )
```

```
00779 {
00781 }
00782
00783 //------------------------------------------------------------------------------
00784 static void AVR_Opcode_JMP( void )
00785 {
00786     Unconditional_Jump(  (uint16_t)stCPU.k );
00787 }
00788
00789 //------------------------------------------------------------------------------
00790 static void AVR_Opcode_RCALL( void )
00791 {
00792     // Push the next instruction address onto the stack
00793     uint16_t u16SP = (((uint16_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
00794                      (((uint16_t)stCPU.pstRAM->stRegisters.SPL.r));
00795
00796     uint16_t u16StoredPC = stCPU.u16PC + 1;
00797
00798     Data_Write(  u16SP, (uint8_t)(u16StoredPC & 0x00FF));
00799     Data_Write(  u16SP - 1, (uint8_t)(u16StoredPC >> 8));
00800
00801     // Stack is post-decremented
00802     u16SP -= 2;
00803
00804     // Set the new PC (relative call)
00805     int32_t s32NewPC = (int32_t)stCPU.u16PC + (int32_t)stCPU.k_s + 1;
00806     uint16_t u16NewPC = (uint16_t)s32NewPC;
00807
00808     // Store the new SP.
00809     stCPU.pstRAM->stRegisters.SPH.r = (u16SP >> 8);
00810     stCPU.pstRAM->stRegisters.SPL.r = (u16SP & 0x00FF);
00811
00812     // Set the new PC
00813     Unconditional_Jump(  u16NewPC );
00814 }
00815
00816 //------------------------------------------------------------------------------
00817 static void AVR_Opcode_ICALL( void )
00818 {
00819     // Push the next instruction address onto the stack
00820     uint16_t u16SP = (((uint16_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
00821                      (((uint16_t)stCPU.pstRAM->stRegisters.SPL.r));
00822
00823     uint16_t u16StoredPC = stCPU.u16PC + 1;
00824
00825     Data_Write(  u16SP, (uint8_t)(u16StoredPC & 0x00FF));
00826     Data_Write(  u16SP - 1, (uint8_t)(u16StoredPC >> 8));
00827
00828     // Stack is post-decremented
00829     u16SP -= 2;
00830
00831     // Set the new PC
00832     uint16_t u16NewPC = stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z;
00833
00834     // Store the new SP.
00835     stCPU.pstRAM->stRegisters.SPH.r = (u16SP >> 8);
00836     stCPU.pstRAM->stRegisters.SPL.r = (u16SP & 0x00FF);
00837
00838     // Set the new PC
00839     Unconditional_Jump(  u16NewPC );
00840 }
00841
00842 //------------------------------------------------------------------------------
00843 static void AVR_Opcode_EICALL( void )
00844 {
00846 }
00847
00848 //------------------------------------------------------------------------------
00849 static void AVR_Opcode_CALL( void )
00850 {
00851     // See ICALL for documentation
00852     uint16_t u16SP = (((uint16_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
00853                      (((uint16_t)stCPU.pstRAM->stRegisters.SPL.r));
00854
00855     uint16_t u16StoredPC = stCPU.u16PC + 2;
00856
00857     Data_Write(  u16SP, (uint8_t)(u16StoredPC & 0x00FF));
00858     Data_Write(  u16SP - 1, (uint8_t)(u16StoredPC >> 8));
00859
00860     u16SP -= 2;
00861
00862     uint16_t u16NewPC = stCPU.k;
00863
00864     stCPU.pstRAM->stRegisters.SPH.r = (u16SP >> 8);
00865     stCPU.pstRAM->stRegisters.SPL.r = (u16SP & 0x00FF);
00866
00867     Unconditional_Jump(  u16NewPC );
```

```
00868 }
00869
00870 //-----------------------------------------------------------------------------
00871 static void AVR_Opcode_RET( void )
00872 {
00873     // Pop the next instruction off of the stack, pre-incrementing
00874     uint16_t u16SP = (((uint16_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
00875                      (((uint16_t)stCPU.pstRAM->stRegisters.SPL.r));
00876     u16SP += 2;
00877
00878     uint16_t u16High = Data_Read(  u16SP - 1 );
00879     uint16_t u16Low = Data_Read(  u16SP );
00880     uint16_t u16NewPC = (u16High << 8) | u16Low;
00881
00882     stCPU.pstRAM->stRegisters.SPH.r = (u16SP >> 8);
00883     stCPU.pstRAM->stRegisters.SPL.r = (u16SP & 0x00FF);
00884
00885     // Set new PC based on address read from stack
00886     Unconditional_Jump(  u16NewPC );
00887 }
00888
00889 //-----------------------------------------------------------------------------
00890 static void AVR_Opcode_RETI( void )
00891 {
00892     uint16_t u16SP = (((uint16_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
00893                      (((uint16_t)stCPU.pstRAM->stRegisters.SPL.r));
00894     u16SP += 2;
00895
00896     uint16_t u16High = Data_Read(  u16SP - 1 );
00897     uint16_t u16Low = Data_Read(  u16SP );
00898     uint16_t u16NewPC = (u16High << 8) | u16Low;
00899
00900     stCPU.pstRAM->stRegisters.SPH.r = (u16SP >> 8);
00901     stCPU.pstRAM->stRegisters.SPL.r = (u16SP & 0x00FF);
00902
00903 //-- Enable interrupts
00904     stCPU.pstRAM->stRegisters.SREG.I = 1;
00905     Unconditional_Jump(  u16NewPC );
00906 }
00907
00908 //-----------------------------------------------------------------------------
00909 static void AVR_Opcode_CPSE( void )
00910 {
00911     if (*stCPU.Rr == *stCPU.Rd)
00912     {
00913         uint8_t u8NextOpSize = AVR_Opcode_Size( stCPU.pu16ROM[ stCPU.u16PC + 1 ] );
00914         Relative_Jump(  u8NextOpSize + 1 );
00915     }
00916 }
00917
00918 //-----------------------------------------------------------------------------
00919 inline void CP_Half_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
00920 {
00921     stCPU.pstRAM->stRegisters.SREG.H =
00922             ( ((~Rd_ & Rr_)    & 0x08 )
00923             | ((Rr_ & (Result_)) & 0x08 )
00924             | (((Result_) & ~Rd_) & 0x08) ) != false;
00925 }
00926
00927 //-----------------------------------------------------------------------------
00928 inline void CP_Full_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
00929 {
00930     stCPU.pstRAM->stRegisters.SREG.C =
00931             ( ((~Rd_ & Rr_)    & 0x80 )
00932             | ((Rr_ & (Result_)) & 0x80 )
00933             | (((Result_) & ~Rd_) & 0x80) ) != false;
00934 }
00935
00936 //-----------------------------------------------------------------------------
00937 inline void CP_Overflow_Flag( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
00938 {
00939     stCPU.pstRAM->stRegisters.SREG.V =
00940              ( ((Rd_ & ~Rr_ & ~Result_)  & 0x80 )
00941              | ((~Rd_ & Rr_ & Result_) & 0x80 ) ) != 0;
00942 }
00943
00944 //-----------------------------------------------------------------------------
00945 static void AVR_Opcode_CP( void )
00946 {
00947     // Compare
00948     uint8_t u8Result;
00949     uint8_t u8Rd = *stCPU.Rd;
00950     uint8_t u8Rr  = *stCPU.Rr;
00951
00952     u8Result = u8Rd - u8Rr;
00953
00954     //---
```

```
00955     CP_Half_Carry(  u8Rd, u8Rr, u8Result );
00956     CP_Overflow_Flag(  u8Rd, u8Rr, u8Result );
00957     CP_Full_Carry(  u8Rd, u8Rr, u8Result );
00958 
00959     R8_Zero_Flag(  u8Result );
00960     R8_Negative_Flag(  u8Result );
00961 
00962     Signed_Flag();
00963 }
00964 
00965 //-------------------------------------------------------------------------
00966 static void AVR_Opcode_CPC( void )
00967 {
00968     // Compare with carry
00969     uint8_t u8Result;
00970     uint8_t u8Rd = *stCPU.Rd;
00971     uint8_t u8Rr  = *stCPU.Rr;
00972     uint8_t u8C = (stCPU.pstRAM->stRegisters.SREG.C == 1);
00973 
00974     u8Result = u8Rd - u8Rr - u8C;
00975 
00976     //---
00977     CP_Half_Carry(  u8Rd, u8Rr, u8Result );
00978     CP_Overflow_Flag(  u8Rd, u8Rr, u8Result );
00979     CP_Full_Carry(  u8Rd, u8Rr, u8Result );
00980 
00981     R8_CPC_Zero_Flag(  u8Result );
00982     R8_Negative_Flag(  u8Result );
00983 
00984     Signed_Flag();
00985 }
00986 
00987 //-------------------------------------------------------------------------
00988 static void AVR_Opcode_CPI( void )
00989 {
00990     // Compare with immediate
00991     uint8_t u8Result;
00992     uint8_t u8Rd = *stCPU.Rd;
00993     uint8_t u8K  = stCPU.K;
00994 
00995     u8Result = u8Rd - u8K;
00996 
00997     //---
00998     CP_Half_Carry(  u8Rd, u8K, u8Result );
00999     CP_Overflow_Flag(  u8Rd, u8K, u8Result );
01000     CP_Full_Carry(  u8Rd, u8K, u8Result );
01001 
01002     R8_Zero_Flag(  u8Result );
01003     R8_Negative_Flag(  u8Result );
01004 
01005     Signed_Flag();
01006 }
01007 
01008 //-------------------------------------------------------------------------
01009 static void AVR_Opcode_SBRC( void )
01010 {
01011     // Skip if Bit in IO register clear
01012     if ((*stCPU.Rd & (1 << stCPU.b)) == 0)
01013     {
01014         uint8_t u8NextOpSize = AVR_Opcode_Size( stCPU.pu16ROM[ stCPU.u16PC + 1 ] );
01015         Relative_Jump(  u8NextOpSize + 1 );
01016     }
01017 }
01018 
01019 //-------------------------------------------------------------------------
01020 static void AVR_Opcode_SBRS( void )
01021 {
01022     // Skip if Bit in IO register set
01023     if ((*stCPU.Rd & (1 << stCPU.b)) != 0)
01024     {
01025         uint8_t u8NextOpSize = AVR_Opcode_Size( stCPU.pu16ROM[ stCPU.u16PC + 1 ] );
01026         Relative_Jump(  u8NextOpSize + 1 );
01027     }
01028 }
01029 
01030 //-------------------------------------------------------------------------
01031 static void AVR_Opcode_SBIC( void )
01032 {
01033     // Skip if Bit in IO register clear
01034     uint8_t u8IOVal = Data_Read(  32 + stCPU.A );
01035     if ((u8IOVal & (1 << stCPU.b)) == 0)
01036     {
01037         uint8_t u8NextOpSize = AVR_Opcode_Size( stCPU.pu16ROM[ stCPU.u16PC + 1 ] );
01038         Relative_Jump(  u8NextOpSize + 1 );
01039     }
01040 }
01041 
```

```
01042 //-----------------------------------------------------------------------------
01043 static void AVR_Opcode_SBIS( void )
01044 {
01045     // Skip if Bit in IO register set
01046     uint8_t u8IOVal = Data_Read(  32 + stCPU.A );
01047     if ((u8IOVal & (1 << stCPU.b)) != 0)
01048     {
01049         uint8_t u8NextOpSize = AVR_Opcode_Size( stCPU.pu16ROM[ stCPU.u16PC + 1 ] );
01050         Relative_Jump(  u8NextOpSize + 1 );
01051     }
01052 }
01053
01054 //-----------------------------------------------------------------------------
01055 static inline Conditional_Branch( void )
01056 {
01057     stCPU.u16PC = (uint16_t)((int16_t)stCPU.u16PC + stCPU.k_s + 1);
01058     stCPU.u16ExtraPC = 0;
01059     stCPU.u16ExtraCycles++;
01060 }
01061
01062 //-----------------------------------------------------------------------------
01063 static void AVR_Opcode_BRBS( void )
01064 {
01065     if (0 != (stCPU.pstRAM->stRegisters.SREG.r & (1 << stCPU.b)))
01066     {
01067         Conditional_Branch();
01068     }
01069 }
01070
01071 //-----------------------------------------------------------------------------
01072 static void AVR_Opcode_BRBC( void )
01073 {
01074     if (0 == (stCPU.pstRAM->stRegisters.SREG.r & (1 << stCPU.b)))
01075     {
01076         Conditional_Branch();
01077     }
01078 }
01079
01080 //-----------------------------------------------------------------------------
01081 static void AVR_Opcode_BREQ( void )
01082 {
01083     if (1 == stCPU.pstRAM->stRegisters.SREG.Z)
01084     {
01085         Conditional_Branch();
01086     }
01087 }
01088
01089 //-----------------------------------------------------------------------------
01090 static void AVR_Opcode_BRNE( void )
01091 {
01092     if (0 == stCPU.pstRAM->stRegisters.SREG.Z)
01093     {
01094         Conditional_Branch();
01095     }
01096 }
01097
01098 //-----------------------------------------------------------------------------
01099 static void AVR_Opcode_BRCS( void )
01100 {
01101     if (1 == stCPU.pstRAM->stRegisters.SREG.C)
01102     {
01103         Conditional_Branch();
01104     }
01105 }
01106
01107 //-----------------------------------------------------------------------------
01108 static void AVR_Opcode_BRCC( void )
01109 {
01110     if (0 == stCPU.pstRAM->stRegisters.SREG.C)
01111     {
01112         Conditional_Branch();
01113     }
01114 }
01115
01116 //-----------------------------------------------------------------------------
01117 static void AVR_Opcode_BRSH( void )
01118 {
01119     if (0 == stCPU.pstRAM->stRegisters.SREG.C)
01120     {
01121         Conditional_Branch();
01122     }
01123 }
01124
01125 //-----------------------------------------------------------------------------
01126 static void AVR_Opcode_BRLO( void )
01127 {
01128     if (1 == stCPU.pstRAM->stRegisters.SREG.C)
```

```
01129     {
01130         Conditional_Branch();
01131     }
01132 }
01133
01134 //------------------------------------------------------------------------------
01135 static void AVR_Opcode_BRMI( void )
01136 {
01137     if (1 == stCPU.pstRAM->stRegisters.SREG.N)
01138     {
01139         Conditional_Branch();
01140     }
01141 }
01142
01143 //------------------------------------------------------------------------------
01144 static void AVR_Opcode_BRPL( void )
01145 {
01146     if (0 == stCPU.pstRAM->stRegisters.SREG.N)
01147     {
01148         Conditional_Branch();
01149     }
01150 }
01151
01152 //------------------------------------------------------------------------------
01153 static void AVR_Opcode_BRGE( void )
01154 {
01155     if (0 == stCPU.pstRAM->stRegisters.SREG.S)
01156     {
01157         Conditional_Branch();
01158     }
01159 }
01160
01161 //------------------------------------------------------------------------------
01162 static void AVR_Opcode_BRLT( void )
01163 {
01164     if (1 == stCPU.pstRAM->stRegisters.SREG.S)
01165     {
01166         Conditional_Branch();
01167     }
01168 }
01169
01170 //------------------------------------------------------------------------------
01171 static void AVR_Opcode_BRHS( void )
01172 {
01173     if (1 == stCPU.pstRAM->stRegisters.SREG.H)
01174     {
01175         Conditional_Branch();
01176     }
01177 }
01178
01179 //------------------------------------------------------------------------------
01180 static void AVR_Opcode_BRHC( void )
01181 {
01182     if (0 == stCPU.pstRAM->stRegisters.SREG.H)
01183     {
01184         Conditional_Branch();
01185     }
01186 }
01187
01188 //------------------------------------------------------------------------------
01189 static void AVR_Opcode_BRTS( void )
01190 {
01191     if (1 == stCPU.pstRAM->stRegisters.SREG.T)
01192     {
01193         Conditional_Branch();
01194     }
01195 }
01196
01197 //------------------------------------------------------------------------------
01198 static void AVR_Opcode_BRTC( void )
01199 {
01200     if (0 == stCPU.pstRAM->stRegisters.SREG.T)
01201     {
01202         Conditional_Branch();
01203     }
01204 }
01205
01206 //------------------------------------------------------------------------------
01207 static void AVR_Opcode_BRVS( void )
01208 {
01209     if (1 == stCPU.pstRAM->stRegisters.SREG.V)
01210     {
01211         Conditional_Branch();
01212     }
01213 }
01214
01215 //------------------------------------------------------------------------------
```

```
01216 static void AVR_Opcode_BRVC( void )
01217 {
01218     if (0 == stCPU.pstRAM->stRegisters.SREG.V)
01219     {
01220         Conditional_Branch();
01221     }
01222 }
01223
01224 //-----------------------------------------------------------------------
01225 static void AVR_Opcode_BRIE( void )
01226 {
01227     if (1 == stCPU.pstRAM->stRegisters.SREG.I)
01228     {
01229         Conditional_Branch();
01230     }
01231 }
01232
01233 //-----------------------------------------------------------------------
01234 static void AVR_Opcode_BRID( void )
01235 {
01236     if (0 == stCPU.pstRAM->stRegisters.SREG.I)
01237     {
01238         Conditional_Branch();
01239     }
01240 }
01241
01242 //-----------------------------------------------------------------------
01243 static void AVR_Opcode_MOV( void )
01244 {
01245     *stCPU.Rd = *stCPU.Rr;
01246 }
01247
01248 //-----------------------------------------------------------------------
01249 static void AVR_Opcode_MOVW( void )
01250 {
01251     *stCPU.Rd16 = *stCPU.Rr16;
01252 }
01253
01254 //-----------------------------------------------------------------------
01255 static void AVR_Opcode_LDI( void )
01256 {
01257     *stCPU.Rd = stCPU.K;
01258 }
01259
01260 //-----------------------------------------------------------------------
01261 static void AVR_Opcode_LDS( void )
01262 {
01263     *stCPU.Rd = Data_Read(  stCPU.K );
01264 }
01265
01266 //-----------------------------------------------------------------------
01267 static void AVR_Opcode_LD_X_Indirect( void )
01268 {
01269     *stCPU.Rd =
01270             Data_Read(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.X );
01271 }
01272
01273 //-----------------------------------------------------------------------
01274 static void AVR_Opcode_LD_X_Indirect_Postinc( void )
01275 {
01276     *stCPU.Rd =
01277         Data_Read(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.X++ );
01278 }
01279
01280 //-----------------------------------------------------------------------
01281 static void AVR_Opcode_LD_X_Indirect_Predec( void )
01282 {
01283     *stCPU.Rd =
01284         Data_Read(  --stCPU.pstRAM->stRegisters.CORE_REGISTERS.X );
01285 }
01286
01287 //-----------------------------------------------------------------------
01288 static void AVR_Opcode_LD_Y_Indirect( void )
01289 {
01290     *stCPU.Rd =
01291         Data_Read(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y );
01292 }
01293
01294 //-----------------------------------------------------------------------
01295 static void AVR_Opcode_LD_Y_Indirect_Postinc( void )
01296 {
01297     *stCPU.Rd =
01298         Data_Read(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y++ );
01299 }
01300
01301 //-----------------------------------------------------------------------
01302 static void AVR_Opcode_LD_Y_Indirect_Predec( void )
```

```
01303 {
01304     *stCPU.Rd =
01305         Data_Read(  --stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y );
01306 }
01307
01308 //-----------------------------------------------------------------------------
01309 static void AVR_Opcode_LDD_Y( void )
01310 {
01311     *stCPU.Rd =
01312         Data_Read(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y + stCPU.q );
01313 }
01314
01315 //-----------------------------------------------------------------------------
01316 static void AVR_Opcode_LD_Z_Indirect( void )
01317 {
01318     *stCPU.Rd =
01319         Data_Read(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z );
01320 }
01321
01322 //-----------------------------------------------------------------------------
01323 static void AVR_Opcode_LD_Z_Indirect_Postinc( void )
01324 {
01325     *stCPU.Rd =
01326         Data_Read(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z++ );
01327
01328 }
01329
01330 //-----------------------------------------------------------------------------
01331 static void AVR_Opcode_LD_Z_Indirect_Predec( void )
01332 {
01333     *stCPU.Rd =
01334         Data_Read(  --stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z );
01335 }
01336
01337 //-----------------------------------------------------------------------------
01338 static void AVR_Opcode_LDD_Z( void )
01339 {
01340     *stCPU.Rd =
01341         Data_Read(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z + stCPU.q );
01342 }
01343
01344 //-----------------------------------------------------------------------------
01345 static void AVR_Opcode_STS( void )
01346 {
01347     Data_Write(  stCPU.K, *stCPU.Rd );
01348 }
01349
01350 //-----------------------------------------------------------------------------
01351 static void AVR_Opcode_ST_X_Indirect( void )
01352 {
01353     Data_Write(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.X, *stCPU.Rd );
01354 }
01355
01356 //-----------------------------------------------------------------------------
01357 static void AVR_Opcode_ST_X_Indirect_Postinc( void )
01358 {
01359     Data_Write(   stCPU.pstRAM->stRegisters.CORE_REGISTERS.X++, *stCPU.Rd );
01360 }
01361
01362 //-----------------------------------------------------------------------------
01363 static void AVR_Opcode_ST_X_Indirect_Predec( void )
01364 {
01365     Data_Write(  --stCPU.pstRAM->stRegisters.CORE_REGISTERS.X, *stCPU.Rd );
01366 }
01367
01368 //-----------------------------------------------------------------------------
01369 static void AVR_Opcode_ST_Y_Indirect( void )
01370 {
01371     Data_Write(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y, *stCPU.Rd );
01372 }
01373
01374 //-----------------------------------------------------------------------------
01375 static void AVR_Opcode_ST_Y_Indirect_Postinc( void )
01376 {
01377     Data_Write(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y++, *stCPU.Rd );
01378 }
01379
01380 //-----------------------------------------------------------------------------
01381 static void AVR_Opcode_ST_Y_Indirect_Predec( void )
01382 {
01383     Data_Write(  --stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y, *stCPU.Rd );
01384 }
01385
01386 //-----------------------------------------------------------------------------
01387 static void AVR_Opcode_STD_Y( void )
01388 {
01389     Data_Write(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y + stCPU.q, *stCPU.Rd );
```

```
01390 }
01391
01392 //-----------------------------------------------------------------------------
01393 static void AVR_Opcode_ST_Z_Indirect( void )
01394 {
01395     Data_Write(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z, *stCPU.Rd );
01396 }
01397
01398 //-----------------------------------------------------------------------------
01399 static void AVR_Opcode_ST_Z_Indirect_Postinc( void )
01400 {
01401     Data_Write(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z++ , *stCPU.Rd );
01402 }
01403
01404 //-----------------------------------------------------------------------------
01405 static void AVR_Opcode_ST_Z_Indirect_Predec( void )
01406 {
01407     Data_Write(  --stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z , *stCPU.Rd );
01408 }
01409
01410 //-----------------------------------------------------------------------------
01411 static void AVR_Opcode_STD_Z( void )
01412 {
01413     Data_Write(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z + stCPU.q, *stCPU.Rd );
01414 }
01415
01416 //-----------------------------------------------------------------------------
01417 static void AVR_Opcode_LPM( void )
01418 {
01419     uint8_t u8Temp;
01420     if (stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z & 0x0001)
01421     {
01422         u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] >> 8);
01423     }
01424     else
01425     {
01426         u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] & 0x00FF);
01427     }
01428
01429     stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0 = u8Temp;
01430 }
01431
01432 //-----------------------------------------------------------------------------
01433 static void AVR_Opcode_LPM_Z( void )
01434 {
01435     uint8_t u8Temp;
01436     if (stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z & 0x0001)
01437     {
01438         u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] >> 8);
01439     }
01440     else
01441     {
01442         u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] & 0x00FF);
01443     }
01444
01445     *stCPU.Rd = u8Temp;
01446 }
01447
01448 //-----------------------------------------------------------------------------
01449 static void AVR_Opcode_LPM_Z_Postinc( void )
01450 {
01451     uint8_t u8Temp;
01452     if (stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z & 0x0001)
01453     {
01454         u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] >> 8);
01455     }
01456     else
01457     {
01458         u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] & 0x00FF);
01459     }
01460
01461     *stCPU.Rd = u8Temp;
01462     stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z++;
01463 }
01464
01465 //-----------------------------------------------------------------------------
01466 static void AVR_Opcode_ELPM( void )
01467 {
01469     uint8_t u8Temp;
01470     if (stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z & 0x0001)
01471     {
01472         u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] >> 8);
01473     }
01474     else
01475     {
01476         u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] & 0x00FF);
01477     }
```

```
01478
01479     stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0 = u8Temp;
01480 }
01481
01482 //------------------------------------------------------------------------
01483 static void AVR_Opcode_ELPM_Z( void )
01484 {
01485     uint8_t u8Temp;
01486     if (stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z & 0x0001)
01487     {
01488         u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] >> 8);
01489     }
01490     else
01491     {
01492         u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] & 0x00FF);
01493     }
01494
01495     *stCPU.Rd = u8Temp;
01496 }
01497
01498 //------------------------------------------------------------------------
01499 static void AVR_Opcode_ELPM_Z_Postinc( void )
01500 {
01501     uint8_t u8Temp;
01502     if (stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z & 0x0001)
01503     {
01504         u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] >> 8);
01505     }
01506     else
01507     {
01508         u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] & 0x00FF);
01509     }
01510
01511     *stCPU.Rd = u8Temp;
01512
01513     stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z++;
01514 }
01515
01516 //------------------------------------------------------------------------
01517 static void AVR_Opcode_SPM( void )
01518 {
01520 }
01521
01522 //------------------------------------------------------------------------
01523 static void AVR_Opcode_SPM_Z_Postinc2( void )
01524 {
01526 }
01527
01528 //------------------------------------------------------------------------
01529 static void AVR_Opcode_IN( void )
01530 {
01531     *stCPU.Rd = Data_Read(  32 + stCPU.A );
01532 }
01533
01534 //------------------------------------------------------------------------
01535 static void AVR_Opcode_OUT( void )
01536 {
01537      Data_Write(  32 + stCPU.A , *stCPU.Rd );
01538 }
01539
01540 //------------------------------------------------------------------------
01541 static void AVR_Opcode_PUSH( void )
01542 {
01543     uint16_t u16SP = (stCPU.pstRAM->stRegisters.SPL.r) |
01544                      ((uint16_t)(stCPU.pstRAM->stRegisters.SPH.r) << 8);
01545
01546     // Store contents from SP to destination register
01547     Data_Write(  u16SP, *stCPU.Rd );
01548
01549     // Postdecrement the SP
01550     u16SP--;
01551
01552     // Update the SP registers
01553     stCPU.pstRAM->stRegisters.SPH.r = (uint8_t)(u16SP >> 8);
01554     stCPU.pstRAM->stRegisters.SPL.r = (uint8_t)(u16SP & 0x00FF);
01555 }
01556
01557 //------------------------------------------------------------------------
01558 static void AVR_Opcode_POP( void )
01559 {
01560     // Preincrement the SP
01561     uint16_t u16SP = (stCPU.pstRAM->stRegisters.SPL.r) |
01562                      ((uint16_t)(stCPU.pstRAM->stRegisters.SPH.r) << 8);
01563     u16SP++;
01564
01565     // Load contents from SP to destination register
01566     *stCPU.Rd = Data_Read(  u16SP );
```

```
01567
01568     // Update the SP registers
01569     stCPU.pstRAM->stRegisters.SPH.r = (uint8_t)(u16SP >> 8);
01570     stCPU.pstRAM->stRegisters.SPL.r = (uint8_t)(u16SP & 0x00FF);
01571 }
01572
01573 //------------------------------------------------------------------------------
01574 static void AVR_Opcode_XCH( void )
01575 {
01576     uint8_t u8Z;
01577     uint8_t u8Temp;
01578     uint16_t u16Addr = stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z;
01579
01580     u8Z = Data_Read(  u16Addr );
01581     u8Temp = *stCPU.Rd;
01582
01583     *stCPU.Rd = u8Z;
01584     Data_Write(  u16Addr, u8Temp );
01585 }
01586
01587 //------------------------------------------------------------------------------
01588 static void AVR_Opcode_LAS( void )
01589 {
01590     uint8_t u8Z;
01591     uint8_t u8Temp;
01592
01593     uint16_t u16Addr = stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z;
01594
01595     u8Z = Data_Read(  u16Addr );
01596     u8Temp = *stCPU.Rd | u8Z;
01597
01598     *stCPU.Rd = u8Z;
01599     Data_Write(  u16Addr, u8Temp );
01600 }
01601
01602 //------------------------------------------------------------------------------
01603 static void AVR_Opcode_LAC( void )
01604 {
01605     uint8_t u8Z;
01606     uint8_t u8Temp;
01607
01608     uint16_t u16Addr = stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z;
01609
01610     u8Z = Data_Read(  u16Addr );
01611     u8Temp = *stCPU.Rd & ~(u8Z);
01612     *stCPU.Rd = u8Z;
01613
01614     Data_Write(  u16Addr, u8Temp );
01615 }
01616
01617 //------------------------------------------------------------------------------
01618 static void AVR_Opcode_LAT( void )
01619 {
01620     uint8_t u8Z;
01621     uint8_t u8Temp;
01622
01623     uint16_t u16Addr = stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z;
01624
01625     u8Z = Data_Read(  u16Addr );
01626     u8Temp = *stCPU.Rd ^ u8Z;
01627     *stCPU.Rd = u8Z;
01628
01629     Data_Write(  u16Addr, u8Temp );
01630 }
01631
01632 //------------------------------------------------------------------------------
01633 inline void LSL_HalfCarry_Flag( uint8_t R_ )
01634 {
01635     stCPU.pstRAM->stRegisters.SREG.H = ((R_ & 0x08) == 0x08);
01636 }
01637
01638 //------------------------------------------------------------------------------
01639 inline void Left_Carry_Flag( uint8_t R_  )
01640 {
01641     stCPU.pstRAM->stRegisters.SREG.C = ((R_ & 0x80) == 0x80);
01642 }
01643
01644 //------------------------------------------------------------------------------
01645 inline void Rotate_Overflow_Flag()
01646 {
01647     stCPU.pstRAM->stRegisters.SREG.V = ( stCPU.pstRAM->stRegisters.SREG.N ^ stCPU.pstRAM->stRegisters.SREG.
    C );
01648 }
01649
01650 //------------------------------------------------------------------------------
01651 static void AVR_Opcode_LSL( void )
01652 {
```

```
01653     // Logical shift left
01654     uint8_t u8Result = 0;
01655     uint8_t u8Temp = *stCPU.Rd;
01656
01657     u8Result = (u8Temp << 1);
01658     *stCPU.Rd = u8Result;
01659
01660     // ---- Update flags ----
01661     LSL_HalfCarry_Flag(  u8Result);
01662     Left_Carry_Flag( u8Temp);
01663
01664     R8_Negative_Flag(  u8Result );
01665     R8_Zero_Flag(  u8Result );
01666     Rotate_Overflow_Flag();
01667     Signed_Flag();
01668 }
01669
01670 //-------------------------------------------------------------------------
01671 inline void Right_Carry_Flag( uint8_t R_  )
01672 {
01673     stCPU.pstRAM->stRegisters.SREG.C = ((R_ & 0x01) == 0x01);
01674 }
01675
01676 //-------------------------------------------------------------------------
01677 static void AVR_Opcode_LSR( void )
01678 {
01679     // Logical shift left
01680     uint8_t u8Result = 0;
01681     uint8_t u8Temp = *stCPU.Rd;
01682
01683     u8Result = (u8Temp >> 1);
01684     *stCPU.Rd = u8Result;
01685
01686     // ---- Update flags ----
01687     Right_Carry_Flag(  u8Temp );
01688     stCPU.pstRAM->stRegisters.SREG.N = 0;
01689     R8_Zero_Flag(  u8Result );
01690     Rotate_Overflow_Flag();
01691     Signed_Flag();
01692 }
01693
01694 //-------------------------------------------------------------------------
01695 static void AVR_Opcode_ROL( void )
01696 {
01697     // Rotate left through carry
01698     uint8_t u8Result = 0;
01699     uint8_t u8Temp = *stCPU.Rd;
01700
01701     u8Result = (u8Temp << 1);
01702     if (stCPU.pstRAM->stRegisters.SREG.C)
01703     {
01704         u8Result |= 0x01;
01705     }
01706     *stCPU.Rd = u8Result;
01707
01708     // ---- Update flags ----
01709     Left_Carry_Flag(  u8Temp );
01710     R8_Negative_Flag(  u8Result );
01711     R8_Zero_Flag(  u8Result );
01712     Rotate_Overflow_Flag();
01713     Signed_Flag();
01714 }
01715
01716 //-------------------------------------------------------------------------
01717 static void AVR_Opcode_ROR( void )
01718 {
01719     // Rotate right through carry
01720     uint8_t u8Result = 0;
01721     uint8_t u8Temp = *stCPU.Rd;
01722
01723     u8Result = (u8Temp >> 1);
01724     if (stCPU.pstRAM->stRegisters.SREG.C)
01725     {
01726         u8Result |= 0x80;
01727     }
01728     *stCPU.Rd = u8Result;
01729
01730     // ---- Update flags ----
01731     Right_Carry_Flag(  u8Temp );
01732     R8_Negative_Flag(  u8Result );
01733     R8_Zero_Flag(  u8Result );
01734     Rotate_Overflow_Flag();
01735     Signed_Flag();
01736 }
01737
01738 //-------------------------------------------------------------------------
01739 static void AVR_Opcode_ASR( void )
```

```
01740 {
01741     // Shift all bits to the right, keeping sign bit intact
01742     uint8_t u8Result;
01743     uint8_t u8Temp = *stCPU.Rd;
01744     u8Result = (u8Temp & 0x80) | (u8Temp >> 1);
01745     *stCPU.Rd = u8Result;
01746 
01747     // ---- Update flags ----
01748     Right_Carry_Flag(  u8Temp );
01749     R8_Negative_Flag(  u8Result );
01750     R8_Zero_Flag(  u8Result );
01751     Rotate_Overflow_Flag();
01752     Signed_Flag();
01753 }
01754 
01755 //-------------------------------------------------------------------------
01756 static void AVR_Opcode_SWAP( void )
01757 {
01758     uint8_t u8temp;
01759     u8temp = ((*stCPU.Rd) >> 4) |
01760              ((*stCPU.Rd) << 4) ;
01761 
01762     *stCPU.Rd = u8temp;
01763 }
01764 
01765 //-------------------------------------------------------------------------
01766 static void AVR_Opcode_BSET( void )
01767 {
01768     stCPU.pstRAM->stRegisters.SREG.r |= (1 << stCPU.b);
01769 }
01770 
01771 //-------------------------------------------------------------------------
01772 static void AVR_Opcode_BCLR( void )
01773 {
01774     stCPU.pstRAM->stRegisters.SREG.r &= ~(1 << stCPU.b);
01775 }
01776 
01777 //-------------------------------------------------------------------------
01778 static void AVR_Opcode_SBI( void )
01779 {
01780     uint8_t u8Temp = Data_Read(  stCPU.A + 32 );
01781     u8Temp |= (1 << stCPU.b);
01782     Data_Write(  stCPU.A + 32, u8Temp );
01783 }
01784 
01785 //-------------------------------------------------------------------------
01786 static void AVR_Opcode_CBI( void )
01787 {
01788     uint8_t u8Temp = Data_Read(  stCPU.A + 32 );
01789     u8Temp &= ~(1 << stCPU.b);
01790     Data_Write(  stCPU.A + 32, u8Temp );
01791 }
01792 
01793 //-------------------------------------------------------------------------
01794 static void AVR_Opcode_BST( void )
01795 {
01796     if ((*stCPU.Rd) & (1 << stCPU.b))
01797     {
01798         stCPU.pstRAM->stRegisters.SREG.T = 1;
01799     }
01800     else
01801     {
01802         stCPU.pstRAM->stRegisters.SREG.T = 0;
01803     }
01804 }
01805 
01806 //-------------------------------------------------------------------------
01807 static void AVR_Opcode_BLD( void )
01808 {
01809     if (stCPU.pstRAM->stRegisters.SREG.T)
01810     {
01811         *(stCPU.Rd) |= (1 << stCPU.b);
01812     }
01813     else
01814     {
01815         *(stCPU.Rd) &= ~(1 << stCPU.b);
01816     }
01817 }
01818 
01819 //-------------------------------------------------------------------------
01820 static void AVR_Opcode_BREAK( void )
01821 {
01822     // Unimplemented - since this requires debugging HW...
01823 }
01824 
01825 //-------------------------------------------------------------------------
01826 static void AVR_Opcode_SLEEP( void )
```

```
01827 {
01828     stCPU.bAsleep = true;
01829 }
01830
01831 //-----------------------------------------------------------------------
01832 static void AVR_Opcode_WDR( void )
01833 {
01834     stCPU.u32WDTCount = 0;    // Reset watchdog timer counter
01835 }
01836
01837 //-----------------------------------------------------------------------
01838 AVR_Opcode AVR_Opcode_Function( uint16_t OP_ )
01839 {
01840     switch (OP_)
01841     {
01842     case 0x0000: return AVR_Opcode_NOP;
01843
01844     case 0x9409: return AVR_Opcode_IJMP;
01845     case 0x9419: return AVR_Opcode_EIJMP;
01846
01847     case 0x9508: return AVR_Opcode_RET;
01848     case 0x9509: return AVR_Opcode_ICALL;
01849     case 0x9518: return AVR_Opcode_RETI;
01850     case 0x9519: return AVR_Opcode_EICALL;
01851     case 0x9588: return AVR_Opcode_SLEEP;
01852     case 0x9598: return AVR_Opcode_BREAK;
01853     case 0x95A8: return AVR_Opcode_WDR;
01854     case 0x95C8: return AVR_Opcode_LPM;
01855     case 0x95D8: return AVR_Opcode_ELPM;
01856     case 0x95E8: return AVR_Opcode_SPM;
01857     case 0x95F8: return AVR_Opcode_SPM_Z_Postinc2;
01858     }
01859
01860     switch( OP_ & 0xFF8F)
01861     {
01862     case 0x9408: return AVR_Opcode_BSET;
01863     case 0x9488: return AVR_Opcode_BCLR;
01864     }
01865
01866     switch (OP_ & 0xFF88)
01867     {
01868     case 0x0300: return AVR_Opcode_MULSU;
01869     case 0x0308: return AVR_Opcode_FMUL;
01870     case 0x0380: return AVR_Opcode_FMULS;
01871     case 0x0388: return AVR_Opcode_FMULSU;
01872     }
01873
01874     switch (OP_ & 0xFF0F)
01875     {
01876     case 0x940B: return AVR_Opcode_DES;
01877     case 0xEF0F: return AVR_Opcode_SER;
01878     }
01879
01880     switch (OP_ & 0xFF00)
01881     {
01882     case 0x0100: return AVR_Opcode_MOVW;
01883     case 0x9600: return AVR_Opcode_ADIW;
01884     case 0x9700: return AVR_Opcode_SBIW;
01885
01886     case 0x9800: return AVR_Opcode_CBI;
01887     case 0x9900: return AVR_Opcode_SBIC;
01888     case 0x9A00: return AVR_Opcode_SBI;
01889     case 0x9B00: return AVR_Opcode_SBIS;
01890     }
01891
01892     switch (OP_ & 0xFE0F)
01893     {
01894     case 0x8008: return AVR_Opcode_LD_Y_Indirect;
01895     case 0x8000: return AVR_Opcode_LD_Z_Indirect;
01896     case 0x8200: return AVR_Opcode_ST_Z_Indirect;
01897     case 0x8208: return AVR_Opcode_ST_Y_Indirect;
01898
01899     // -- Single 5-bit register...
01900     case 0x9000: return AVR_Opcode_LDS;
01901     case 0x9001: return AVR_Opcode_LD_Z_Indirect_Postinc;
01902     case 0x9002: return AVR_Opcode_LD_Z_Indirect_Predec;
01903     case 0x9004: return AVR_Opcode_LPM_Z;
01904     case 0x9005: return AVR_Opcode_LPM_Z_Postinc;
01905     case 0x9006: return AVR_Opcode_ELPM_Z;
01906     case 0x9007: return AVR_Opcode_ELPM_Z_Postinc;
01907     case 0x9009: return AVR_Opcode_LD_Y_Indirect_Postinc;
01908     case 0x900A: return AVR_Opcode_LD_Y_Indirect_Predec;
01909     case 0x900C: return AVR_Opcode_LD_X_Indirect;
01910     case 0x900D: return AVR_Opcode_LD_X_Indirect_Postinc;
01911     case 0x900E: return AVR_Opcode_LD_X_Indirect_Predec;
01912     case 0x900F: return AVR_Opcode_POP;
01913
```

```
01914     case 0x9200: return AVR_Opcode_STS;
01915     case 0x9201: return AVR_Opcode_ST_Z_Indirect_Postinc;
01916     case 0x9202: return AVR_Opcode_ST_Z_Indirect_Predec;
01917     case 0x9204: return AVR_Opcode_XCH;
01918     case 0x9205: return AVR_Opcode_LAS;
01919     case 0x9206: return AVR_Opcode_LAC;
01920     case 0x9207: return AVR_Opcode_LAT;
01921     case 0x9209: return AVR_Opcode_ST_Y_Indirect_Postinc;
01922     case 0x920A: return AVR_Opcode_ST_Y_Indirect_Predec;
01923     case 0x920C: return AVR_Opcode_ST_X_Indirect;
01924     case 0x920D: return AVR_Opcode_ST_X_Indirect_Postinc;
01925     case 0x920E: return AVR_Opcode_ST_X_Indirect_Predec;
01926     case 0x920F: return AVR_Opcode_PUSH;
01927
01928     // -- One-operand instructions
01929     case 0x9400: return AVR_Opcode_COM;
01930     case 0x9401: return AVR_Opcode_NEG;
01931     case 0x9402: return AVR_Opcode_SWAP;
01932     case 0x9403: return AVR_Opcode_INC;
01933     case 0x9405: return AVR_Opcode_ASR;
01934     case 0x9406: return AVR_Opcode_LSR;
01935     case 0x9407: return AVR_Opcode_ROR;
01936     case 0x940A: return AVR_Opcode_DEC;
01937
01938     }
01939     switch (OP_ & 0xFE0E)
01940     {
01941     case 0x940C: return AVR_Opcode_JMP;
01942     case 0x940E: return AVR_Opcode_CALL;
01943     }
01944
01945     switch (OP_ & 0xFE08)
01946     {
01947
01948     // -- BLD/BST Encoding
01949     case 0xF800: return AVR_Opcode_BLD;
01950     case 0xFA00: return AVR_Opcode_BST;
01951     // -- SBRC/SBRS Encoding
01952     case 0xFC00: return AVR_Opcode_SBRC;
01953     case 0xFE00: return AVR_Opcode_SBRS;
01954     }
01955
01956     switch (OP_ & 0xFC07)
01957     {
01958     // -- Conditional branches
01959     case 0xF000: return AVR_Opcode_BRCS;
01960     // case 0xF000: return AVR_Opcode_BRLO;            // AKA AVR_Opcode_BRCS;
01961     case 0xF001: return AVR_Opcode_BREQ;
01962     case 0xF002: return AVR_Opcode_BRMI;
01963     case 0xF003: return AVR_Opcode_BRVS;
01964     case 0xF004: return AVR_Opcode_BRLT;
01965     case 0xF006: return AVR_Opcode_BRTS;
01966     case 0xF007: return AVR_Opcode_BRIE;
01967     case 0xF400: return AVR_Opcode_BRCC;
01968     // case 0xF400: return AVR_Opcode_BRSH;            // AKA AVR_Opcode_BRCC;
01969     case 0xF401: return AVR_Opcode_BRNE;
01970     case 0xF402: return AVR_Opcode_BRPL;
01971     case 0xF403: return AVR_Opcode_BRVC;
01972     case 0xF404: return AVR_Opcode_BRGE;
01973     case 0xF405: return AVR_Opcode_BRHC;
01974     case 0xF406: return AVR_Opcode_BRTC;
01975     case 0xF407: return AVR_Opcode_BRID;
01976     }
01977
01978     switch (OP_ & 0xFC00)
01979     {
01980     // -- 4-bit register pair
01981     case 0x0200: return AVR_Opcode_MULS;
01982
01983     // -- 5-bit register pairs --
01984     case 0x0400: return AVR_Opcode_CPC;
01985     case 0x0800: return AVR_Opcode_SBC;
01986     case 0x0C00: return AVR_Opcode_ADD;
01987     // case 0x0C00: return AVR_Opcode_LSL; (!! Implemented with: " add rd, rd"
01988     case 0x1000: return AVR_Opcode_CPSE;
01989     case 0x1300: return AVR_Opcode_ROL;
01990     case 0x1400: return AVR_Opcode_CP;
01991     case 0x1C00: return AVR_Opcode_ADC;
01992     case 0x1800: return AVR_Opcode_SUB;
01993     case 0x2000: return AVR_Opcode_AND;
01994     // case 0x2000: return AVR_Opcode_TST; (!! Implemented with: " and rd, rd"
01995     case 0x2400: return AVR_Opcode_EOR;
01996     case 0x2C00: return AVR_Opcode_MOV;
01997     case 0x2800: return AVR_Opcode_OR;
01998
01999     // -- 5-bit register pairs -- Destination = R1:R0
02000     case 0x9C00: return AVR_Opcode_MUL;
```

```
02001     }
02002
02003     switch (OP_ & 0xF800)
02004     {
02005     case  0xB800: return AVR_Opcode_OUT;
02006     case  0xB000: return AVR_Opcode_IN;
02007     }
02008
02009     switch (OP_ & 0xF000)
02010     {
02011     // -- Register immediate --
02012     case 0x3000: return AVR_Opcode_CPI;
02013     case 0x4000: return AVR_Opcode_SBCI;
02014     case 0x5000: return AVR_Opcode_SUBI;
02015     case 0x6000: return AVR_Opcode_ORI;// return AVR_Opcode_SBR;
02016     case 0x7000: return AVR_Opcode_ANDI;
02017
02018     //-- 12-bit immediate
02019     case 0xC000: return AVR_Opcode_RJMP;
02020     case 0xD000: return AVR_Opcode_RCALL;
02021
02022     // -- Register immediate
02023     case 0xE000: return AVR_Opcode_LDI;
02024     }
02025
02026     switch (OP_ & 0xD208)
02027     {
02028     // -- 7-bit signed offset
02029     case 0x8000: return AVR_Opcode_LDD_Z;
02030     case 0x8008: return AVR_Opcode_LDD_Y;
02031     case 0x8200: return AVR_Opcode_STD_Z;
02032     case 0x8208: return AVR_Opcode_STD_Y;
02033     }
02034
02035     return AVR_Opcode_NOP;
02036 }
02037
02038 //----------------------------------------------------------------------------
02039 void AVR_RunOpcode(  uint16_t OP_ )
02040 {
02041     AVR_Opcode myOpcode = AVR_Opcode_Function( OP_);
02042     myOpcode();
02043 }
```

4.41 avr_opcodes.h File Reference

AVR CPU - Opcode interface.

```
#include <stdint.h>
#include "avr_cpu.h"
```

Typedefs

- typedef void(∗ **AVR_Opcode**)()

Functions

- AVR_Opcode AVR_Opcode_Function (uint16_t OP_)

  AVR_Opcode_Function.
- void AVR_RunOpcode (uint16_t OP_)

  AVR_RunOpcode.

4.41.1 Detailed Description

AVR CPU - Opcode interface.

Definition in file avr_opcodes.h.

4.41.2 Function Documentation

4.41.2.1 AVR_Opcode AVR_Opcode_Function (uint16_t *OP_*)

AVR_Opcode_Function.

Return a function pointer corresponding to the CPU logic for a given opcode.

Parameters

*OP_*	Opcode to return an "opcode execution" function pointer for

Returns

Opcode execution function pointer corresponding to the given opcode.

Definition at line 1838 of file avr_opcodes.c.

4.41.2.2 void AVR_RunOpcode (uint16_t *OP_*)

AVR_RunOpcode.

Execute the instruction corresponding to the provided opcode, on the provided CPU object. Note that the opcode must have just been decoded on the given CPU object before calling this function.

Parameters

*OP_*	Opcode to execute

Definition at line 2039 of file avr_opcodes.c.

4.42 avr_opcodes.h

```
00001 /*****************************************************************************
00002  *      (     (                       (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\      (    (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ----------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] ------------------
00008  *   | __| | |__    / _ \    \ V /  |   /    | -- [ Runtime ] ------------------
00009  *   |_|   |____| /_/ \_\    \_/   |_|_\    |
00010  *                                          | "Yeah, it does Arduino..."
00011  * -----------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #ifndef __AVR_OPCODES_H__
00022 #define __AVR_OPCODES_H__
00023
00024 #include <stdint.h>
00025 #include "avr_cpu.h"
00026
00027 //-----------------------------------------------------------------------------
00028 // Format opcode function jump table
00029 typedef void (*AVR_Opcode)();
00030
00031 //-----------------------------------------------------------------------------
00041 AVR_Opcode AVR_Opcode_Function( uint16_t OP_ );
00042
00043 //-----------------------------------------------------------------------------
00053 void AVR_RunOpcode(  uint16_t OP_ );
00054
00055 #endif
```

4.43 avr_peripheral.h File Reference

Interfaces for creating AVR peripheral plugins.

```
#include <stdint.h>
```

Data Structures

- struct AVRPeripheral

Typedefs

- typedef void(* **PeriphInit**)(void *context_)
- typedef void(* **PeriphRead**)(void *context_, uint8_t ucAddr_, uint8_t *pucValue_)
- typedef void(* **PeriphWrite**)(void *context_, uint8_t ucAddr_, uint8_t ucValue_)
- typedef void(* **PeriphClock**)(void *context_)
- typedef void(* **InterruptAck**)(uint8_t ucVector_)
- typedef struct AVRPeripheral **AVRPeripheral**

4.43.1 Detailed Description

Interfaces for creating AVR peripheral plugins.

Definition in file avr_peripheral.h.

4.44 avr_peripheral.h

```
00001 /*****************************************************************************
00002  *      (     (                      (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\      (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                         | "Yeah, it does Arduino..."
00011  * ----------------------------------------+-------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #ifndef __AVR_PERIPHERAL_H__
00022 #define __AVR_PERIPHERAL_H__
00023
00024 #include <stdint.h>
00025
00026 //-----------------------------------------------------------------------------
00027 // Peripheral callout functions - used to implement arbitrary peripherals
00028 // which are able to intercept/react to read/write operations to specific
00029 // I/O addresses.
00030 //-----------------------------------------------------------------------------
00031
00032 typedef void (*PeriphInit) (void *context_ );
00033 typedef void (*PeriphRead) (void *context_, uint8_t ucAddr_, uint8_t *pucValue_ );
00034 typedef void (*PeriphWrite)(void *context_, uint8_t ucAddr_, uint8_t ucValue_ );
00035 typedef void (*PeriphClock)(void *context_ );
00036
00037 //-----------------------------------------------------------------------------
00038 typedef void (*InterruptAck)( uint8_t ucVector_);
00039
00040 //-----------------------------------------------------------------------------
00041 typedef struct AVRPeripheral
00042 {
00043     PeriphInit          pfInit;
00044     PeriphRead          pfRead;
00045     PeriphWrite         pfWrite;
00046     PeriphClock         pfClock;
00047
00048     void                *pvContext;
00049
00050     uint8_t             u8AddrStart;
00051     uint8_t             u8AddrEnd;
```

```
00052 } AVRPeripheral;
00053
00054 #endif //__AVR_PERIPHERAL_H__
```

4.45 avr_periphregs.h File Reference

Module defining bitfield/register definitions for memory-mapped peripherals located within IO memory space.

```
#include <stdint.h>
```

Functions

- struct **__attribute__** ((__packed__))

Variables

- **AVR_UCSR0A**
- **AVR_UCSR0B**
- **AVR_UCSR0C**
- **AVR_TWAMR**
- **AVR_TWCR**
- **AVR_TWAR**
- **AVR_TWSR**
- **AVR_ASSR**
- **AVR_TCCR2B**
- **AVR_TCCR2A**
- **AVR_TCCR1A**
- **AVR_TCCR1B**
- **AVR_TCCR1C**
- **AVR_DIDR1**
- **AVR_DIDR0**
- **AVR_ADMUX**
- **AVR_ADCSRA**
- **AVR_ADCSRB**
- **AVR_TIMSK2**
- **AVR_TIMSK1**
- **AVR_TIMSK0**
- **AVR_PCMSK2**
- **AVR_PCMSK1**
- **AVR_PCMSK0**
- **AVR_PCICR**
- **AVR_EICRA**
- **AVR_PRR**
- **AVR_CLKPR**
- **AVR_WDTCSR**
- **AVR_SREG**
- **AVR_SPL**
- **AVR_SPH**
- **AVR_SPMCSR**
- **AVR_MCUCR**
- **AVR_MCUSR**
- **AVR_SMCR**
- **AVR_ACSR**

- **AVR_SPCR**
- **AVR_SPSR**
- **AVR_GTCCR**
- **AVR_TCCR0A**
- **AVR_TCCR0B**
- **AVR_EECR**
- **AVR_EIFR**
- **AVR_EIMSK**
- **AVR_PIN**
- **AVR_DDR**
- **AVR_PORT**
- **AVR_TIFR0**
- **AVR_TIFR1**
- **AVR_TIFR2**
- **AVR_PCIFR**

4.45.1 Detailed Description

Module defining bitfield/register definitions for memory-mapped peripherals located within IO memory space.

Definition in file avr_periphregs.h.

4.46 avr_periphregs.h

```
00001 /**************************************************************************
00002  *      (     (                        (     |
00003  *    )\ )  )\ )     (              )\ )    |
00004  *   (()/( (()/(    )\     (   (  (()/(    | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                         | "Yeah, it does Arduino..."
00011  * ----------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *************************************************************************/
00022 #ifndef __AVR_PERIPHREGS_H__
00023 #define __AVR_PERIPHREGS_H__
00024
00025 #include <stdint.h>
00026
00027 //-------------------------------------------------------------------------
00028 // UART/USART register struct definitions.
00029 //-------------------------------------------------------------------------
00030 typedef struct  __attribute__ ((__packed__))
00031 {
00032     union  __attribute__ ((__packed__))
00033     {
00034         uint8_t r;
00035         struct  __attribute__ ((__packed__))
00036         {
00037             unsigned int MPCM0: 1;
00038             unsigned int U2X0 : 1;
00039             unsigned int UPE0 : 1;
00040             unsigned int DOR0 : 1;
00041             unsigned int FE0  : 1;
00042             unsigned int UDRE0 : 1;
00043             unsigned int TXC0 : 1;
00044             unsigned int RXC0 : 1;
00045         };
00046     };
00047 } AVR_UCSR0A;
00048
00049 //-------------------------------------------------------------------------
00050 typedef struct  __attribute__ ((__packed__))
00051 {
00052     union  __attribute__ ((__packed__))
00053     {
00054         uint8_t r;
```

```
00055         struct  __attribute__ ((__packed__))
00056         {
00057             unsigned int TXB80  : 1;
00058             unsigned int RXB80  : 1;
00059             unsigned int UCSZ02 : 1;
00060             unsigned int TXEN0  : 1;
00061             unsigned int RXEN0  : 1;
00062             unsigned int UDRIE0 : 1;
00063             unsigned int TXCIE0 : 1;
00064             unsigned int RXCIE0 : 1;
00065         };
00066     };
00067 } AVR_UCSR0B;
00068
00069 //------------------------------------------------------------------------------
00070 typedef struct  __attribute__ ((__packed__))
00071 {
00072     union  __attribute__ ((__packed__))
00073     {
00074         uint8_t r;
00075         struct  __attribute__ ((__packed__))
00076         {
00077             unsigned int UCPOL0  : 1;
00078             unsigned int UCPHA0  : 1;
00079             unsigned int UDORD0  : 1;
00080             unsigned int USBS0   : 1;
00081             unsigned int UPM00   : 1;
00082             unsigned int UPM01   : 1;
00083             unsigned int UMSEL00 : 1;
00084             unsigned int UMSEL01 : 1;
00085         };
00086     };
00087 } AVR_UCSR0C;
00088
00089 //------------------------------------------------------------------------------
00090 // TWI interface register struct definitions
00091 //------------------------------------------------------------------------------
00092 typedef struct  __attribute__ ((__packed__))
00093 {
00094     union  __attribute__ ((__packed__))
00095     {
00096         uint8_t r;
00097         struct  __attribute__ ((__packed__))
00098         {
00099             unsigned int reserved : 1;
00100             unsigned int TWAM0 : 1;
00101             unsigned int TWAM1 : 1;
00102             unsigned int TWAM2 : 1;
00103             unsigned int TWAM3 : 1;
00104             unsigned int TWAM4 : 1;
00105             unsigned int TWAM5 : 1;
00106             unsigned int TWAM6 : 1;
00107         };
00108     };
00109 } AVR_TWAMR;
00110
00111 //------------------------------------------------------------------------------
00112 typedef struct  __attribute__ ((__packed__))
00113 {
00114     union  __attribute__ ((__packed__))
00115     {
00116         uint8_t r;
00117         struct  __attribute__ ((__packed__))
00118         {
00119             unsigned int TWIE  : 1;
00120             unsigned int reserved : 1;
00121             unsigned int TWEN  : 1;
00122             unsigned int TWWC  : 1;
00123             unsigned int TWSTO : 1;
00124             unsigned int TWSTA : 1;
00125             unsigned int TWEA  : 1;
00126             unsigned int TWINT : 1;
00127         };
00128     };
00129 } AVR_TWCR;
00130
00131 //------------------------------------------------------------------------------
00132 typedef struct  __attribute__ ((__packed__))
00133 {
00134     union  __attribute__ ((__packed__))
00135     {
00136         uint8_t r;
00137         struct  __attribute__ ((__packed__))
00138         {
00139             unsigned int TWGCE: 1;
00140             unsigned int TWA0 : 1;
00141             unsigned int TWA1 : 1;
```

```
00142             unsigned int TWA2 : 1;
00143             unsigned int TWA3 : 1;
00144             unsigned int TWA4 : 1;
00145             unsigned int TWA5 : 1;
00146             unsigned int TWA6 : 1;
00147         };
00148     };
00149 } AVR_TWAR;
00150
00151 //------------------------------------------------------------------------------
00152 typedef struct  __attribute__ ((__packed__))
00153 {
00154     union  __attribute__ ((__packed__))
00155     {
00156         uint8_t r;
00157         struct  __attribute__ ((__packed__))
00158         {
00159             unsigned int TWPS0 : 1;
00160             unsigned int TWPS1 : 1;
00161             unsigned int reserved : 1;
00162             unsigned int TWPS3 : 1;
00163             unsigned int TWPS4 : 1;
00164             unsigned int TWPS5 : 1;
00165             unsigned int TWPS6 : 1;
00166             unsigned int TWPS7 : 1;
00167         };
00168     };
00169 } AVR_TWSR;
00170
00171 //------------------------------------------------------------------------------
00172 // Timer 2 register struct  __attribute__ ((__packed__)) definitins.
00173 //------------------------------------------------------------------------------
00174 typedef struct  __attribute__ ((__packed__))
00175 {
00176     union  __attribute__ ((__packed__))
00177     {
00178         uint8_t r;
00179         struct  __attribute__ ((__packed__))
00180         {
00181             unsigned int TCR2BUB : 1;
00182             unsigned int TCR2AUB : 1;
00183             unsigned int OCR2BUB : 1;
00184             unsigned int OCR2AUB : 1;
00185             unsigned int TCN2UB  : 1;
00186             unsigned int AS2     : 1;
00187             unsigned int EXCLK   : 1;
00188             unsigned int reserved : 1;
00189         };
00190     };
00191 } AVR_ASSR;
00192
00193 //------------------------------------------------------------------------------
00194 typedef struct  __attribute__ ((__packed__))
00195 {
00196     union  __attribute__ ((__packed__))
00197     {
00198         uint8_t r;
00199         struct  __attribute__ ((__packed__))
00200         {
00201             unsigned int CS20  : 1;
00202             unsigned int CS21  : 1;
00203             unsigned int CS22  : 1;
00204             unsigned int WGM22 : 1;
00205             unsigned int reserved : 2;
00206             unsigned int FOC2B : 1;
00207             unsigned int FOC2A : 1;
00208         };
00209     };
00210 } AVR_TCCR2B;
00211
00212 //------------------------------------------------------------------------------
00213 typedef struct  __attribute__ ((__packed__))
00214 {
00215     union  __attribute__ ((__packed__))
00216     {
00217         uint8_t r;
00218         struct  __attribute__ ((__packed__))
00219         {
00220             unsigned int WGM20 : 1;
00221             unsigned int WGM21 : 1;
00222             unsigned int reserved : 2;
00223             unsigned int COM2B0 : 1;
00224             unsigned int COM2B1 : 1;
00225             unsigned int COM2A0 : 1;
00226             unsigned int COM2A1 : 1;
00227         };
00228     };
```

```
00229 } AVR_TCCR2A;
00230
00231 //------------------------------------------------------------------------
00232 // Timer 1 Register struct  __attribute__ ((__packed__)) definitions
00233 //------------------------------------------------------------------------
00234
00235 typedef struct  __attribute__ ((__packed__))
00236 {
00237     union  __attribute__ ((__packed__))
00238     {
00239         uint8_t r;
00240         struct  __attribute__ ((__packed__))
00241         {
00242             unsigned int WGM10 : 1;
00243             unsigned int WGM11 : 1;
00244             unsigned int reserved : 2;
00245             unsigned int COM1B0 : 1;
00246             unsigned int COM1B1 : 1;
00247             unsigned int COM1A0 : 1;
00248             unsigned int COM1A1 : 1;
00249         };
00250     };
00251 } AVR_TCCR1A;
00252
00253 //------------------------------------------------------------------------
00254 typedef struct  __attribute__ ((__packed__))
00255 {
00256     union  __attribute__ ((__packed__))
00257     {
00258         uint8_t r;
00259         struct  __attribute__ ((__packed__))
00260         {
00261             unsigned int CS10  : 1;
00262             unsigned int CS11  : 1;
00263             unsigned int CS12  : 1;
00264             unsigned int WGM12 : 1;
00265             unsigned int WGM13 : 1;
00266             unsigned int reserved : 1;
00267             unsigned int ICES1 : 1;
00268             unsigned int ICNC1 : 1;
00269         };
00270     };
00271 } AVR_TCCR1B;
00272
00273 //------------------------------------------------------------------------
00274 typedef struct  __attribute__ ((__packed__))
00275 {
00276     union  __attribute__ ((__packed__))
00277     {
00278         uint8_t r;
00279         struct  __attribute__ ((__packed__))
00280         {
00281             unsigned int reserved : 6;
00282             unsigned int FOC1B : 1;
00283             unsigned int FOC1A : 1;
00284         };
00285     };
00286 } AVR_TCCR1C;
00287
00288 //------------------------------------------------------------------------
00289 // A2D converter register definitions
00290 //------------------------------------------------------------------------
00291 typedef struct  __attribute__ ((__packed__))
00292 {
00293     union  __attribute__ ((__packed__))
00294     {
00295         uint8_t r;
00296         struct  __attribute__ ((__packed__))
00297         {
00298             unsigned int AIN0D : 1;
00299             unsigned int AIN1D : 1;
00300             unsigned int reserved : 6;
00301         };
00302     };
00303 } AVR_DIDR1;
00304
00305 //------------------------------------------------------------------------
00306 typedef struct  __attribute__ ((__packed__))
00307 {
00308     union  __attribute__ ((__packed__))
00309     {
00310         uint8_t r;
00311         struct  __attribute__ ((__packed__))
00312         {
00313             unsigned int ADC0D : 1;
00314             unsigned int ADC1D : 1;
00315             unsigned int ADC2D : 1;
```

```
00316             unsigned int ADC3D : 1;
00317             unsigned int ADC4D : 1;
00318             unsigned int ADC5D : 1;
00319             unsigned int reserved : 2;
00320         };
00321     };
00322 } AVR_DIDR0;
00323
00324 //------------------------------------------------------------------------
00325 typedef struct  __attribute__ ((__packed__))
00326 {
00327     union  __attribute__ ((__packed__))
00328     {
00329         uint8_t r;
00330         struct  __attribute__ ((__packed__))
00331         {
00332             unsigned int MUX0     : 1;
00333             unsigned int MUX1     : 1;
00334             unsigned int MUX2     : 1;
00335             unsigned int MUX3     : 1;
00336             unsigned int reserved : 1;
00337             unsigned int ADLAR    : 1;
00338             unsigned int REFS0    : 1;
00339             unsigned int REFS1    : 1;
00340         };
00341     };
00342 } AVR_ADMUX;
00343
00344 //------------------------------------------------------------------------
00345 typedef struct  __attribute__ ((__packed__))
00346 {
00347     union  __attribute__ ((__packed__))
00348     {
00349         uint8_t r;
00350         struct  __attribute__ ((__packed__))
00351         {
00352             unsigned int ADPS0 : 1;
00353             unsigned int ADPS1 : 1;
00354             unsigned int ADPS2 : 1;
00355             unsigned int ADIE  : 1;
00356             unsigned int ADIF  : 1;
00357             unsigned int ADATE : 1;
00358             unsigned int ADSC  : 1;
00359             unsigned int ADEN  : 1;
00360         };
00361     };
00362 } AVR_ADCSRA;
00363
00364 //------------------------------------------------------------------------
00365 typedef struct  __attribute__ ((__packed__))
00366 {
00367     union  __attribute__ ((__packed__))
00368     {
00369         uint8_t r;
00370         struct  __attribute__ ((__packed__))
00371         {
00372             unsigned int ADTS0     : 1;
00373             unsigned int ADTS1     : 1;
00374             unsigned int ADTS2     : 1;
00375             unsigned int reserved  : 3;
00376             unsigned int ACMD      : 1;
00377             unsigned int reserved_ : 1;
00378         };
00379     };
00380 } AVR_ADCSRB;
00381
00382 //------------------------------------------------------------------------
00383 // Timer interrupt mask registers.
00384 //------------------------------------------------------------------------
00385 typedef struct  __attribute__ ((__packed__))
00386 {
00387     union  __attribute__ ((__packed__))
00388     {
00389         uint8_t r;
00390         struct  __attribute__ ((__packed__))
00391         {
00392             unsigned int TOIE2    : 1;
00393             unsigned int OCIE2A   : 1;
00394             unsigned int OCIE2B   : 1;
00395             unsigned int reserved : 5;
00396         };
00397     };
00398 } AVR_TIMSK2;
00399
00400 //------------------------------------------------------------------------
00401 typedef struct  __attribute__ ((__packed__))
00402 {
```

```
00403     union  __attribute__ ((__packed__))
00404     {
00405         uint8_t r;
00406         struct  __attribute__ ((__packed__))
00407         {
00408             unsigned int TOIE1    : 1;
00409             unsigned int OCIE1A   : 1;
00410             unsigned int OCIE1B   : 1;
00411             unsigned int reserved : 2;
00412             unsigned int ICIE1    : 1;
00413             unsigned int reserved_ : 2;
00414         };
00415     };
00416 } AVR_TIMSK1;
00417
00418 //------------------------------------------------------------------------
00419 typedef struct  __attribute__ ((__packed__))
00420 {
00421     union  __attribute__ ((__packed__))
00422     {
00423         uint8_t r;
00424         struct  __attribute__ ((__packed__))
00425         {
00426             unsigned int TOIE0    : 1;
00427             unsigned int OCIE0A   : 1;
00428             unsigned int OCIE0B   : 1;
00429             unsigned int reserved : 5;
00430         };
00431     };
00432 } AVR_TIMSK0;
00433
00434 //------------------------------------------------------------------------
00435 // Pin change interrupt mask bit definitions
00436 //------------------------------------------------------------------------
00437 typedef struct  __attribute__ ((__packed__))
00438 {
00439     union  __attribute__ ((__packed__))
00440     {
00441         uint8_t r;
00442         struct  __attribute__ ((__packed__))
00443         {
00444             unsigned int PCINT16 : 1;
00445             unsigned int PCINT17 : 1;
00446             unsigned int PCINT18 : 1;
00447             unsigned int PCINT19 : 1;
00448             unsigned int PCINT20 : 1;
00449             unsigned int PCINT21 : 1;
00450             unsigned int PCINT22 : 1;
00451             unsigned int PCINT23 : 1;
00452         };
00453     };
00454 } AVR_PCMSK2;
00455
00456 //------------------------------------------------------------------------
00457 typedef struct  __attribute__ ((__packed__))
00458 {
00459     union  __attribute__ ((__packed__))
00460     {
00461         uint8_t r;
00462         struct  __attribute__ ((__packed__))
00463         {
00464             unsigned int PCINT8 : 1;
00465             unsigned int PCINT9 : 1;
00466             unsigned int PCINT10 : 1;
00467             unsigned int PCINT11 : 1;
00468             unsigned int PCINT12 : 1;
00469             unsigned int PCINT13 : 1;
00470             unsigned int PCINT14 : 1;
00471             unsigned int PCINT15 : 1;
00472         };
00473     };
00474 } AVR_PCMSK1;
00475
00476 //------------------------------------------------------------------------
00477 typedef struct  __attribute__ ((__packed__))
00478 {
00479     union  __attribute__ ((__packed__))
00480     {
00481         uint8_t r;
00482         struct  __attribute__ ((__packed__))
00483         {
00484             unsigned int PCINT0 : 1;
00485             unsigned int PCINT1 : 1;
00486             unsigned int PCINT2 : 1;
00487             unsigned int PCINT3 : 1;
00488             unsigned int PCINT4 : 1;
00489             unsigned int PCINT5 : 1;
```

```
00490             unsigned int PCINT6 : 1;
00491             unsigned int PCINT7 : 1;
00492         };
00493     };
00494 } AVR_PCMSK0;
00495
00496 //------------------------------------------------------------------------------
00497 typedef struct  __attribute__ ((__packed__))
00498 {
00499     union  __attribute__ ((__packed__))
00500     {
00501         uint8_t r;
00502         struct  __attribute__ ((__packed__))
00503         {
00504             unsigned int PCIE0 : 1;
00505             unsigned int PCIE1 : 1;
00506             unsigned int PCIE2 : 1;
00507             unsigned int reserved : 5;
00508         };
00509     };
00510 } AVR_PCICR;
00511
00512 //------------------------------------------------------------------------------
00513 typedef struct  __attribute__ ((__packed__))
00514 {
00515     union  __attribute__ ((__packed__))
00516     {
00517         uint8_t r;
00518         struct  __attribute__ ((__packed__))
00519         {
00520             unsigned int ISC00    : 1;
00521             unsigned int ISC01    : 1;
00522             unsigned int ISC10    : 1;
00523             unsigned int ISC11    : 1;
00524             unsigned int reserved : 4;
00525         };
00526     };
00527 } AVR_EICRA;
00528
00529 //------------------------------------------------------------------------------
00530 typedef struct  __attribute__ ((__packed__))
00531 {
00532     union  __attribute__ ((__packed__))
00533     {
00534         uint8_t r;
00535         struct  __attribute__ ((__packed__))
00536         {
00537             unsigned int PRADC : 1;
00538             unsigned int PRUSART0 : 1;
00539             unsigned int PRSPI: 1;
00540             unsigned int PRTIM1: 1;
00541             unsigned int reserved : 1;
00542             unsigned int PRTIM0 : 1;
00543             unsigned int PRTIM2 : 1;
00544             unsigned int PRTWI : 1;
00545         };
00546     };
00547 } AVR_PRR;
00548
00549 //------------------------------------------------------------------------------
00550 typedef struct  __attribute__ ((__packed__))
00551 {
00552     union  __attribute__ ((__packed__))
00553     {
00554         uint8_t r;
00555         struct  __attribute__ ((__packed__))
00556         {
00557             unsigned int CLKPS0 : 1;
00558             unsigned int CLKPS1 : 1;
00559             unsigned int CLKPS2 : 1;
00560             unsigned int CLKPS3 : 1;
00561             unsigned int reserved : 3;
00562             unsigned int CLKPCE : 1;
00563         };
00564     };
00565 } AVR_CLKPR;
00566
00567 //------------------------------------------------------------------------------
00568 typedef struct  __attribute__ ((__packed__))
00569 {
00570     union  __attribute__ ((__packed__))
00571     {
00572         uint8_t r;
00573         struct  __attribute__ ((__packed__))
00574         {
00575             unsigned int WDP0 : 1;
00576             unsigned int WDP1 : 1;
```

```
00577             unsigned int WDP2 : 1;
00578             unsigned int WDE  : 1;
00579             unsigned int WDCE : 1;
00580             unsigned int WDP3 : 1;
00581             unsigned int WDIE : 1;
00582             unsigned int WDIF : 1;
00583         };
00584     };
00585 } AVR_WDTCSR;
00586
00587 //------------------------------------------------------------------------------
00588 typedef struct  __attribute__ ((__packed__))
00589 {
00590     union  __attribute__ ((__packed__))
00591     {
00592         uint8_t r;
00593         struct  __attribute__ ((__packed__))
00594         {
00595             unsigned int C : 1;
00596             unsigned int Z : 1;
00597             unsigned int N : 1;
00598             unsigned int V : 1;
00599             unsigned int S : 1;
00600             unsigned int H : 1;
00601             unsigned int T : 1;
00602             unsigned int I : 1;
00603         };
00604     };
00605 } AVR_SREG;
00606
00607 //------------------------------------------------------------------------------
00608 typedef struct  __attribute__ ((__packed__))
00609 {
00610     union  __attribute__ ((__packed__))
00611     {
00612         uint8_t r;
00613         struct  __attribute__ ((__packed__))
00614         {
00615             unsigned int SP0 : 1;
00616             unsigned int SP1 : 1;
00617             unsigned int SP2 : 1;
00618             unsigned int SP3 : 1;
00619             unsigned int SP4 : 1;
00620             unsigned int SP5 : 1;
00621             unsigned int SP6 : 1;
00622             unsigned int SP7 : 1;
00623         };
00624     };
00625 } AVR_SPL;
00626
00627 //------------------------------------------------------------------------------
00628 typedef struct  __attribute__ ((__packed__))
00629 {
00630     union  __attribute__ ((__packed__))
00631     {
00632         uint8_t r;
00633         struct  __attribute__ ((__packed__))
00634         {
00635             unsigned int SP8      : 1;
00636             unsigned int SP9      : 1;
00637             unsigned int SP10     : 1;
00638             unsigned int reserved : 5;
00639         };
00640     };
00641 } AVR_SPH;
00642
00643 //------------------------------------------------------------------------------
00644 typedef struct  __attribute__ ((__packed__))
00645 {
00646     union  __attribute__ ((__packed__))
00647     {
00648         uint8_t r;
00649         struct  __attribute__ ((__packed__))
00650         {
00651             unsigned int SELFPRGEN : 1;
00652             unsigned int PGERS     : 1;
00653             unsigned int PGWRT     : 1;
00654             unsigned int BLBSET    : 1;
00655             unsigned int RWWSRE    : 1;
00656             unsigned int RWWSB     : 1;
00657             unsigned int SPMIE     : 1;
00658         };
00659     };
00660 } AVR_SPMCSR;
00661
00662 //------------------------------------------------------------------------------
00663 typedef struct  __attribute__ ((__packed__))
```

```
00664 {
00665     union  __attribute__ ((__packed__))
00666     {
00667         uint8_t r;
00668         struct  __attribute__ ((__packed__))
00669         {
00670             unsigned int IVCE      : 1;
00671             unsigned int IVSEL     : 1;
00672             unsigned int reserved  : 2;
00673             unsigned int PUD       : 1;
00674             unsigned int BODSE     : 1;
00675             unsigned int BODS      : 1;
00676             unsigned int reserved_ : 1;
00677         };
00678     };
00679 } AVR_MCUCR;
00680
00681 //------------------------------------------------------------------------------
00682 typedef struct  __attribute__ ((__packed__))
00683 {
00684     union  __attribute__ ((__packed__))
00685     {
00686         uint8_t r;
00687         struct  __attribute__ ((__packed__))
00688         {
00689             unsigned int PORF     : 1;
00690             unsigned int EXTRF    : 1;
00691             unsigned int BORF     : 1;
00692             unsigned int WDRF     : 1;
00693             unsigned int reserved : 4;
00694         };
00695     };
00696 } AVR_MCUSR;
00697
00698 //------------------------------------------------------------------------------
00699 typedef struct  __attribute__ ((__packed__))
00700 {
00701     union  __attribute__ ((__packed__))
00702     {
00703         uint8_t r;
00704         struct  __attribute__ ((__packed__))
00705         {
00706             unsigned int SE       : 1;
00707             unsigned int SM0      : 1;
00708             unsigned int SM1      : 1;
00709             unsigned int SM2      : 1;
00710             unsigned int reserved : 4;
00711         };
00712     };
00713 } AVR_SMCR;
00714
00715 //------------------------------------------------------------------------------
00716 typedef struct  __attribute__ ((__packed__))
00717 {
00718     union  __attribute__ ((__packed__))
00719     {
00720         uint8_t r;
00721         struct  __attribute__ ((__packed__))
00722         {
00723             unsigned int ACIS0 : 1;
00724             unsigned int ACIS1 : 1;
00725             unsigned int ACIC  : 1;
00726             unsigned int ACIE  : 1;
00727             unsigned int ACI   : 1;
00728             unsigned int AC0   : 1;
00729             unsigned int ACBG  : 1;
00730             unsigned int ACD   : 1;
00731         };
00732     };
00733 } AVR_ACSR;
00734
00735 //------------------------------------------------------------------------------
00736 typedef struct  __attribute__ ((__packed__))
00737 {
00738     union  __attribute__ ((__packed__))
00739     {
00740         uint8_t r;
00741         struct  __attribute__ ((__packed__))
00742         {
00743             unsigned int SPR0 : 1;
00744             unsigned int SPR1 : 1;
00745             unsigned int CPHA : 1;
00746             unsigned int CPOL : 1;
00747             unsigned int MSTR : 1;
00748             unsigned int DORD : 1;
00749             unsigned int SPE  : 1;
00750             unsigned int SPIE : 1;
```

```
00751         };
00752     };
00753 } AVR_SPCR;
00754
00755 //------------------------------------------------------------------------------
00756 typedef struct  __attribute__ ((__packed__))
00757 {
00758     union  __attribute__ ((__packed__))
00759     {
00760         uint8_t r;
00761         struct  __attribute__ ((__packed__))
00762         {
00763             unsigned int SPI2X    : 1;
00764             unsigned int reserved : 5;
00765             unsigned int WCOL     : 1;
00766             unsigned int SPIF     : 1;
00767         };
00768     };
00769 } AVR_SPSR;
00770
00771 //------------------------------------------------------------------------------
00772 typedef struct  __attribute__ ((__packed__))
00773 {
00774     union  __attribute__ ((__packed__))
00775     {
00776         uint8_t r;
00777         struct  __attribute__ ((__packed__))
00778         {
00779             unsigned int PSRSYNC  : 1;
00780             unsigned int PSRASY   : 1;
00781             unsigned int reserved : 5;
00782             unsigned int TSM      : 1;
00783         };
00784     };
00785 } AVR_GTCCR;
00786
00787 //------------------------------------------------------------------------------
00788 typedef struct  __attribute__ ((__packed__))
00789 {
00790     union  __attribute__ ((__packed__))
00791     {
00792         uint8_t r;
00793         struct  __attribute__ ((__packed__))
00794         {
00795             unsigned int WGM00    : 1;
00796             unsigned int WGM01    : 1;
00797             unsigned int reserved : 2;
00798             unsigned int COM0B0   : 1;
00799             unsigned int COM0B1   : 1;
00800             unsigned int COM0A0   : 1;
00801             unsigned int COM0A1   : 1;
00802         };
00803     };
00804 } AVR_TCCR0A;
00805
00806 //------------------------------------------------------------------------------
00807 typedef struct  __attribute__ ((__packed__))
00808 {
00809     union  __attribute__ ((__packed__))
00810     {
00811         uint8_t r;
00812         struct  __attribute__ ((__packed__))
00813         {
00814             unsigned int CS00     : 1;
00815             unsigned int CS01     : 1;
00816             unsigned int CS02     : 1;
00817             unsigned int WGM02    : 1;
00818             unsigned int reserved : 2;
00819             unsigned int FOC0B    : 1;
00820             unsigned int FOC0A    : 1;
00821         };
00822     };
00823 } AVR_TCCR0B;
00824
00825 //------------------------------------------------------------------------------
00826 typedef struct  __attribute__ ((__packed__))
00827 {
00828     union  __attribute__ ((__packed__))
00829     {
00830         uint8_t r;
00831         struct  __attribute__ ((__packed__))
00832         {
00833             unsigned int EERE     : 1;
00834             unsigned int EEPE     : 1;
00835             unsigned int EEMPE    : 1;
00836             unsigned int EERIE    : 1;
00837             unsigned int EEPM0    : 1;
```

```
00838             unsigned int EEPM1    : 1;
00839             unsigned int reserved : 2;
00840         };
00841     };
00842 } AVR_EECR;
00843
00844 //---------------------------------------------------------------------------
00845 // External interrupt flag register definitions
00846 //---------------------------------------------------------------------------
00847 typedef struct  __attribute__ ((__packed__))
00848 {
00849     union  __attribute__ ((__packed__))
00850     {
00851         uint8_t r;
00852         struct  __attribute__ ((__packed__))
00853         {
00854             unsigned int INTF0    : 1;
00855             unsigned int INTF1    : 1;
00856             unsigned int reserved : 6;
00857         };
00858     };
00859 } AVR_EIFR;
00860
00861 //---------------------------------------------------------------------------
00862 // External interrupt mask register definitions
00863 //---------------------------------------------------------------------------
00864 typedef struct  __attribute__ ((__packed__))
00865 {
00866     union  __attribute__ ((__packed__))
00867     {
00868         uint8_t r;
00869         struct  __attribute__ ((__packed__))
00870         {
00871             unsigned int INT0    : 1;
00872             unsigned int INT1    : 1;
00873             unsigned int reserved : 6;
00874         };
00875     };
00876 } AVR_EIMSK;
00877
00878 //---------------------------------------------------------------------------
00879 // Pin (GPIO) register definitions
00880 //---------------------------------------------------------------------------
00881 typedef struct  __attribute__ ((__packed__))
00882 {
00883     union  __attribute__ ((__packed__))
00884     {
00885         uint8_t r;
00886         struct  __attribute__ ((__packed__))
00887         {
00888             unsigned int PIN0 : 1;
00889             unsigned int PIN1 : 1;
00890             unsigned int PIN2 : 1;
00891             unsigned int PIN3 : 1;
00892             unsigned int PIN4 : 1;
00893             unsigned int PIN5 : 1;
00894             unsigned int PIN6 : 1;
00895             unsigned int PIN7 : 1;
00896         };
00897     };
00898 } AVR_PIN;
00899
00900 //---------------------------------------------------------------------------
00901 // Data-direction register (GPIO) definitions
00902 //---------------------------------------------------------------------------
00903 typedef struct  __attribute__ ((__packed__))
00904 {
00905     union  __attribute__ ((__packed__))
00906     {
00907         uint8_t r;
00908         struct  __attribute__ ((__packed__))
00909         {
00910             unsigned int DDR0 : 1;
00911             unsigned int DDR1 : 1;
00912             unsigned int DDR2 : 1;
00913             unsigned int DDR3 : 1;
00914             unsigned int DDR4 : 1;
00915             unsigned int DDR5 : 1;
00916             unsigned int DDR6 : 1;
00917             unsigned int DDR7 : 1;
00918         };
00919     };
00920 } AVR_DDR;
00921
00922 //---------------------------------------------------------------------------
00923 // Port (GPIO) register definitions
00924 //---------------------------------------------------------------------------
```

```
00925 typedef struct  __attribute__ ((__packed__))
00926 {
00927     union  __attribute__ ((__packed__))
00928     {
00929         uint8_t r;
00930         struct  __attribute__ ((__packed__))
00931         {
00932             unsigned int PORT0 : 1;
00933             unsigned int PORT1 : 1;
00934             unsigned int PORT2 : 1;
00935             unsigned int PORT3 : 1;
00936             unsigned int PORT4 : 1;
00937             unsigned int PORT5 : 1;
00938             unsigned int PORT6 : 1;
00939             unsigned int PORT7 : 1;
00940         };
00941     };
00942 } AVR_PORT;
00943
00944
00945 //--------------------------------------------------------------------------
00946 // Timer interrupt flag register struct  __attribute__ ((__packed__)) definitions
00947 //--------------------------------------------------------------------------
00948 typedef struct  __attribute__ ((__packed__))
00949 {
00950     union  __attribute__ ((__packed__))
00951     {
00952         uint8_t r;
00953         struct  __attribute__ ((__packed__))
00954         {
00955             unsigned int TOV0     : 1;
00956             unsigned int OCF0A    : 1;
00957             unsigned int OCF0B    : 1;
00958             unsigned int reserved : 5;
00959         };
00960     };
00961 } AVR_TIFR0;
00962
00963 //--------------------------------------------------------------------------
00964 typedef struct  __attribute__ ((__packed__))
00965 {
00966     union  __attribute__ ((__packed__))
00967     {
00968         uint8_t r;
00969         struct  __attribute__ ((__packed__))
00970         {
00971             unsigned int TOV1      : 1;
00972             unsigned int OCF1A     : 1;
00973             unsigned int OCF1B     : 1;
00974             unsigned int reserved  : 2;
00975             unsigned int ICF1      : 1;
00976             unsigned int reserved_ : 2;
00977         };
00978     };
00979 } AVR_TIFR1;
00980
00981 //--------------------------------------------------------------------------
00982 typedef struct  __attribute__ ((__packed__))
00983 {
00984     union  __attribute__ ((__packed__))
00985     {
00986         uint8_t r;
00987         struct  __attribute__ ((__packed__))
00988         {
00989             unsigned int TOV2     : 1;
00990             unsigned int OCF2A    : 1;
00991             unsigned int OCF2B    : 1;
00992             unsigned int reserved : 5;
00993         };
00994     };
00995 } AVR_TIFR2;
00996
00997 //--------------------------------------------------------------------------
00998 // Pin-change interrupt flag bits
00999 //--------------------------------------------------------------------------
01000 typedef struct  __attribute__ ((__packed__))
01001 {
01002     union  __attribute__ ((__packed__))
01003     {
01004         uint8_t r;
01005         struct  __attribute__ ((__packed__))
01006         {
01007             unsigned int PCIF0    : 1;
01008             unsigned int PCIF1    : 1;
01009             unsigned int PCIF2    : 1;
01010             unsigned int reserved : 5;
01011         };
```

```
01012     };
01013 } AVR_PCIFR;
01014
01015 #endif // __AVR_PERIPHREGS_H__
```

4.47 avr_registerfile.h File Reference

Module providing a mapping of IO memory to the AVR register file.

```
#include "avr_coreregs.h"
#include "avr_periphregs.h"
```

Data Structures

- struct AVRRegisterFile

  The first 256 bytes of the AVR memory space is composed of the core 32 general-purpse registers (R0-R31), and 224 bytes for the remaining I/O registers (used by peripherals).

4.47.1 Detailed Description

Module providing a mapping of IO memory to the AVR register file.

Definition in file avr_registerfile.h.

4.48 avr_registerfile.h

```
00001 /*****************************************************************************
00002  *      (      (                       (      |
00003  *    )\ )   )\ )     (              )\ )    |
00004  *   (()/( (()/(     )\      (    (  (()/(    | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\   )\   /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ----------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \     | -- [ Virtual ] ------------------
00008  *   | __| | |__    / _ \    \ V /  |   /    | -- [ Runtime ] ------------------
00009  *   |_|   |____| /_/ \_\    \_/   |_|_\    |
00010  *                                          | "Yeah, it does Arduino..."
00011  * -----------------------------------------+-------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #ifndef __AVR_REGISTERFILE_H__
00022 #define __AVR_REGISTERFILE_H__
00023
00024 //------------------------------------------------------------------------------
00025 #include "avr_coreregs.h"
00026 #include "avr_periphregs.h"
00027
00028 //------------------------------------------------------------------------------
00038 typedef struct
00039 {
00040     //-- 0x00
00041     AVR_CoreRegisters CORE_REGISTERS;
00042
00043     //-- 0x20
00044     AVR_PIN     PINA;
00045     AVR_DDR     DDRA;
00046     AVR_PORT    PORTA;
00047
00048     //-- 0x23
00049     AVR_PIN     PINB;
00050     AVR_DDR     DDRB;
00051     AVR_PORT    PORTB;
00052
00053     //-- 0x26
00054     AVR_PIN     PINC;
00055     AVR_DDR     DDRC;
00056     AVR_PORT    PORTC;
00057
```

```
00058     //-- 0x29
00059     AVR_PIN     PIND;
00060     AVR_DDR     DDRD;
00061     AVR_PORT    PORTD;
00062
00063     //-- 0x2C
00064     uint8_t     RESERVED_0x2C;
00065     uint8_t     RESERVED_0x2D;
00066     uint8_t     RESERVED_0x2E;
00067     uint8_t     RESERVED_0x2F;
00068     uint8_t     RESERVED_0x30;
00069     uint8_t     RESERVED_0x31;
00070     uint8_t     RESERVED_0x32;
00071     uint8_t     RESERVED_0x33;
00072     uint8_t     RESERVED_0x34;
00073
00074     //-- 0x35
00075     AVR_TIFR0   TIFR0;
00076     AVR_TIFR1   TIFR1;
00077     AVR_TIFR2   TIFR2;
00078
00079     //-- 0x38
00080     uint8_t     RESERVED_0x38;
00081     uint8_t     RESERVED_0x39;
00082     uint8_t     RESERVED_0x3A;
00083
00084     //-- 0x3B
00085     AVR_PCIFR   PCIFR;
00086     AVR_EIFR    EIFR;
00087     AVR_EIMSK   EIMSK;
00088
00089     //-- 0x3E
00090     uint8_t     GPIOR0;
00091
00092     //-- 0x3F
00093     AVR_EECR    EECR;
00094
00095     //-- 0x40
00096     uint8_t     EEDR;
00097     uint8_t     EEARL;
00098     uint8_t     EEARH;
00099
00100     //-- 0x43
00101     AVR_GTCCR   GTCCR;
00102     AVR_TCCR0A  TCCR0A;
00103     AVR_TCCR0B  TCCR0B;
00104     uint8_t     TCNT0;
00105     uint8_t     OCR0A;
00106     uint8_t     OCR0B;
00107
00108     //-- 0x49
00109     uint8_t     RESERVED_0x49;
00110     uint8_t     GPIOR1;
00111     uint8_t     GPIOR2;
00112
00113     AVR_SPCR    SPCR;
00114     AVR_SPSR    SPSR;
00115     uint8_t     SPDR;
00116
00117     uint8_t     RESERVED_0x4F;
00118     AVR_ACSR    ACSR;
00119
00120     uint8_t     RESERVED_0x51;
00121     uint8_t     RESERVED_0x52;
00122
00123     //-- 0x53
00124     AVR_SMCR    SMCR;
00125     AVR_MCUSR   MCUSR;
00126     AVR_MCUCR   MCUCR;
00127     uint8_t     RESERVED_0x56;
00128
00129     AVR_SPMCSR  SPMCSR;
00130     uint8_t     RESERVED_0x58;
00131     uint8_t     RESERVED_0x59;
00132     uint8_t     RESERVED_0x5A;
00133     uint8_t     RESERVED_0x5B;
00134     uint8_t     RESERVED_0x5C;
00135     AVR_SPL     SPL;
00136     AVR_SPH     SPH;
00137     AVR_SREG    SREG;
00138
00139     //-- 0x60
00140     AVR_WDTCSR  WDTCSR;
00141     AVR_CLKPR   CLKPR;
00142     uint8_t     RESERVED_0x62;
00143     uint8_t     RESERVED_0x63;
00144     AVR_PRR     PRR;
```

```
00145     uint8_t     RESERVED_0x65;
00146     uint8_t     OSCCAL;
00147     uint8_t     RESERVED_0x67;
00148
00149     AVR_PCICR   PCICR;
00150     AVR_EICRA   EICRA;
00151     uint8_t     RESERVED_0x6A;
00152
00153     AVR_PCMSK0  PCMSK0;
00154     AVR_PCMSK1  PCMSK1;
00155     AVR_PCMSK2  PCMSK2;
00156     AVR_TIMSK0  TIMSK0;
00157     AVR_TIMSK1  TIMSK1;
00158     AVR_TIMSK2  TIMSK2;
00159
00160     uint8_t     RESERVED_0x71;
00161     uint8_t     RESERVED_0x72;
00162     uint8_t     RESERVED_0x73;
00163     uint8_t     RESERVED_0x74;
00164     uint8_t     RESERVED_0x75;
00165     uint8_t     RESERVED_0x76;
00166     uint8_t     RESERVED_0x77;
00167
00168     uint8_t     ADCL;
00169     uint8_t     ADCH;
00170     AVR_ADCSRA  ADSRA;
00171     AVR_ADCSRB  ADSRB;
00172     AVR_ADMUX   ADMXUX;
00173     uint8_t     RESERVED_0x7F;
00174
00175     AVR_DIDR0   DIDR0;
00176     AVR_DIDR1   DIDR1;
00177     AVR_TCCR1A  TCCR1A;
00178     AVR_TCCR1B  TCCR1B;
00179     AVR_TCCR1C  TCCR1C;
00180     uint8_t     RESERVED_0x83;
00181
00182     uint8_t     TCNT1L;
00183     uint8_t     TCNT1H;
00184     uint8_t     ICR1L;
00185     uint8_t     ICR1H;
00186     uint8_t     OCR1AL;
00187     uint8_t     OCR1AH;
00188     uint8_t     OCR1BL;
00189     uint8_t     OCR1BH;
00190
00191     uint8_t     RESERVED_0x8C;
00192     uint8_t     RESERVED_0x8D;
00193     uint8_t     RESERVED_0x8E;
00194     uint8_t     RESERVED_0x8F;
00195
00196     uint8_t     RESERVED_0x90;
00197     uint8_t     RESERVED_0x91;
00198     uint8_t     RESERVED_0x92;
00199     uint8_t     RESERVED_0x93;
00200     uint8_t     RESERVED_0x94;
00201     uint8_t     RESERVED_0x95;
00202     uint8_t     RESERVED_0x96;
00203     uint8_t     RESERVED_0x97;
00204     uint8_t     RESERVED_0x98;
00205     uint8_t     RESERVED_0x99;
00206     uint8_t     RESERVED_0x9A;
00207     uint8_t     RESERVED_0x9B;
00208     uint8_t     RESERVED_0x9C;
00209     uint8_t     RESERVED_0x9D;
00210     uint8_t     RESERVED_0x9E;
00211     uint8_t     RESERVED_0x9F;
00212
00213     uint8_t     RESERVED_0xA0;
00214     uint8_t     RESERVED_0xA1;
00215     uint8_t     RESERVED_0xA2;
00216     uint8_t     RESERVED_0xA3;
00217     uint8_t     RESERVED_0xA4;
00218     uint8_t     RESERVED_0xA5;
00219     uint8_t     RESERVED_0xA6;
00220     uint8_t     RESERVED_0xA7;
00221     uint8_t     RESERVED_0xA8;
00222     uint8_t     RESERVED_0xA9;
00223     uint8_t     RESERVED_0xAA;
00224     uint8_t     RESERVED_0xAB;
00225     uint8_t     RESERVED_0xAC;
00226     uint8_t     RESERVED_0xAD;
00227     uint8_t     RESERVED_0xAE;
00228     uint8_t     RESERVED_0xAF;
00229
00230     //--0xB0
00231     AVR_TCCR2A  TCCR2A;
```

```
00232     AVR_TCCR2B  TCCR2B;
00233     uint8_t     TCNT2;
00234     uint8_t     OCR2A;
00235     uint8_t     OCR2B;
00236
00237     uint8_t     RESERVED_0xB5;
00238     AVR_ASSR    ASSR;
00239     uint8_t     RESERVED_0xB7;
00240     uint8_t     TWBR;
00241     AVR_TWSR    TWSR;
00242     AVR_TWAR    TWAR;
00243     uint8_t     TWDR;
00244     AVR_TWCR    TWCR;
00245     AVR_TWAMR   TWAMR;
00246
00247     uint8_t     RESERVED_0xBE;
00248     uint8_t     RESERVED_0xBF;
00249
00250     //--0xC0
00251     AVR_UCSR0A  UCSR0A;
00252     AVR_UCSR0B  UCSR0B;
00253     AVR_UCSR0C  UCSR0C;
00254
00255     uint8_t     RESERVED_0xC3;
00256
00257     uint8_t     UBRR0L;
00258     uint8_t     UBRR0H;
00259     uint8_t     UDR0;
00260
00261     uint8_t     RESERVED_0xC7;
00262     uint8_t     RESERVED_0xC8;
00263     uint8_t     RESERVED_0xC9;
00264     uint8_t     RESERVED_0xCA;
00265     uint8_t     RESERVED_0xCB;
00266     uint8_t     RESERVED_0xCC;
00267     uint8_t     RESERVED_0xCD;
00268     uint8_t     RESERVED_0xCE;
00269     uint8_t     RESERVED_0xCF;
00270
00271     uint8_t     RESERVED_0xD0;
00272     uint8_t     RESERVED_0xD1;
00273     uint8_t     RESERVED_0xD2;
00274     uint8_t     RESERVED_0xD3;
00275     uint8_t     RESERVED_0xD4;
00276     uint8_t     RESERVED_0xD5;
00277     uint8_t     RESERVED_0xD6;
00278     uint8_t     RESERVED_0xD7;
00279     uint8_t     RESERVED_0xD8;
00280     uint8_t     RESERVED_0xD9;
00281     uint8_t     RESERVED_0xDA;
00282     uint8_t     RESERVED_0xDB;
00283     uint8_t     RESERVED_0xDC;
00284     uint8_t     RESERVED_0xDD;
00285     uint8_t     RESERVED_0xDE;
00286     uint8_t     RESERVED_0xDF;
00287
00288     uint8_t     RESERVED_0xE0;
00289     uint8_t     RESERVED_0xE1;
00290     uint8_t     RESERVED_0xE2;
00291     uint8_t     RESERVED_0xE3;
00292     uint8_t     RESERVED_0xE4;
00293     uint8_t     RESERVED_0xE5;
00294     uint8_t     RESERVED_0xE6;
00295     uint8_t     RESERVED_0xE7;
00296     uint8_t     RESERVED_0xE8;
00297     uint8_t     RESERVED_0xE9;
00298     uint8_t     RESERVED_0xEA;
00299     uint8_t     RESERVED_0xEB;
00300     uint8_t     RESERVED_0xEC;
00301     uint8_t     RESERVED_0xED;
00302     uint8_t     RESERVED_0xEE;
00303     uint8_t     RESERVED_0xEF;
00304
00305     uint8_t     RESERVED_0xF0;
00306     uint8_t     RESERVED_0xF1;
00307     uint8_t     RESERVED_0xF2;
00308     uint8_t     RESERVED_0xF3;
00309     uint8_t     RESERVED_0xF4;
00310     uint8_t     RESERVED_0xF5;
00311     uint8_t     RESERVED_0xF6;
00312     uint8_t     RESERVED_0xF7;
00313     uint8_t     RESERVED_0xF8;
00314     uint8_t     RESERVED_0xF9;
00315     uint8_t     RESERVED_0xFA;
00316     uint8_t     RESERVED_0xFB;
00317     uint8_t     RESERVED_0xFC;
00318     uint8_t     RESERVED_0xFD;
```

```
00319     uint8_t     RESERVED_0xFE;
00320     uint8_t     RESERVED_0xFF;
00321
00322 } AVRRegisterFile;
00323
00324
00325 #endif // __AVR_REGISTERFILE_H__
```

4.49 breakpoint.c File Reference

Implements instruction breakpoints for debugging based on code path.

```
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include "breakpoint.h"
```

Functions

- void BreakPoint_Insert (uint16_t u16Addr_)

  BreakPoint_Insert.
- void BreakPoint_Delete (uint16_t u16Addr_)

  BreakPoint_Delete.
- bool BreakPoint_EnabledAtAddress (uint16_t u16Addr_)

  BreakPoint_EnabledAtAddress.

4.49.1 Detailed Description

Implements instruction breakpoints for debugging based on code path.

Definition in file breakpoint.c.

4.49.2 Function Documentation

4.49.2.1 void BreakPoint_Delete (uint16_t *u16Addr_*)

BreakPoint_Delete.

Delete a breakpoint at a given address (if it exists). Has no effect if there isn't a breakpoint installed at the location

Parameters

*u16Addr_*	Address of the breakpoint to delete.

Definition at line 55 of file breakpoint.c.

4.49.2.2 bool BreakPoint_EnabledAtAddress (uint16_t *u16Addr_*)

BreakPoint_EnabledAtAddress.

Check to see whether or not a CPU execution breakpoint has been installed at the given address.

Parameters

*u16Addr_*	Address (in flash) to check for breakpoint on.

Returns

true if a breakpoint has been set on the given address.

Definition at line 95 of file breakpoint.c.

4.49.2.3 void BreakPoint_Insert (uint16_t *u16Addr_*)

BreakPoint_Insert.

Insert a CPU breakpoint at a given address. Has no effect if a breakpoint is already present at the given address.

Parameters

*u16Addr_*	Address of the breakpoint.

Definition at line 29 of file breakpoint.c.

4.50 breakpoint.c

```
00001 /*****************************************************************************
00002  *      (     (                       (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(    )\     (   (  (()/(   | -- [ Funkenstein ] ------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\   |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+-----------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #include <stdint.h>
00022 #include <stdbool.h>
00023 #include <stdio.h>
00024 #include <stdlib.h>
00025
00026 #include "breakpoint.h"
00027
00028 //-----------------------------------------------------------------------------
00029 void BreakPoint_Insert( uint16_t u16Addr_ )
00030 {
00031     // Don't add multiple breakpoints at the same address
00032     if (BreakPoint_EnabledAtAddress(  u16Addr_ ))
00033     {
00034         return;
00035     }
00036
00037     BreakPoint_t *pstNewBreak = NULL;
00038
00039     pstNewBreak = (BreakPoint_t*)malloc( sizeof(BreakPoint_t) );
00040
00041     pstNewBreak->next = stCPU.pstBreakPoints;
00042     pstNewBreak->prev = NULL;
00043
00044     pstNewBreak->u16Addr = u16Addr_;
00045
00046     if (stCPU.pstBreakPoints)
00047     {
00048         BreakPoint_t *pstTemp = stCPU.pstBreakPoints;
00049         pstTemp->prev = pstNewBreak;
00050     }
00051     stCPU.pstBreakPoints = pstNewBreak;
00052 }
00053
00054 //-----------------------------------------------------------------------------
00055 void BreakPoint_Delete( uint16_t u16Addr_ )
00056 {
00057     BreakPoint_t *pstTemp = stCPU.pstBreakPoints;
```

```
00058
00059     while (pstTemp)
00060     {
00061         if (pstTemp->u16Addr == u16Addr_)
00062         {
00063             // Remove node -- reconnect surrounding elements
00064             BreakPoint_t *pstNext = pstTemp->next;
00065             if (pstNext)
00066             {
00067                 pstNext->prev = pstTemp->prev;
00068             }
00069
00070             BreakPoint_t *pstPrev = pstTemp->prev;
00071             if (pstPrev)
00072             {
00073                 pstPrev->next = pstTemp->next;
00074             }
00075
00076             // Adjust list-head if necessary
00077             if (pstTemp == stCPU.pstBreakPoints)
00078             {
00079                 stCPU.pstBreakPoints = pstNext;
00080             }
00081
00082             // Free the node/iterate to next node.
00083             pstPrev = pstTemp;
00084             pstTemp = pstTemp->next;
00085             free(pstPrev);
00086         }
00087         else
00088         {
00089             pstTemp = pstTemp->next;
00090         }
00091     }
00092 }
00093
00094 //-----------------------------------------------------------------------------
00095 bool BreakPoint_EnabledAtAddress( uint16_t u16Addr_ )
00096 {
00097     BreakPoint_t *pstTemp = stCPU.pstBreakPoints;
00098
00099     while (pstTemp)
00100     {
00101         if (pstTemp->u16Addr == u16Addr_)
00102         {
00103             return true;
00104         }
00105         pstTemp = pstTemp->next;
00106     }
00107     return false;
00108 }
```

4.51 breakpoint.h File Reference

Implements instruction breakpoints for debugging based on code path.

```
#include <stdint.h>
#include <stdbool.h>
#include "avr_cpu.h"
```

Data Structures

- struct _BreakPoint

  Node-structure for a linked-list of breakpoint addresses.

Typedefs

- typedef struct _BreakPoint BreakPoint_t

  Node-structure for a linked-list of breakpoint addresses.

Functions

- void BreakPoint_Insert (uint16_t u16Addr_)

  BreakPoint_Insert.
- void BreakPoint_Delete (uint16_t u16Addr_)

  BreakPoint_Delete.
- bool BreakPoint_EnabledAtAddress (uint16_t u16Addr_)

  BreakPoint_EnabledAtAddress.

4.51.1 Detailed Description

Implements instruction breakpoints for debugging based on code path.

Definition in file breakpoint.h.

4.51.2 Function Documentation

4.51.2.1 void BreakPoint_Delete (uint16_t *u16Addr_*)

BreakPoint_Delete.

Delete a breakpoint at a given address (if it exists). Has no effect if there isn't a breakpoint installed at the location

Parameters

*u16Addr_*	Address of the breakpoint to delete.

Definition at line 55 of file breakpoint.c.

4.51.2.2 bool BreakPoint_EnabledAtAddress (uint16_t *u16Addr_*)

BreakPoint_EnabledAtAddress.

Check to see whether or not a CPU execution breakpoint has been installed at the given address.

Parameters

*u16Addr_*	Address (in flash) to check for breakpoint on.

Returns

true if a breakpoint has been set on the given address.

Definition at line 95 of file breakpoint.c.

4.51.2.3 void BreakPoint_Insert (uint16_t *u16Addr_*)

BreakPoint_Insert.

Insert a CPU breakpoint at a given address. Has no effect if a breakpoint is already present at the given address.

Parameters

*u16Addr_*	Address of the breakpoint.

Definition at line 29 of file breakpoint.c.

4.52 breakpoint.h

```
00001 /*************************************************************************
00002  *      (     (                        (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\      (   (  (()/(   | -- [ Funkenstein ] ------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] -------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *************************************************************************/
00021 #ifndef __BREAKPOINT_H__
00022 #define __BREAKPOINT_H__
00023
00024 #include <stdint.h>
00025 #include <stdbool.h>
00026
00027 #include "avr_cpu.h"
00028
00029 //-------------------------------------------------------------------------
00033 typedef struct _BreakPoint
00034 {
00035     struct _BreakPoint *next;
00036     struct _BreakPoint *prev;
00037
00038     uint16_t    u16Addr;
00039 } BreakPoint_t;
00040
00041 //-------------------------------------------------------------------------
00050 void BreakPoint_Insert( uint16_t u16Addr_ );
00051
00052 //-------------------------------------------------------------------------
00061 void BreakPoint_Delete( uint16_t u16Addr_ );
00062
00063 //-------------------------------------------------------------------------
00073 bool BreakPoint_EnabledAtAddress( uint16_t u16Addr_ );
00074
00075 #endif
00076
```

4.53 emu_config.h File Reference

configuration file - used to configure features used by the emulator at build-time.

```
#include <stdint.h>
#include <stdbool.h>
```

Macros

- #define **CONFIG_IO_ADDRESS_BYTES** (256)
- #define FEATURE_USE_JUMPTABLES (1)

  Jump-tables can be used to optimize the execution of opcodes by building CPU instruction decode and execute jump tables at runtime.

- #define CONFIG_TRACEBUFFER_SIZE (1000)

  Sets the "execution history" buffer to a set number of instructions.

4.53.1 Detailed Description

configuration file - used to configure features used by the emulator at build-time.

Definition in file emu_config.h.

4.53.2 Macro Definition Documentation

4.53.2.1 #define CONFIG_TRACEBUFFER_SIZE (1000)

Sets the "execution history" buffer to a set number of instructions.

The larger the number, the further back in time you can look. Note that for each sample we store a CPU register context, as well as a variety of bookkeeping information. Full contents of RAM are not preserved here, however.

Definition at line 53 of file emu_config.h.

4.53.2.2 #define FEATURE_USE_JUMPTABLES (1)

Jump-tables can be used to optimize the execution of opcodes by building CPU instruction decode and execute jump tables at runtime.

Once the tables are generated, decode/execute are reduced to a lookup table operation, as opposed to a complex series of if/else statements for each decode/execute of a 16-bit opcode.

This comes at a cost, however, as jump-tables require RAM (one function pointer for each possible 16-bit value, for each lookup type).

It's a huge speed boost though, so it is recommended to keep this feature enabled unless you're trying to self-host flavr on a low-resource microcontroller (or even self-hosting a virtual AVR on an AVR...).

Definition at line 44 of file emu_config.h.

4.54 emu_config.h

```
00001 /*************************************************************************
00002  *      (     (                       (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\      (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ----------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] ------------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] ------------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                         | "Yeah, it does Arduino..."
00011  * ----------------------------------------+-----------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *************************************************************************/
00022 #ifndef __EMU_CONFIG_H__
00023 #define __EMU_CONFIG_H__
00024
00025 #include <stdint.h>
00026 #include <stdbool.h>
00027
00028 #define CONFIG_IO_ADDRESS_BYTES        (256)                    // First bytes of address space are I/O
       range
00029
00044 #define FEATURE_USE_JUMPTABLES          (1)
00045
00053 #define CONFIG_TRACEBUFFER_SIZE        (1000)
00054
00055 #endif
00056
```

4.55 flavr.c File Reference

Main AVR emulator entrypoint, commandline-use with built-in interactive debugger.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include "emu_config.h"
#include "variant.h"
#include "avr_coreregs.h"
#include "avr_periphregs.h"
#include "avr_op_cycles.h"
#include "avr_op_decode.h"
#include "avr_op_size.h"
#include "avr_cpu_print.h"
#include "avr_cpu.h"
#include "avr_loader.h"
#include "avr_disasm.h"
#include "trace_buffer.h"
#include "options.h"
#include "interactive.h"
#include "breakpoint.h"
#include "watchpoint.h"
#include "mega_uart.h"
#include "mega_eint.h"
#include "mega_timer16.h"
#include "mega_timer8.h"
```

Enumerations

- enum **ErrorReason_t** {
  EEPROM_TOO_BIG, **RAM_TOO_BIG**, **RAM_TOO_SMALL**, **ROM_TOO_BIG**,
  INVALID_HEX_FILE, **INVALID_VARIANT** }

Functions

- void **splash** (void)
- void **error_out** (ErrorReason_t eReason_)
- void **emulator_loop** (void)
- void **add_plugins** (void)
- void **flavr_disasm** (void)
- void **emulator_init** (void)
- int **main** (int argc, char ∗∗argv)

Variables

- static TraceBuffer_t **stTraceBuffer**

4.55.1 Detailed Description

Main AVR emulator entrypoint, commandline-use with built-in interactive debugger.

Definition in file flavr.c.

4.56 flavr.c

```
00001 /*****************************************************************************
00002  *      (     (                          (     |
00003  *    )\ )  )\ )    (              )\ )   |
00004  *   (()/( (()/(    )\     (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ----------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] ------------------
00008  *   | __| | |__    / _ \   \ V /  |   /   | -- [ Runtime ] ------------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\   |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00022 #include <stdio.h>
00023 #include <stdlib.h>
00024 #include <string.h>
00025 #include <stdint.h>
00026
00027 #include "emu_config.h"
00028 #include "variant.h"
00029
00030 //------------------------------------------------------------------------------
00031 #include "avr_coreregs.h"
00032 #include "avr_periphregs.h"
00033 #include "avr_op_cycles.h"
00034 #include "avr_op_decode.h"
00035 #include "avr_op_size.h"
00036 #include "avr_cpu_print.h"
00037 #include "avr_cpu.h"
00038 #include "avr_loader.h"
00039
00040 #include "avr_disasm.h"
00041 #include "trace_buffer.h"
00042 #include "options.h"
00043 #include "interactive.h"
00044 #include "breakpoint.h"
00045 #include "watchpoint.h"
00046
00047 //------------------------------------------------------------------------------
00048 #include "mega_uart.h"
00049 #include "mega_eint.h"
00050 #include "mega_timer16.h"
00051 #include "mega_timer8.h"
00052
00053 //------------------------------------------------------------------------------
00054 typedef enum
00055 {
00056     EEPROM_TOO_BIG,
00057     RAM_TOO_BIG,
00058     RAM_TOO_SMALL,
00059     ROM_TOO_BIG,
00060     INVALID_HEX_FILE,
00061     INVALID_VARIANT
00062 } ErrorReason_t;
00063
00064 //------------------------------------------------------------------------------
00065 static TraceBuffer_t stTraceBuffer;
00066
00067 //------------------------------------------------------------------------------
00068 void splash(void)
00069 {
00070 printf(
00071        "* ---------------------------------------+---------------------------------\n"
00072        "*      (     (                          (     |\n"
00073        "*    )\\ )  )\\ )    (              )\\ )   |\n"
00074        "*   (()/( (()/(    )\\     (   (  (()/(   | -- [ Funkenstein ] -------------\n"
00075        "*    /(_)) /(_))((((_)()\\  )\\  /(_))     | -- [ Litle ] --------------------\n"
00076        "*   (_))_|(_))    )\\ _ )\\ ((_)((_)(_))    | -- [ AVR ] ----------------------\n"
00077        "*   | |_  | |     (_)_\\(_)\\ \\ / / | _ \\   | -- [ Virtual ] ------------------\n"
00078        "*   | __| | |__    / _ \\   \\ V /  |   /   | -- [ Runtime ] ------------------\n"
00079        "*   |_|   |____| /_/ \\_\\   \\_/   |_|_\\   |\n"
00080        "*                                        | \"From the makers of Mark3!\"\n"
00081        "* ---------------------------------------+---------------------------------\n"
00082        "* (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved\n"
00083        "*     See license.txt for details\n"
00084       );
00085 }
00086
00087 //------------------------------------------------------------------------------
00088 void error_out( ErrorReason_t eReason_ )
00089 {
00090     switch (eReason_)
00091     {
```

```
00092         case EEPROM_TOO_BIG:
00093             printf( "EERPOM Size specified is too large\n" );
00094             break;
00095         case RAM_TOO_BIG:
00096             printf( "RAM Size specified is too large\n" );
00097             break;
00098         case RAM_TOO_SMALL:
00099             printf( "RAM Size specified is too small\n" );
00100             break;
00101         case ROM_TOO_BIG:
00102             printf( "ROM Size specified is too large\n" );
00103             break;
00104         case INVALID_HEX_FILE:
00105             printf( "HEX Programming file cannot be loaded\n");
00106             break;
00107         case INVALID_VARIANT:
00108             printf( "Unknown variant not supported\n");
00109             break;
00110         default:
00111             printf( "Some other reason\n" );
00112     }
00113     exit (-1);
00114 }
00115
00116 //-------------------------------------------------------------------------
00117 void emulator_loop(void)
00118 {
00119     bool bUseTrace = false;
00120
00121     if ( Options_GetByName("--trace") && Options_GetByName("--debug") )
00122     {
00123         bUseTrace = true;
00124     }
00125
00126     while (1)
00127     {
00128         // Check to see if we've hit a breakpoint
00129         if (BreakPoint_EnabledAtAddress(stCPU.u16PC))
00130         {
00131             Interactive_Set();
00132         }
00133
00134         // Check to see if we're in interactive debug mode, and thus need to wait for input
00135         Interactive_CheckAndExecute();
00136
00137         // Store the current CPU state into the tracebuffer
00138         if (bUseTrace)
00139         {
00140             TraceBuffer_StoreFromCPU(&stTraceBuffer);
00141         }
00142
00143         // Execute a machine cycle
00144         CPU_RunCycle();
00145     }
00146     // doesn't return, except by quitting from debugger, or by signal.
00147 }
00148
00149 //-------------------------------------------------------------------------
00150 void add_plugins(void)
00151 {
00152     CPU_AddPeriph(&stUART);
00153     CPU_AddPeriph(&stEINT_a);
00154     CPU_AddPeriph(&stEINT_b);
00155     CPU_AddPeriph(&stTimer16);
00156     CPU_AddPeriph(&stTimer16a);
00157     CPU_AddPeriph(&stTimer16b);
00158     CPU_AddPeriph(&stTimer8);
00159     CPU_AddPeriph(&stTimer8a);
00160     CPU_AddPeriph(&stTimer8b);
00161 }
00162
00163 //-------------------------------------------------------------------------
00164 void flavr_disasm(void)
00165 {
00166     uint32_t u32Size;
00167
00168     u32Size = stCPU.u32ROMSize / sizeof(uint16_t);
00169     stCPU.u16PC = 0;
00170
00171     while (stCPU.u16PC < u32Size)
00172     {
00173         uint16_t OP = stCPU.pu16ROM[stCPU.u16PC];
00174         printf("0x%04X: [0x%04X] ", stCPU.u16PC, OP);
00175         AVR_Decode(OP);
00176         AVR_Disasm_Function(OP)();
00177         stCPU.u16PC += AVR_Opcode_Size(OP);
00178     }
```

```
00179     exit(0);
00180 }
00181
00182 //-----------------------------------------------------------------------------
00183 void emulator_init(void)
00184 {
00185     AVR_CPU_Config_t stConfig;
00186
00187     // -- Initialize the emulator based on command-line args
00188     const AVR_Variant_t *pstVariant;
00189
00190     pstVariant = Variant_GetByName( Options_GetByName("--variant") );
00191     if (!pstVariant)
00192     {
00193         error_out( INVALID_VARIANT );
00194     }
00195
00196     stConfig.u32EESize  = pstVariant->u32EESize;
00197     stConfig.u32RAMSize = pstVariant->u32RAMSize;
00198     stConfig.u32ROMSize = pstVariant->u32ROMSize;
00199
00200     if (stConfig.u32EESize >= 32768)
00201     {
00202         error_out( EEPROM_TOO_BIG );
00203     }
00204
00205     if (stConfig.u32RAMSize >= 65535)
00206     {
00207         error_out( RAM_TOO_BIG );
00208     }
00209     else if (stConfig.u32RAMSize < 256)
00210     {
00211         error_out( RAM_TOO_SMALL );
00212     }
00213
00214     if (stConfig.u32ROMSize >= (256*1024))
00215     {
00216         error_out( ROM_TOO_BIG );
00217     }
00218
00219     CPU_Init(&stConfig);
00220
00221     TraceBuffer_Init( &stTraceBuffer);
00222     Interactive_Init( &stTraceBuffer );
00223
00224     // Only insert a breakpoint/enter interactive debugging mode if specified.
00225     // Otherwise, start with the emulator running.
00226     if (Options_GetByName("--debug"))
00227     {
00228         BreakPoint_Insert( 0 );
00229     }
00230
00231     if (Options_GetByName("--hexfile"))
00232     {
00233         if( !AVR_Load_HEX( Options_GetByName("--hexfile") ) ) {
00234             error_out( INVALID_HEX_FILE );
00235         }
00236     }
00237     else
00238     {
00239         error_out( INVALID_HEX_FILE );
00240     }
00241
00242     if (Options_GetByName("--disasm"))
00243     {
00244         // terminates after disassembly is complete
00245         flavr_disasm();
00246     }
00247
00248     add_plugins();
00249 }
00250
00251 //-----------------------------------------------------------------------------
00252 int main( int argc, char **argv )
00253 {
00254     // Initialize all emulator data
00255     Options_Init(argc, argv);
00256
00257     if (!Options_GetByName("--silent"))
00258     {
00259         splash();
00260     }
00261
00262     emulator_init();
00263
00264     // Run the emulator/debugger loop.
00265     emulator_loop();
```

```
00266
00267     return 0;
00268
00269 }
```

4.57 intel_hex.c File Reference

Module for decoding Intel hex formatted programming files.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <sys/stat.h>
#include <sys/fcntl.h>
#include "emu_config.h"
#include "intel_hex.h"
```

Functions

- void HEX_Print_Record (HEX_Record_t *stRecord_)

  HEX_Print_Record.
- static bool **HEX_Read_Header** (int fd_)
- static bool **HEX_Next_Line** (int fd_, HEX_Record_t *stRecord_)
- static bool **HEX_Read_Record_Type** (int fd_, HEX_Record_t *stRecord_)
- static bool **HEX_Read_Byte_Count** (int fd_, HEX_Record_t *stRecord_)
- static bool **HEX_Read_Address** (int fd_, HEX_Record_t *stRecord_)
- static bool **HEX_Read_Data** (int fd_, HEX_Record_t *stRecord_)
- static bool **HEX_Read_Checksum** (int fd_, HEX_Record_t *stRecord_)
- static bool **HEX_Line_Validate** (HEX_Record_t *stRecord_)
- bool HEX_Read_Record (int fd_, HEX_Record_t *stRecord_)

  HEX_Read_Record.

4.57.1 Detailed Description

Module for decoding Intel hex formatted programming files.

Definition in file intel_hex.c.

4.57.2 Function Documentation

4.57.2.1 void HEX_Print_Record (HEX_Record_t * *stRecord_*)

HEX_Print_Record.

Print the contents of a single Intel hex record to standard output.

Parameters

*stRecord_*	Pointer to a valid, initialized hex record

Definition at line 33 of file intel_hex.c.

4.57.2.2 bool HEX_Read_Record (int *fd_*, HEX_Record_t * *stRecord_*)

HEX_Read_Record.

Read the next Intel Hex file record from an open Intel Hex programming file.

Parameters

*fd_*	[in] Open file handle corresponding to the hex file
*stRecord_*	[out] Pointer to a valid hex record struct

Returns

true - hex record read succeeded, false - failure or EOF.

Definition at line 216 of file intel_hex.c.

4.58 intel_hex.c

```
00001 /*******************************************************************************
00002  *     (     (                      (     |
00003  *    )\ )  )\ )    (              )\ )   |
00004  *   (()/( (()/(    )\     (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \    \ V /  |   /   | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\    \_/   |_|_\   |
00010  *                                         | "Yeah, it does Arduino..."
00011  * ----------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  ******************************************************************************/
00021 #include <stdio.h>
00022 #include <stdlib.h>
00023 #include <string.h>
00024 #include <stdint.h>
00025 #include <sys/stat.h>
00026 #include <sys/fcntl.h>
00027
00028 #include "emu_config.h"
00029
00030 #include "intel_hex.h"
00031
00032 //------------------------------------------------------------------------------
00033 void HEX_Print_Record( HEX_Record_t *stRecord_ )
00034 {
00035     printf(  "Line: %d\n"
00036              "ByteCount: %d\n"
00037              "RecordType: %d\n"
00038              "Address: %X\n"
00039              "Data:",
00040              stRecord_->u32Line,
00041              stRecord_->u8ByteCount,
00042              stRecord_->u8RecordType,
00043              stRecord_->u16Address );
00044     int i;
00045     for (i = 0; i < stRecord_->u8ByteCount; i++)
00046     {
00047         printf( " %02X", stRecord_->u8Data[i]);
00048     }
00049     printf( "\n" );
00050 }
00051
00052 //------------------------------------------------------------------------------
00053 static bool HEX_Read_Header( int fd_ )
00054 {
00055     ssize_t bytes_read;
00056     char acBuf[2] = {0};
00057
00058     bytes_read = read(fd_, acBuf, 1);
00059     if (1 != bytes_read)
00060     {
00061         return false;
00062     }
00063     if (':' == acBuf[0])
00064     {
00065         return true;
00066     }
00067     return false;
00068 }
00069
00070 //------------------------------------------------------------------------------
00071 static bool HEX_Next_Line( int fd_, HEX_Record_t *stRecord_ )
00072 {
```

```
00073     ssize_t bytes_read;
00074     char acBuf[2] = {0};
00075
00076     stRecord_->u32Line++;
00077     do
00078     {
00079         bytes_read = read(fd_, acBuf, 1);
00080         if (1 != bytes_read)
00081         {
00082             return false;
00083         }
00084     } while(acBuf[0] != '\n');
00085
00086     return true;
00087 }
00088
00089 //---------------------------------------------------------------------------
00090 static bool HEX_Read_Record_Type( int fd_, HEX_Record_t *stRecord_ )
00091 {
00092     ssize_t bytes_read;
00093     uint32_t u32Hex;
00094     char acBuf[3] = {0};
00095
00096     bytes_read = read(fd_, acBuf, 2);
00097     if (2 != bytes_read)
00098     {
00099         return false;
00100     }
00101     sscanf(acBuf, "%02X", &u32Hex);
00102     stRecord_->u8RecordType = (uint8_t)u32Hex;
00103
00104     if (stRecord_->u8RecordType >= RECORD_TYPE_MAX)
00105     {
00106         return false;
00107     }
00108
00109     return true;
00110 }
00111
00112 //---------------------------------------------------------------------------
00113 static bool HEX_Read_Byte_Count( int fd_, HEX_Record_t *stRecord_ )
00114 {
00115     ssize_t bytes_read;
00116     uint32_t u32Hex;
00117     char acBuf[3] = {0};
00118
00119     bytes_read = read(fd_, acBuf, 2);
00120     if (2 != bytes_read)
00121     {
00122         return false;
00123     }
00124     sscanf(acBuf, "%02X", &u32Hex);
00125     stRecord_->u8ByteCount = (uint8_t)u32Hex;
00126
00127     return true;
00128 }
00129
00130 //---------------------------------------------------------------------------
00131 static bool HEX_Read_Address( int fd_, HEX_Record_t *stRecord_ )
00132 {
00133     ssize_t bytes_read;
00134     uint32_t u32Hex;
00135     char acBuf[5] = {0};
00136
00137     bytes_read = read(fd_, acBuf, 4);
00138     if (4 != bytes_read)
00139     {
00140         return false;
00141     }
00142     sscanf(acBuf, "%04X", &u32Hex);
00143     stRecord_->u16Address = (uint16_t)u32Hex;
00144
00145     return true;
00146 }
00147
00148 //---------------------------------------------------------------------------
00149 static bool HEX_Read_Data( int fd_, HEX_Record_t *stRecord_ )
00150 {
00151     ssize_t bytes_read;
00152     uint32_t u32Hex;
00153     char acBuf[MAX_HEX_DATA_BYTES * 2] = {0};
00154
00155     int i;
00156     for (i = 0; i < stRecord_->u8ByteCount; i++)
00157     {
00158         // printf("i:%d\n", i);
00159         bytes_read = read(fd_, acBuf, 2);
```

```
00160         if (2 != bytes_read)
00161         {
00162             return false;
00163         }
00164         sscanf(acBuf, "%02X", &u32Hex);
00165         stRecord_->u8Data[i] = (uint8_t)u32Hex;
00166     }
00167 
00168     return true;
00169 }
00170 
00171 //-----------------------------------------------------------------------------
00172 static bool HEX_Read_Checksum( int fd_, HEX_Record_t *stRecord_ )
00173 {
00174     ssize_t bytes_read;
00175     uint32_t u32Hex;
00176     char acBuf[3] = {0,0,0};
00177 
00178     bytes_read = read(fd_, acBuf, 2);
00179     if (2 != bytes_read)
00180     {
00181         return false;
00182     }
00183     sscanf(acBuf, "%02X", &u32Hex);
00184     stRecord_->u8Checksum = (uint8_t)u32Hex;
00185 
00186     return true;
00187 }
00188 
00189 //-----------------------------------------------------------------------------
00190 static bool HEX_Line_Validate( HEX_Record_t *stRecord_ )
00191 {
00192     // Calculate the CRC for the fields in the struct and compare
00193     // against the value read from file...
00194     uint8_t u8CRC = 0;
00195     u8CRC += (uint8_t)(stRecord_->u16Address >> 8);
00196     u8CRC += (uint8_t)(stRecord_->u16Address & 0x00FF);
00197     u8CRC += stRecord_->u8ByteCount;
00198     u8CRC += stRecord_->u8RecordType;
00199 
00200     uint8_t i;
00201     for (i = 0; i < stRecord_->u8ByteCount; i++)
00202     {
00203         u8CRC += stRecord_->u8Data[i];
00204     }
00205 
00206     u8CRC = (~u8CRC) + 1;   // Spec says to take the 2's complement
00207     if (u8CRC != stRecord_->u8Checksum)
00208     {
00209         return false;
00210     }
00211 
00212     return true;
00213 }
00214 
00215 //-----------------------------------------------------------------------------
00216 bool HEX_Read_Record( int fd_, HEX_Record_t *stRecord_ )
00217 {
00218     bool rc = true;
00219     if (rc)
00220     {
00221         rc = HEX_Read_Header(fd_);
00222     }
00223     if (rc)
00224     {
00225         rc = HEX_Read_Byte_Count(fd_, stRecord_);
00226     }
00227     if (rc)
00228     {
00229         rc = HEX_Read_Address(fd_, stRecord_);
00230     }
00231     if (rc)
00232     {
00233         rc = HEX_Read_Record_Type(fd_, stRecord_);
00234     }
00235     if (rc)
00236     {
00237         rc = HEX_Read_Data(fd_, stRecord_);
00238     }
00239     if (rc)
00240     {
00241         rc = HEX_Read_Checksum(fd_, stRecord_);
00242     }
00243     if (rc)
00244     {
00245         rc = HEX_Line_Validate(stRecord_);
00246     }
```

```
00247
00248     HEX_Next_Line(fd_, stRecord_);
00249     return rc;
00250 }
```

4.59 intel_hex.h File Reference

Module for decoding Intel hex formatted programming files.

```
#include <stdint.h>
#include <stdbool.h>
```

Data Structures

- struct HEX_Record_t

  Data type used to represent a single Intel Hex Record.

Macros

- #define **MAX_HEX_DATA_BYTES** (255)
- #define **RECORD_DATA** (0)
- #define **RECORD_EOF** (1)
- #define **RECORD_EXTENDED_SEGMENT** (2)
- #define **RECORD_START_SEGMENT** (3)
- #define **RECORD_EXTENDED_LINEAR** (4)
- #define **RECORD_START_LINEAR** (5)
- #define **RECORD_TYPE_MAX** (5)

Functions

- void HEX_Print_Record (HEX_Record_t *stRecord_)

  HEX_Print_Record.
- bool HEX_Read_Record (int fd_, HEX_Record_t *stRecord_)

  HEX_Read_Record.

4.59.1 Detailed Description

Module for decoding Intel hex formatted programming files.

Definition in file intel_hex.h.

4.59.2 Function Documentation

4.59.2.1 void HEX_Print_Record (HEX_Record_t * *stRecord_*)

HEX_Print_Record.

Print the contents of a single Intel hex record to standard output.

Parameters

*stRecord_*	Pointer to a valid, initialized hex record

Definition at line 33 of file intel_hex.c.

4.59.2.2 bool HEX_Read_Record (int *fd_*, HEX_Record_t * *stRecord_*)

HEX_Read_Record.

Read the next Intel Hex file record from an open Intel Hex programming file.

Parameters

*fd_*	[in] Open file handle corresponding to the hex file
*stRecord_*	[out] Pointer to a valid hex record struct

Returns

true - hex record read succeeded, false - failure or EOF.

Definition at line 216 of file intel_hex.c.

4.60 intel_hex.h

```
00001 /*****************************************************************************
00002  *      (     (                       (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\      (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+-----------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #ifndef __INTEL_HEX_H__
00022 #define __INTEL_HEX_H__
00023
00024 #include <stdint.h>
00025 #include <stdbool.h>
00026
00027 //-----------------------------------------------------------------------------
00028 // Load a hex file into the ROM section of a virtual AVR.
00029 #define MAX_HEX_DATA_BYTES      (255)    // max data bytes per line in a record
00030
00031 //-----------------------------------------------------------------------------
00032 // Record types in the HEX specification
00033 #define RECORD_DATA             (0)
00034 #define RECORD_EOF              (1)
00035 #define RECORD_EXTENDED_SEGMENT (2)
00036 #define RECORD_START_SEGMENT    (3)
00037 #define RECORD_EXTENDED_LINEAR  (4)
00038 #define RECORD_START_LINEAR     (5)
00039
00040 //-----------------------------------------------------------------------------
00041 #define RECORD_TYPE_MAX         (5)
00042
00043 //-----------------------------------------------------------------------------
00044 // For reference, this is the line format for an intel hex record.
00045 // :WWXXYYYYzz.....zzCC
00046 //  Where : = the ":" start code
00047 //  WW = the byte count in the data field
00048 //  XX = the record type
00049 //  YYYY = record address
00050 //  zz = data bytes
00051 //  CC = 2's complement checksum of all fields, excluding start code and checksum
00052
00053 //-----------------------------------------------------------------------------
00057 typedef struct
00058 {
```

```
00059     uint8_t  u8ByteCount;
00060     uint8_t  u8RecordType;
00061     uint16_t u16Address;
00062     uint8_t  u8Data[MAX_HEX_DATA_BYTES];
00063     uint8_t  u8Checksum;
00064     uint32_t u32Line;
00065 } HEX_Record_t;
00066
00067 //------------------------------------------------------------------------
00075 void HEX_Print_Record( HEX_Record_t *stRecord_ );
00076
00077 //------------------------------------------------------------------------
00090 bool HEX_Read_Record( int fd_, HEX_Record_t *stRecord_ );
00091
00092 #endif
```

4.61 interactive.c File Reference

Interactive debugging support.

```
#include "emu_config.h"
#include "avr_cpu.h"
#include "avr_cpu_print.h"
#include "watchpoint.h"
#include "breakpoint.h"
#include "avr_disasm.h"
#include "trace_buffer.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Data Structures

- struct Interactive_Command_t

  Struct type used to map debugger command-line inputs to command handlers.

Typedefs

- typedef bool(* Interactive_Handler)(char *szCommand_)

  Function pointer type used to implement interactive command handlers.

Functions

- static bool Interactive_Continue (char *szCommand_)

  Interactive_Continue.
- static bool Interactive_Step (char *szCommand_)

  Interactive_Step.
- static bool Interactive_Break (char *szCommand_)

  Interactive_Break.
- static bool Interactive_Watch (char *szCommand_)

  Interactive_Watch.
- static bool Interactive_ROM (char *szCommand_)

  Interactive_ROM.
- static bool Interactive_RAM (char *szCommand_)

  Interactive_RAM.

- static bool Interactive_EE (char *szCommand_)

  Interactive_EE.
- static bool Interactive_Registers (char *szCommand_)

  Interactive_Registers.
- static bool Interactive_Quit (char *szCommand_)

  Interactive_Quit.
- static bool Interactive_Help (char *szCommand_)

  Interactive_Help.
- static bool Interactive_Disasm (char *szCommand_)

  Interactive_Disasm.
- static bool Interactive_Trace (char *szCommand_)

  Interactive_Trace.
- static bool **Interactive_Execute_i** (void)
- void Interactive_CheckAndExecute (void)

  Interactive_CheckAndExecute.
- void Interactive_Set (void)

  Interactive_Set.
- void Interactive_Init (TraceBuffer_t *pstTrace_)

  Interactive_Init.
- static bool **Token_ScanNext** (char *szCommand_, int iStart_, int *piTokenStart_, int *piTokenLen_)
- static bool **Token_DiscardNext** (char *szCommand_, int iStart_, int *piNextTokenStart_)
- static bool **Token_ReadNextHex** (char *szCommand_, int iStart_, int *piNextTokenStart_, unsigned int *puiVal_)

Variables

- static bool bIsInteractive

  "true" when interactive debugger is running
- static bool bRetrigger

  "true" when the debugger needs to be enabled on the next cycle
- static TraceBuffer_t * pstTrace = 0

  Pointer to a tracebuffer object used for printing CPU execution trace.
- static Interactive_Command_t **astCommands** []

4.61.1 Detailed Description

Interactive debugging support. Provides mechanim for debugging a virtual AVR microcontroller with a variety of functionality common to external debuggers, such as GDB.

Definition in file interactive.c.

4.61.2 Typedef Documentation

4.61.2.1 typedef bool(* Interactive_Handler)(char *szCommand_)

Function pointer type used to implement interactive command handlers.

szCommand_ is a pointer to a string of command-line data entered from the debug console. returns a boolean value of "true" if executing this command should cause the parser to exit interactive mode.

Definition at line 44 of file interactive.c.

4.61.3 Function Documentation

4.61.3.1 static bool Interactive_Break (char * *szCommand_*) [static]

Interactive_Break.

Inserts a CPU breakpoint at a hex-address specified in the commandline

Parameters

*szCommand_*	command-line data passed in by the user.

Returns

false - continue interactive debugging

Definition at line 402 of file interactive.c.

4.61.3.2 void Interactive_CheckAndExecute (void)

Interactive_CheckAndExecute.

Wait for feedback and execute if running interactive. Otherwise, continue execution without waiting.

Definition at line 285 of file interactive.c.

4.61.3.3 static bool Interactive_Continue (char * *szCommand_*) [static]

Interactive_Continue.

Handler function used to implement the debugger's "continue" function, which exits interactive mode until the next breakpoint or watchpoint is hit.

Parameters

*szCommand_*	commnd-line data passed in by the user

Returns

true - exit interactive debugging

Definition at line 394 of file interactive.c.

4.61.3.4 static bool Interactive_Disasm (char * *szCommand_*) [static]

Interactive_Disasm.

Show the disassembly for the CPU's current opcode on the console.

Parameters

*szCommand_*	command-line data passed in by the user.

Returns

false - continue interactive debugging

Definition at line 570 of file interactive.c.

4.61.3.5 static bool Interactive_EE (char * *szCommand_*) [static]

Interactive_EE.

Display the contents of EEPROM (hex address, hex words) on the console

Parameters

*szCommand_*	command-line data passed in by the user.

Returns

false - continue interactive debugging

Definition at line 510 of file interactive.c.

4.61.3.6 static bool Interactive_Help (char * *szCommand_*) [static]

Interactive_Help.

Display the interactive help menu, listing available debugger commands on the console.

Parameters

*szCommand_*	command-line data passed in by the user.

Returns

false - continue interactive debugging

Definition at line 557 of file interactive.c.

4.61.3.7 void Interactive_Init (TraceBuffer_t * *pstTrace_*)

Interactive_Init.

Initialize the interactive debugger session for the given CPU struct and associated debug data

Parameters

*pstTrace_*	Pointer to the tracebuffer object

Definition at line 312 of file interactive.c.

4.61.3.8 static bool Interactive_Quit (char * *szCommand_*) [static]

Interactive_Quit.

Stop debugging, and exit flAVR.

Parameters

*szCommand_*	command-line data passed in by the user.

Returns

N/A - does not return (program terminates)

Definition at line 544 of file interactive.c.

4.61.3.9 static bool Interactive_RAM (char * *szCommand_*) [static]

Interactive_RAM.

Display the contents of RAM (hex address, hex words) on the console

Parameters

*szCommand_*	command-line data passed in by the user.

Returns

false - continue interactive debugging

Definition at line 483 of file interactive.c.

4.61.3.10 static bool Interactive_Registers (char * *szCommand_*) [static]

Interactive_Registers.

Display the contents of the core CPU registers on the console

Parameters

*szCommand_*	command-line data passed in by the user.

Returns

false - continue interactive debugging

Definition at line 537 of file interactive.c.

4.61.3.11 static bool Interactive_ROM (char * *szCommand_*) [static]

Interactive_ROM.

Display the contents of ROM (hex address, hex words) on the console

Parameters

*szCommand_*	command-line data passed in by the user.

Returns

false - continue interactive debugging

Definition at line 456 of file interactive.c.

4.61.3.12 void Interactive_Set (void)

Interactive_Set.

Enable interactive-debug mode

Definition at line 305 of file interactive.c.

4.61.3.13 static bool Interactive_Step (char * *szCommand_*) [static]

Interactive_Step.

Cause the debugger to step to the next CPU instruction and return back to the debug console for further input.

Parameters

*szCommand_*	commnd-line data passed in by the user

Returns

true - exit interactive debugging

Definition at line 550 of file interactive.c.

4.61.3.14 static bool Interactive_Trace (char * *szCommand_*) [static]

Interactive_Trace.

Dump the contents of the simulator's tracebuffer to the command-line

Parameters

*szCommand_*	command-line data passed in by the user.

Returns

false - continue interactive debugging

Definition at line 580 of file interactive.c.

4.61.3.15 static bool Interactive_Watch (char * *szCommand_*) [static]

Interactive_Watch.

Insert a CPU data watchpoint at a hex-address specified in the commandline

Parameters

*szCommand_*	command-line data passed in by the user.

Returns

false - continue interactive debugging

Definition at line 430 of file interactive.c.

4.61.4 Variable Documentation

4.61.4.1 Interactive_Command_t astCommands[] [static]

Initial value:

```
=
{
    { "registers","Dump registers to console",  Interactive_Registers },
    { "continue", "continue execution", Interactive_Continue },
    { "disasm",   "show disassembly", Interactive_Disasm },
    { "trace",    "Dump tracebuffer to console", Interactive_Trace},
    { "break",    "toggle breakpoint at address",  Interactive_Break },
    { "watch",    "toggle watchpoint at address",  Interactive_Watch },
    { "help",     "List commands", Interactive_Help },
    { "step",     "Step to next instruction", Interactive_Step },
    { "quit",     "Quit emulator", Interactive_Quit },
    { "reg",      "Dump registers to console",  Interactive_Registers },
    { "rom",      "Dump x bytes of ROM to console", Interactive_ROM },
    { "ram",      "Dump x bytes of RAM to console", Interactive_RAM },
    { "ee",       "Dump x bytes of RAM to console", Interactive_EE },
```

```
    { "b",        "toggle breakpoint at address",  Interactive_Break },
    { "c",        "continue execution", Interactive_Continue },
    { "d",        "show disassembly", Interactive_Disasm },
    { "w",        "toggle watchpoint at address",  Interactive_Watch },
    { "q",        "Quit emulator", Interactive_Quit },
    { "s",        "Step to next instruction", Interactive_Step },
    { "t",        "Dump tracebuffer to console", Interactive_Trace},
    { "h",        "List commands", Interactive_Help },
    { 0 }
}
```

Definition at line 200 of file interactive.c.

4.62 interactive.c

```
00001 /*****************************************************************************
00002  *      (     (                        (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\      (   (  (()/(   | -- [ Funkenstein ] ------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] -------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                         | "Yeah, it does Arduino..."
00011  * ----------------------------------------+-------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00023 #include "emu_config.h"
00024 #include "avr_cpu.h"
00025 #include "avr_cpu_print.h"
00026 #include "watchpoint.h"
00027 #include "breakpoint.h"
00028 #include "avr_disasm.h"
00029 #include "trace_buffer.h"
00030
00031 #include <stdint.h>
00032 #include <stdio.h>
00033 #include <stdlib.h>
00034 #include <string.h>
00035
00036 //------------------------------------------------------------------------------
00044 typedef bool (*Interactive_Handler)( char *szCommand_ );
00045
00046 //------------------------------------------------------------------------------
00050 typedef struct
00051 {
00052     const char *szCommand;
00053     const char *szDescription;
00054     Interactive_Handler pfHandler;
00055 } Interactive_Command_t;
00056
00057 //------------------------------------------------------------------------------
00058 static bool bIsInteractive;
00059 static bool bRetrigger;
00060
00061 static TraceBuffer_t *pstTrace = 0;
00062
00063 //------------------------------------------------------------------------------
00073 static bool Interactive_Continue( char *szCommand_ );
00074
00075 //------------------------------------------------------------------------------
00085 static bool Interactive_Step( char *szCommand_ );
00086
00087 //------------------------------------------------------------------------------
00096 static bool Interactive_Break( char *szCommand_ );
00097
00098 //------------------------------------------------------------------------------
00107 static bool Interactive_Watch( char *szCommand_ );
00108
00109 //------------------------------------------------------------------------------
00118 static bool Interactive_ROM( char *szCommand_ );
00119
00120 //------------------------------------------------------------------------------
00129 static bool Interactive_RAM( char *szCommand_ );
00130
00131 //------------------------------------------------------------------------------
00140 static bool Interactive_EE( char *szCommand_ );
00141
00142 //------------------------------------------------------------------------------
00151 static bool Interactive_Registers( char *szCommand_ );
```

```
00152
00153 //-------------------------------------------------------------------------
00162 static bool Interactive_Quit( char *szCommand_ );
00163
00164 //-------------------------------------------------------------------------
00174 static bool Interactive_Help( char *szCommand_ );
00175
00176 //-------------------------------------------------------------------------
00185 static bool Interactive_Disasm( char *szCommand_ );
00186
00187 //-------------------------------------------------------------------------
00196 static bool Interactive_Trace( char *szCommand_ );
00197
00198 //-------------------------------------------------------------------------
00199 // Command-handler table
00200 static Interactive_Command_t astCommands[] =
00201 {
00202     { "registers","Dump registers to console",  Interactive_Registers },
00203     { "continue", "continue execution", Interactive_Continue },
00204     { "disasm",   "show disassembly", Interactive_Disasm },
00205     { "trace",    "Dump tracebuffer to console", Interactive_Trace},
00206     { "break",    "toggle breakpoint at address",  Interactive_Break },
00207     { "watch",    "toggle watchpoint at address",  Interactive_Watch },
00208     { "help",     "List commands", Interactive_Help },
00209     { "step",     "Step to next instruction", Interactive_Step },
00210     { "quit",     "Quit emulator", Interactive_Quit },
00211     { "reg",      "Dump registers to console",  Interactive_Registers },
00212     { "rom",      "Dump x bytes of ROM to console", Interactive_ROM },
00213     { "ram",      "Dump x bytes of RAM to console", Interactive_RAM },
00214     { "ee",       "Dump x bytes of RAM to console", Interactive_EE },
00215     { "b",        "toggle breakpoint at address",  Interactive_Break },
00216     { "c",        "continue execution", Interactive_Continue },
00217     { "d",        "show disassembly", Interactive_Disasm },
00218     { "w",        "toggle watchpoint at address",  Interactive_Watch },
00219     { "q",        "Quit emulator", Interactive_Quit },
00220     { "s",        "Step to next instruction", Interactive_Step },
00221     { "t",        "Dump tracebuffer to console", Interactive_Trace},
00222     { "h",        "List commands", Interactive_Help },
00223     { 0 }
00224 };
00225
00226 //-------------------------------------------------------------------------
00227 static bool Interactive_Execute_i( void )
00228 {
00229     // Interactive mode - grab a line from standard input.
00230     char szCmdBuf[256];
00231     int iCmd = 0;
00232
00233     printf( "> " );
00234
00235     // Bail if stdin reaches EOF...
00236     if (0 == fgets(szCmdBuf, 255, stdin))
00237     {
00238         printf("[EOF]\n");
00239         exit(0);
00240     }
00241
00242     iCmd = strlen(szCmdBuf);
00243     if ( iCmd <= 1 )
00244     {
00245         printf("\n");
00246         iCmd = 0;
00247     }
00248     else
00249     {
00250         szCmdBuf[ iCmd - 1 ] = 0;
00251     }
00252
00253     // Compare command w/elements in the command table
00254     Interactive_Command_t *pstCommand = astCommands;
00255     bool bFound = false;
00256     bool bContinue = false;
00257
00258     while (pstCommand->szCommand)
00259     {
00260         if ( (0 == strncmp(pstCommand->szCommand, szCmdBuf, strlen(pstCommand->
      szCommand)))
00261                 && ( szCmdBuf[ strlen(pstCommand->szCommand) ] == ' ' ||
00262                      szCmdBuf[ strlen(pstCommand->szCommand) ] == '\0' ||
00263                      szCmdBuf[ strlen(pstCommand->szCommand) ] == '\n' ||
00264                      szCmdBuf[ strlen(pstCommand->szCommand) ] == '\r' ) )
00265         {
00266
00267             // printf( "Found match: %s\n", pstCommand->szCommand );
00268             bFound = true;
00269             bContinue = pstCommand->pfHandler( szCmdBuf );
00270             break;
```

```
00271         }
00272         // Next command
00273         pstCommand++;
00274     }
00275
00276     if (!bFound)
00277     {
00278         printf( "Invalid Command\n");
00279     }
00280
00281     return bContinue;
00282 }
00283
00284 //-------------------------------------------------------------------------
00285 void Interactive_CheckAndExecute( void )
00286 {
00287     // If we're in non-interactive mode (i.e. native execution), then return
00288     // out instantly.
00289     if (false == bIsInteractive)
00290     {
00291         if (false == bRetrigger)
00292         {
00293             return;
00294         }
00295         bIsInteractive = true;
00296         bRetrigger = false;
00297     }
00298     printf( "Debugging @ Address [0x%X]\n", stCPU.u16PC );
00299
00300     // Keep attempting to parse commands until a valid one was encountered
00301     while (!Interactive_Execute_i()) { /* Do Nothing */ }
00302 }
00303
00304 //-------------------------------------------------------------------------
00305 void Interactive_Set( void )
00306 {
00307     bIsInteractive = true;
00308     bRetrigger = false;
00309 }
00310
00311 //-------------------------------------------------------------------------
00312 void Interactive_Init( TraceBuffer_t *pstTrace_ )
00313 {
00314     pstTrace = pstTrace_;
00315     bIsInteractive = false;
00316     bRetrigger = false;
00317 }
00318
00319 //-------------------------------------------------------------------------
00320 static bool Token_ScanNext( char *szCommand_, int iStart_, int *piTokenStart_, int *piTokenLen_)
00321 {
00322     int i = iStart_;
00323
00324     // Parse leading whitespace
00325     while ( (szCommand_[i] == ' ') ||
00326             (szCommand_[i] == '\t') ||
00327             (szCommand_[i] == '\r') ||
00328             (szCommand_[i] == '\n')
00329             ) { i++; }
00330
00331     // Check null termination
00332     if (szCommand_[i] == '\0' )
00333     {
00334         return false;
00335     }
00336
00337     // Parse token
00338     *piTokenStart_ = i;
00339     while ( (szCommand_[i] != ' ') &&
00340             (szCommand_[i] != '\t') &&
00341             (szCommand_[i] != '\r') &&
00342             (szCommand_[i] != '\n') &&
00343             (szCommand_[i] != '\0')
00344             ) { i++; }
00345     *piTokenLen_ = (i - *piTokenStart_);
00346
00347     // printf( "Start, Len: %d, %d\n", i, *piTokenLen_ );
00348     return true;
00349 }
00350
00351 //-------------------------------------------------------------------------
00352 static bool Token_DiscardNext( char *szCommand_, int iStart_, int *piNextTokenStart_ )
00353 {
00354     int iTempStart;
00355     int iTempLen;
00356     if (!Token_ScanNext(szCommand_, iStart_, &iTempStart, &iTempLen ))
00357     {
```

```
00358         return false;
00359     }
00360     *piNextTokenStart_ = iTempStart + iTempLen + 1;
00361     return true;
00362 }
00363
00364 //-------------------------------------------------------------------------
00365 static bool Token_ReadNextHex( char *szCommand_, int iStart_, int *piNextTokenStart_, unsigned int *puiVal_
      )
00366 {
00367     int iTempStart = iStart_;
00368     int iTempLen;
00369
00370     if (!Token_ScanNext(szCommand_, iStart_, &iTempStart, &iTempLen ))
00371     {
00372         return false;
00373     }
00374
00375     szCommand_[iTempStart + iTempLen] = 0;
00376
00377     if (0 == sscanf( &szCommand_[iTempStart], "%x", puiVal_ ))
00378     {
00379         if (0 == sscanf( &szCommand_[iTempStart], "x%x", puiVal_ ))
00380         {
00381             if (0 == sscanf( &szCommand_[iTempStart], "0x%x", puiVal_ ))
00382             {
00383                 printf( "Missing Argument\n" );
00384                 return false;
00385             }
00386         }
00387     }
00388
00389     *piNextTokenStart_ = iTempStart + iTempLen + 1;
00390     return true;
00391 }
00392
00393 //-------------------------------------------------------------------------
00394 static bool Interactive_Continue( char *szCommand_ )
00395 {
00396     bIsInteractive = false;
00397     bRetrigger = false;
00398     return true;
00399 }
00400
00401 //-------------------------------------------------------------------------
00402 static bool Interactive_Break( char *szCommand_ )
00403 {
00404     unsigned int uiAddr;
00405     int iTokenStart;
00406
00407     if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
00408     {
00409         return false;
00410     }
00411
00412     if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiAddr))
00413     {
00414         return false;
00415     }
00416
00417     if (BreakPoint_EnabledAtAddress( (uint16_t)uiAddr))
00418     {
00419         BreakPoint_Delete( (uint16_t)uiAddr);
00420     }
00421     else
00422     {
00423         BreakPoint_Insert( (uint16_t)uiAddr);
00424     }
00425
00426     return false;
00427 }
00428
00429 //-------------------------------------------------------------------------
00430 static bool Interactive_Watch( char *szCommand_ )
00431 {
00432     unsigned int uiAddr;
00433     int iTokenStart;
00434
00435     if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
00436     {
00437         return false;
00438     }
00439
00440     if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiAddr))
00441     {
00442         return false;
00443     }
```

```
00444
00445     if (WatchPoint_EnabledAtAddress((uint16_t)uiAddr))
00446     {
00447         WatchPoint_Delete( (uint16_t)uiAddr);
00448     }
00449     else
00450     {
00451         WatchPoint_Insert( (uint16_t)uiAddr);
00452     }
00453     return false;
00454 }
00455 //-----------------------------------------------------------------------------
00456 static bool Interactive_ROM( char *szCommand_ )
00457 {
00458     unsigned int uiAddr;
00459     unsigned int uiLen;
00460     int iTokenStart;
00461
00462     if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
00463     {
00464         return false;
00465     }
00466
00467     if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiAddr))
00468     {
00469         return false;
00470     }
00471
00472     if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiLen))
00473     {
00474         return false;
00475     }
00476
00477     print_rom( (uint16_t)uiAddr, (uint16_t)uiLen );
00478
00479     return false;
00480 }
00481
00482 //-----------------------------------------------------------------------------
00483 static bool Interactive_RAM( char *szCommand_ )
00484 {
00485     unsigned int uiAddr;
00486     unsigned int uiLen;
00487     int iTokenStart;
00488
00489     if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
00490     {
00491         return false;
00492     }
00493
00494     if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiAddr))
00495     {
00496         return false;
00497     }
00498
00499     if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiLen))
00500     {
00501         return false;
00502     }
00503
00504     print_ram( (uint16_t)uiAddr, (uint16_t)uiLen );
00505
00506     return false;
00507 }
00508
00509 //-----------------------------------------------------------------------------
00510 static bool Interactive_EE( char *szCommand_ )
00511 {
00512     unsigned int uiAddr;
00513     unsigned int uiLen;
00514     int iTokenStart;
00515
00516     if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
00517     {
00518         return false;
00519     }
00520
00521     if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiAddr))
00522     {
00523         return false;
00524     }
00525
00526     if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiLen))
00527     {
00528         return false;
00529     }
00530
```

```
00531     printf( "Dump EEPROM [%x:%x]\n", uiAddr, uiLen );
00532
00533     return false;
00534 }
00535
00536 //-----------------------------------------------------------------------------
00537 static bool Interactive_Registers( char *szCommand_ )
00538 {
00539     print_core_regs();
00540     return false;
00541 }
00542
00543 //-----------------------------------------------------------------------------
00544 static bool Interactive_Quit( char *szCommand_ )
00545 {
00546     exit(0);
00547 }
00548
00549 //-----------------------------------------------------------------------------
00550 static bool Interactive_Step( char *szCommand_ )
00551 {
00552     bRetrigger = true; // retrigger debugging on next loop
00553     return true;
00554 }
00555
00556 //-----------------------------------------------------------------------------
00557 static bool Interactive_Help( char *szCommand_ )
00558 {
00559     Interactive_Command_t *pstCommand_ = astCommands;
00560     printf( "FLAVR interactive debugger commands:\n");
00561     while (pstCommand_->szCommand)
00562     {
00563         printf( "    %s: %s\n", pstCommand_->szCommand, pstCommand_->
    szDescription );
00564         pstCommand_++;
00565     }
00566     return false;
00567 }
00568
00569 //-----------------------------------------------------------------------------
00570 static bool Interactive_Disasm( char *szCommand_ )
00571 {
00572     uint16_t OP = stCPU.pu16ROM[stCPU.u16PC];
00573     printf("0x%04X: [0x%04X] ", stCPU.u16PC, OP);
00574     AVR_Decode(OP);
00575     AVR_Disasm_Function(OP)();
00576     return false;
00577 }
00578
00579 //-----------------------------------------------------------------------------
00580 static bool Interactive_Trace( char *szCommand_ )
00581 {
00582     TraceBuffer_Print( pstTrace, TRACE_PRINT_COMPACT | TRACE_PRINT_DISASSEMBLY );
00583     return false;
00584 }
```

4.63 interactive.h File Reference

Interactive debugging support.

```
#include "emu_config.h"
#include "avr_cpu.h"
#include "trace_buffer.h"
```

Functions

- void Interactive_CheckAndExecute (void)

  Interactive_CheckAndExecute.

- void Interactive_Set (void)

  Interactive_Set.

- void Interactive_Init (TraceBuffer_t *pstTrace_)

  Interactive_Init.

4.63.1 Detailed Description

Interactive debugging support.

Definition in file interactive.h.

4.63.2 Function Documentation

4.63.2.1 void Interactive_CheckAndExecute (void)

Interactive_CheckAndExecute.

Wait for feedback and execute if running interactive. Otherwise, continue execution without waiting.

Definition at line 285 of file interactive.c.

4.63.2.2 void Interactive_Init (TraceBuffer_t * *pstTrace_*)

Interactive_Init.

Initialize the interactive debugger session for the given CPU struct and associated debug data

Parameters

*pstTrace_*	Pointer to the tracebuffer object

Definition at line 312 of file interactive.c.

4.63.2.3 void Interactive_Set (void)

Interactive_Set.

Enable interactive-debug mode

Definition at line 305 of file interactive.c.

4.64 interactive.h

```
00001 /*****************************************************************************
00002  *      (     (                       (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\     (   (  (()/(   | -- [ Funkenstein ] ------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\   |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+-----------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #ifndef __INTERACTIVE_H__
00022 #define __INTERACTIVE_H__
00023
00024 #include "emu_config.h"
00025 #include "avr_cpu.h"
00026 #include "trace_buffer.h"
00027
00028 //-----------------------------------------------------------------------------
00035 void Interactive_CheckAndExecute( void );
00036
00037 //-----------------------------------------------------------------------------
00043 void Interactive_Set( void );
00044
00045 //-----------------------------------------------------------------------------
00054 void Interactive_Init( TraceBuffer_t *pstTrace_);
00055
00056 #endif
```

4.65 mega_eint.c File Reference

ATMega External Interrupt Implementation.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "avr_cpu.h"
#include "avr_peripheral.h"
#include "avr_periphregs.h"
#include "avr_interrupt.h"
```

Macros

- #define **DEBUG_PRINT**(...)

Enumerations

- enum InterruptSense_t { INT_SENSE_LOW = 0, INT_SENSE_CHANGE, INT_SENSE_FALL, INT_SENSE_RISE }

Functions

- static void **EINT_AckInt** (uint8_t ucVector_)
- static void **EINT_Init** (void *context_)
- static void **EINT_Read** (void *context_, uint8_t ucAddr_, uint8_t *pucValue_)
- static void **EICRA_Write** (uint8_t ucValue_)
- static void **EIFR_Write** (uint8_t ucValue_)
- static void **EIMSK_Write** (uint8_t ucValue_)
- static void **EINT_Write** (void *context_, uint8_t ucAddr_, uint8_t ucValue_)
- static void EINT_Clock (void *context_)

Variables

- static InterruptSense_t **eINT0Sense**
- static InterruptSense_t **eINT1Sense**
- static uint8_t **ucLastINT0**
- static uint8_t **ucLastINT1**
- AVRPeripheral **stEINT_a**
- AVRPeripheral **stEINT_b**

4.65.1 Detailed Description

ATMega External Interrupt Implementation.

Definition in file mega_eint.c.

4.65.2 Enumeration Type Documentation

4.65.2.1 enum InterruptSense_t

Enumerator

INT_SENSE_LOW Logic low triggers interrupt.

INT_SENSE_CHANGE Change in state triggers interrupt.

INT_SENSE_FALL Falling edge triggers interrupt.

INT_SENSE_RISE Rising edge triggers interrupt.

Definition at line 32 of file mega_eint.c.

4.65.3 Function Documentation

4.65.3.1 static void EINT_Clock (void * *context_*) [static]

! ToDo - Consider adding support for external stimulus (which would ! Invoke inputs on PIND as opposed to PORTD)... This will only work ! as software interrupts in its current state

Definition at line 169 of file mega_eint.c.

4.65.4 Variable Documentation

4.65.4.1 AVRPeripheral stEINT_a

Initial value:

```
=
{
    EINT_Init,
    EINT_Read,
    EINT_Write,
    EINT_Clock,
    NULL,
    0x69,
    0x69
}
```

Definition at line 282 of file mega_eint.c.

4.65.4.2 AVRPeripheral stEINT_b

Initial value:

```
=
{
    NULL,
    EINT_Read,
    EINT_Write,
    NULL,
    NULL,
    0x3C,
    0x3D
}
```

Definition at line 294 of file mega_eint.c.

4.66 mega_eint.c

```
00001 /*****************************************************************************
00002  *      (     (                          (      |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(    )\      (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\   |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #include <stdio.h>
00022 #include <stdlib.h>
00023 #include <string.h>
00024 #include "avr_cpu.h"
00025 #include "avr_peripheral.h"
00026 #include "avr_periphregs.h"
00027 #include "avr_interrupt.h"
00028
00029 #define DEBUG_PRINT(...)
00030
00031 //-----------------------------------------------------------------------------
00032 typedef enum
00033 {
00034     INT_SENSE_LOW = 0,
00035     INT_SENSE_CHANGE,
00036     INT_SENSE_FALL,
00037     INT_SENSE_RISE
00038 } InterruptSense_t;
00039
00040
00041 //-----------------------------------------------------------------------------
00042 static InterruptSense_t eINT0Sense;
00043 static InterruptSense_t eINT1Sense;
00044 static uint8_t ucLastINT0;
00045 static uint8_t ucLastINT1;
00046
00047 //-----------------------------------------------------------------------------
00048 static void EINT_AckInt(  uint8_t ucVector_);
00049
00050 //-----------------------------------------------------------------------------
00051 static void EINT_Init(void *context_ )
00052 {
00053     eINT0Sense = INT_SENSE_LOW;
00054     eINT1Sense = INT_SENSE_LOW;
00055     ucLastINT0 = 0;
00056     ucLastINT1 = 0;
00057
00058     // Register interrupt callback functions
00059     CPU_RegisterInterruptCallback(EINT_AckInt, 0x01);
00060     CPU_RegisterInterruptCallback(EINT_AckInt, 0x02);
00061 }
00062
00063 //-----------------------------------------------------------------------------
00064 static void EINT_Read(void *context_, uint8_t ucAddr_, uint8_t *pucValue_ )
00065 {
00066     *pucValue_ = stCPU.pstRAM->au8RAM[ucAddr_];
00067 }
00068
00069 //-----------------------------------------------------------------------------
00070 static void EICRA_Write( uint8_t ucValue_ )
00071 {
00072     DEBUG_PRINT("EICRA Clock\n");
00073     ucValue_ &= 0x0F;    // Only the bottom 2 bits are valid.
00074     stCPU.pstRAM->stRegisters.EICRA.r = ucValue_;
00075
00076     // Change local interrupt sense value.
00077     if ((stCPU.pstRAM->stRegisters.EICRA.ISC00 == 0) &&
00078         (stCPU.pstRAM->stRegisters.EICRA.ISC01 == 0))
00079     {
00080         DEBUG_PRINT("I0-low\n");
00081         eINT0Sense = INT_SENSE_LOW;
00082     }
00083     else if ((stCPU.pstRAM->stRegisters.EICRA.ISC00 == 1) &&
00084              (stCPU.pstRAM->stRegisters.EICRA.ISC01 == 0))
00085     {
00086         DEBUG_PRINT("I0-change\n");
00087         eINT0Sense = INT_SENSE_CHANGE;
00088     }
00089     else if ((stCPU.pstRAM->stRegisters.EICRA.ISC00 == 0) &&
00090              (stCPU.pstRAM->stRegisters.EICRA.ISC01 == 1))
```

```
00091     {
00092         DEBUG_PRINT("I0-fall\n");
00093         eINT0Sense = INT_SENSE_FALL;
00094     }
00095     else if ((stCPU.pstRAM->stRegisters.EICRA.ISC00 == 1) &&
00096              (stCPU.pstRAM->stRegisters.EICRA.ISC01 == 1))
00097     {
00098         DEBUG_PRINT("I0-risel\n");
00099         eINT0Sense = INT_SENSE_RISE;
00100     }
00101
00102     if ((stCPU.pstRAM->stRegisters.EICRA.ISC10 == 0) &&
00103         (stCPU.pstRAM->stRegisters.EICRA.ISC11 == 0))
00104     {
00105         eINT1Sense = INT_SENSE_LOW;
00106     }
00107     else if ((stCPU.pstRAM->stRegisters.EICRA.ISC10 == 1) &&
00108              (stCPU.pstRAM->stRegisters.EICRA.ISC11 == 0))
00109     {
00110         eINT1Sense = INT_SENSE_CHANGE;
00111     }
00112     else if ((stCPU.pstRAM->stRegisters.EICRA.ISC10 == 0) &&
00113              (stCPU.pstRAM->stRegisters.EICRA.ISC11 == 1))
00114     {
00115         eINT1Sense = INT_SENSE_RISE;
00116     }
00117     else if ((stCPU.pstRAM->stRegisters.EICRA.ISC10 == 1) &&
00118              (stCPU.pstRAM->stRegisters.EICRA.ISC11 == 1))
00119     {
00120         eINT1Sense = INT_SENSE_FALL;
00121     }
00122     DEBUG_PRINT ("IntSense0,1: %d, %d\n", eINT0Sense, eINT1Sense);
00123     DEBUG_PRINT ("EICRA: %d, ISC00 : %d, ISC01 : %d, ISC10: %d, ISC11: %d\n",
00124                  stCPU.pstRAM->stRegisters.EICRA.r,
00125                  stCPU.pstRAM->stRegisters.EICRA.ISC00,
00126                  stCPU.pstRAM->stRegisters.EICRA.ISC01,
00127                  stCPU.pstRAM->stRegisters.EICRA.ISC10,
00128                  stCPU.pstRAM->stRegisters.EICRA.ISC11
00129             );
00130 }
00131
00132 //-----------------------------------------------------------------------------
00133 static void EIFR_Write( uint8_t ucValue_ )
00134 {
00135     DEBUG_PRINT("EIFR Clock\n");
00136     ucValue_ &= 0x03;   // Only the bottom-2 bits are set
00137     stCPU.pstRAM->stRegisters.EIFR.r = ucValue_;
00138 }
00139
00140 //-----------------------------------------------------------------------------
00141 static void EIMSK_Write( uint8_t ucValue_ )
00142 {
00143     DEBUG_PRINT("EIMSK Write\n");
00144     ucValue_ &= 0x03;   // Only the bottom-2 bits are set
00145     stCPU.pstRAM->stRegisters.EIMSK.r = ucValue_;
00146 }
00147
00148 //-----------------------------------------------------------------------------
00149 static void EINT_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
00150 {
00151     DEBUG_PRINT("EINT Write\n");
00152     switch (ucAddr_)
00153     {
00154     case 0x69:  // EICRA
00155         EICRA_Write(ucValue_);
00156         break;
00157     case 0x3C:  // EIFR
00158         EIFR_Write(ucValue_);
00159         break;
00160     case 0x3D:  // EIMSK
00161         EIMSK_Write(ucValue_);
00162         break;
00163     default:
00164         break;
00165     }
00166 }
00167
00168 //-----------------------------------------------------------------------------
00169 static void EINT_Clock(void *context_ )
00170 {
00171     // Check to see if interrupts are enabled.  If so, check to see if the
00172     // interrupt mask is set, and then finally - whether or not an interupt
00173     // condition has occurred based on the interrupt sense mode.
00174     bool bSetINT0 = false;
00175     bool bSetINT1 = false;
00176
00180
```

```
00181     if (stCPU.pstRAM->stRegisters.EIMSK.INT0 == 1)
00182     {
00183         switch (eINT0Sense)
00184         {
00185         case INT_SENSE_LOW:
00186             if (stCPU.pstRAM->stRegisters.PORTD.PORT2 == 0)
00187             {
00188                 DEBUG_PRINT(" SET INT0\n");
00189                 bSetINT0 = true;
00190             }
00191             break;
00192         case INT_SENSE_CHANGE:
00193             if (stCPU.pstRAM->stRegisters.PORTD.PORT2 != ucLastINT0)
00194             {
00195                 DEBUG_PRINT(" SET INT0\n");
00196                 bSetINT0 = true;
00197             }
00198             break;
00199         case INT_SENSE_FALL:
00200             if ((stCPU.pstRAM->stRegisters.PORTD.PORT2 == 0) && (ucLastINT0 == 1))
00201             {
00202                 DEBUG_PRINT(" SET INT0\n");
00203                 bSetINT0 = true;
00204             }
00205             break;
00206         case INT_SENSE_RISE:
00207             if ((stCPU.pstRAM->stRegisters.PORTD.PORT2 == 1) && (ucLastINT0 == 0))
00208             {
00209                 DEBUG_PRINT(" SET INT0\n");
00210                 bSetINT0 = true;
00211             }
00212             break;
00213         }
00214     }
00215     if (stCPU.pstRAM->stRegisters.EIMSK.INT1 == 1)
00216     {
00217         switch (eINT0Sense)
00218         {
00219         case INT_SENSE_LOW:
00220             if (stCPU.pstRAM->stRegisters.PORTD.PORT3 == 0)
00221             {
00222                 bSetINT1 = true;
00223             }
00224             break;
00225         case INT_SENSE_CHANGE:
00226             if (stCPU.pstRAM->stRegisters.PORTD.PORT3 != ucLastINT1)
00227             {
00228                 bSetINT1 = true;
00229             }
00230             break;
00231         case INT_SENSE_FALL:
00232             if ((stCPU.pstRAM->stRegisters.PORTD.PORT3 == 0) && (ucLastINT1 == 1))
00233             {
00234                 bSetINT1 = true;
00235             }
00236             break;
00237         case INT_SENSE_RISE:
00238             if ((stCPU.pstRAM->stRegisters.PORTD.PORT3 == 1) && (ucLastINT1 == 0))
00239             {
00240                 bSetINT1 = true;
00241             }
00242             break;
00243         }
00244     }
00245
00246     // Trigger interrupts where necessary
00247     if (bSetINT0)
00248     {
00249         stCPU.pstRAM->stRegisters.EIFR.INTF0 = 1;
00250         AVR_InterruptCandidate(0x01);
00251     }
00252     if (bSetINT1)
00253     {
00254         stCPU.pstRAM->stRegisters.EIFR.INTF1 = 1;
00255         AVR_InterruptCandidate(0x02);
00256
00257     }
00258     // Update locally-cached copy of previous INT0/INT1 pin status.
00259     ucLastINT0 = stCPU.pstRAM->stRegisters.PORTD.PORT2;
00260     ucLastINT1 = stCPU.pstRAM->stRegisters.PORTD.PORT3;
00261 }
00262
00263 //-------------------------------------------------------------------------
00264 static void EINT_AckInt(  uint8_t ucVector_)
00265 {
00266     // We automatically clear the INTx flag as soon as the interrupt
00267     // is acknowledged.
```

```
00268     switch (ucVector_)
00269     {
00270     case 0x01:
00271         DEBUG_PRINT("INT0!\n");
00272         stCPU.pstRAM->stRegisters.EIFR.INTF0 = 0;
00273         break;
00274     case 0x02:
00275         DEBUG_PRINT("INT1!\n");
00276         stCPU.pstRAM->stRegisters.EIFR.INTF1 = 0;
00277         break;
00278     }
00279 }
00280
00281 //-------------------------------------------------------------------------
00282 AVRPeripheral stEINT_a =
00283 {
00284     EINT_Init,
00285     EINT_Read,
00286     EINT_Write,
00287     EINT_Clock,
00288     NULL,
00289     0x69,
00290     0x69
00291 };
00292
00293 //-------------------------------------------------------------------------
00294 AVRPeripheral stEINT_b =
00295 {
00296     NULL,
00297     EINT_Read,
00298     EINT_Write,
00299     NULL,
00300     NULL,
00301     0x3C,
00302     0x3D
00303 };
```

4.67 mega_eint.h File Reference

ATMega External Interrupt Implementation.

```
#include "avr_peripheral.h"
```

Variables

- AVRPeripheral **stEINT_a**
- AVRPeripheral **stEINT_b**

4.67.1 Detailed Description

ATMega External Interrupt Implementation.

Definition in file mega_eint.h.

4.68 mega_eint.h

```
00001 /*************************************************************************
00002  *      (     (                       (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\      (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ----------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] ------------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] ------------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
```

```
00014  ******************************************************************************/
00021 #ifndef __MEGA_EINT_H__
00022 #define __MEGA_EINT_H__
00023
00024 #include "avr_peripheral.h"
00025
00026 extern AVRPeripheral stEINT_a;
00027 extern AVRPeripheral stEINT_b;
00028
00029 #endif //__MEGA_EINT_H__
```

4.69 mega_timer16.c File Reference

ATMega 16-bit timer implementation.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "avr_cpu.h"
#include "avr_peripheral.h"
#include "avr_periphregs.h"
#include "avr_interrupt.h"
```

Macros

- #define **DEBUG_PRINT**(...)

Enumerations

- enum ClockSource_t {
  CLK_SRC_OFF, **CLK_SRC_DIV_1**, **CLK_SRC_DIV_8**, **CLK_SRC_DIV_64**,
  CLK_SRC_DIV_256, **CLK_SRC_DIV_1024**, **CLK_SRC_T1_FALL**, **CLK_SRC_T1_RISE**,
  CLK_SRC_OFF, **CLK_SRC_DIV_1**, **CLK_SRC_DIV_8**, **CLK_SRC_DIV_64**,
  CLK_SRC_DIV_256, **CLK_SRC_DIV_1024**, **CLK_SRC_T1_FALL**, **CLK_SRC_T1_RISE** }
- enum **WaveformGeneratorMode_t** {
  WGM_NORMAL, **WGM_PWM_PC_8BIT**, **WGM_PWM_PC_9BIT**, **WGM_PWM_PC_10BIT**,
  WGM_CTC_OCR, **WGM_PWM_8BIT**, **WGM_PWM_9BIT**, **WGM_PWM_10BIT**,
  WGM_PWM_PC_FC_ICR, **WGM_PWM_PC_FC_OCR**, **WGM_PWM_PC_ICR**, **WGM_PWM_PC_OCR**,
  WGM_CTC_ICR, **WGM_RESERVED**, **WGM_FAST_PWM_ICR**, **WGM_FAST_PWM_OCR**,
  WGM_NORMAL, **WGM_PWM_PC_FF**, **WGM_CTC_OCR**, **WGM_FAST_PWM_FF**,
  WGM_RESERVED_1, **WGM_PWM_PC_OCR**, **WGM_RESERVED_2**, **WGM_FAST_PWM_OCR** }
- enum **CompareOutputMode_t** {
  COM_NORMAL, **COM_TOGGLE_MATCH**, **COM_CLEAR_MATCH**, **COM_SET_MATCH**,
  COM_NORMAL, **COM_TOGGLE_MATCH**, **COM_CLEAR_MATCH**, **COM_SET_MATCH** }

Functions

- static void **TCNT1_Increment** ()
- static uint16_t **TCNT1_Read** ()
- static void **TCNT1_Clear** ()
- static uint16_t **OCR1A_Read** ()
- static uint16_t **OCR1B_Read** ()
- static uint16_t **ICR1_Read** ()
- static bool **Timer16_Is_TOIE1_Enabled** ()
- static bool **Timer16_Is_OCIE1A_Enabled** ()
- static bool **Timer16_Is_OCIE1B_Enabled** ()

- static bool **Timer16_Is_ICIE1_Enabled** ()
- static void **OV1_Ack** (uint8_t ucVector_)
- static void **IC1_Ack** (uint8_t ucVector_)
- static void **COMP1A_Ack** (uint8_t ucVector_)
- static void **COMP1B_Ack** (uint8_t ucVector_)
- static void **Timer16_Init** (void *context_)
- static void **Timer16_Read** (void *context_, uint8_t ucAddr_, uint8_t *pucValue_)
- static void **TCCR1A_Write** (uint8_t ucAddr_, uint8_t ucValue_)
- static void **TCCR1B_Write** (uint8_t ucAddr_, uint8_t ucValue_)
- static void **TCCR1C_Write** (uint8_t ucAddr_, uint8_t ucValue_)
- static void **TCNT1L_Write** (uint8_t ucAddr_, uint8_t ucValue_)
- static void **TCNT1H_Write** (uint8_t ucAddr_, uint8_t ucValue_)
- static void **ICR1L_Write** (uint8_t ucAddr_, uint8_t ucValue_)
- static void **ICR1H_Write** (uint8_t ucAddr_, uint8_t ucValue_)
- static void **OCR1AL_Write** (uint8_t ucAddr_, uint8_t ucValue_)
- static void **OCR1AH_Write** (uint8_t ucAddr_, uint8_t ucValue_)
- static void **OCR1BL_Write** (uint8_t ucAddr_, uint8_t ucValue_)
- static void **OCR1BH_Write** (uint8_t ucAddr_, uint8_t ucValue_)
- static void **Timer16_IntFlagUpdate** (void)
- static void **Timer16b_Write** (void *context_, uint8_t ucAddr_, uint8_t ucValue_)
- static void **Timer16_Write** (void *context_, uint8_t ucAddr_, uint8_t ucValue_)
- static void Timer16_Clock (void *context_)

Variables

- static uint16_t **u16DivCycles** = 0
- static uint16_t **u16DivRemain** = 0
- static ClockSource_t **eClockSource** = CLK_SRC_OFF
- static WaveformGeneratorMode_t **eWGM** = WGM_NORMAL
- static CompareOutputMode_t **eCOM1A** = COM_NORMAL
- static CompareOutputMode_t **eCOM1B** = COM_NORMAL
- static uint8_t **u8Temp**
- static uint16_t **u8Count**
- AVRPeripheral **stTimer16**
- AVRPeripheral **stTimer16a**
- AVRPeripheral **stTimer16b**

4.69.1 Detailed Description

ATMega 16-bit timer implementation.

Definition in file mega_timer16.c.

4.69.2 Enumeration Type Documentation

4.69.2.1 enum ClockSource_t

! This implementation only tracks the basic timer/capture/compare functionality of the peripheral, to match what's used in Mark3. Future considerations, TBD.

Definition at line 38 of file mega_timer16.c.

4.69.3 Function Documentation

4.69.3.1 static void Timer16_Clock (void * *context_*) [static]

! ToDo - Handle external timer generated events.

Definition at line 448 of file mega_timer16.c.

4.69.4 Variable Documentation

4.69.4.1 AVRPeripheral stTimer16

Initial value:

```
=
{
    Timer16_Init,
    Timer16_Read,
    Timer16_Write,
    Timer16_Clock,
    0,
    0x80,
    0x8B
}
```

Definition at line 580 of file mega_timer16.c.

4.69.4.2 AVRPeripheral stTimer16a

Initial value:

```
=
{
    0,
    Timer16_Read,
    Timer16b_Write,
    0,
    0,
    0x36,
    0x36
}
```

Definition at line 592 of file mega_timer16.c.

4.69.4.3 AVRPeripheral stTimer16b

Initial value:

```
=
{
    0,
    Timer16_Read,
    Timer16b_Write,
    0,
    0,
    0x6F,
    0x6F
}
```

Definition at line 604 of file mega_timer16.c.

4.70 mega_timer16.c

```
00001 /*****************************************************************************
```

```
00002  *     (     (                      (     |
00003  *    )\ )  )\ )    (              )\ )   |
00004  *   (()/( (()/(    )\      (   (  (()/(   | -- [ Funkenstein ] ------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\   |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+-------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  ******************************************************************************/
00021 #include <stdio.h>
00022 #include <stdlib.h>
00023 #include <string.h>
00024 #include "avr_cpu.h"
00025 #include "avr_peripheral.h"
00026 #include "avr_periphregs.h"
00027 #include "avr_interrupt.h"
00028
00029 #define DEBUG_PRINT(...)
00030
00031 //------------------------------------------------------------------------------
00035 //------------------------------------------------------------------------------
00036
00037 //------------------------------------------------------------------------------
00038 typedef enum
00039 {
00040     CLK_SRC_OFF,
00041     CLK_SRC_DIV_1,
00042     CLK_SRC_DIV_8,
00043     CLK_SRC_DIV_64,
00044     CLK_SRC_DIV_256,
00045     CLK_SRC_DIV_1024,
00046     CLK_SRC_T1_FALL,
00047     CLK_SRC_T1_RISE
00048 } ClockSource_t;
00049
00050 //------------------------------------------------------------------------------
00051 typedef enum
00052 {
00053     WGM_NORMAL,
00054     WGM_PWM_PC_8BIT,
00055     WGM_PWM_PC_9BIT,
00056     WGM_PWM_PC_10BIT,
00057     WGM_CTC_OCR,
00058     WGM_PWM_8BIT,
00059     WGM_PWM_9BIT,
00060     WGM_PWM_10BIT,
00061     WGM_PWM_PC_FC_ICR,
00062     WGM_PWM_PC_FC_OCR,
00063     WGM_PWM_PC_ICR,
00064     WGM_PWM_PC_OCR,
00065     WGM_CTC_ICR,
00066     WGM_RESERVED,
00067     WGM_FAST_PWM_ICR,
00068     WGM_FAST_PWM_OCR
00069 } WaveformGeneratorMode_t;
00070
00071 //------------------------------------------------------------------------------
00072 typedef enum
00073 {
00074     COM_NORMAL,         // OCA1/B disconnected
00075     COM_TOGGLE_MATCH,   // Toggle on match
00076     COM_CLEAR_MATCH,
00077     COM_SET_MATCH
00078 } CompareOutputMode_t;
00079
00080 //------------------------------------------------------------------------------
00081 static uint16_t u16DivCycles = 0;
00082 static uint16_t u16DivRemain = 0;
00083 static ClockSource_t eClockSource   = CLK_SRC_OFF;
00084 static WaveformGeneratorMode_t eWGM = WGM_NORMAL;
00085 static CompareOutputMode_t eCOM1A = COM_NORMAL;
00086 static CompareOutputMode_t eCOM1B = COM_NORMAL;
00087
00088 //------------------------------------------------------------------------------
00089 static uint8_t  u8Temp;  // The 8-bit temporary register used in 16-bit register accesses
00090 static uint16_t u8Count; // Internal 16-bit count register
00091
00092 //------------------------------------------------------------------------------
00093 static void TCNT1_Increment()
00094 {
00095     uint16_t u16NewVal = 0;
00096
00097     u16NewVal = (stCPU.pstRAM->stRegisters.TCNT1H << 8 ) |
```

```
00098                 stCPU.pstRAM->stRegisters.TCNT1L;
00099
00100     u16NewVal++;
00101     stCPU.pstRAM->stRegisters.TCNT1L = (u16NewVal & 0x00FF);
00102     stCPU.pstRAM->stRegisters.TCNT1H = (u16NewVal >> 8);
00103 }
00104
00105 //-----------------------------------------------------------------------
00106 static uint16_t TCNT1_Read()
00107 {
00108     uint16_t u16Ret = 0;
00109
00110     u16Ret = (stCPU.pstRAM->stRegisters.TCNT1H << 8 ) |
00111              stCPU.pstRAM->stRegisters.TCNT1L;
00112     return u16Ret;
00113 }
00114
00115 //-----------------------------------------------------------------------
00116 static void TCNT1_Clear()
00117 {
00118     stCPU.pstRAM->stRegisters.TCNT1H = 0;
00119     stCPU.pstRAM->stRegisters.TCNT1L = 0;
00120 }
00121
00122 //-----------------------------------------------------------------------
00123 static uint16_t OCR1A_Read()
00124 {
00125     uint16_t u16Ret = 0;
00126
00127     u16Ret = (stCPU.pstRAM->stRegisters.OCR1AH << 8 ) |
00128              stCPU.pstRAM->stRegisters.OCR1AL;
00129     return u16Ret;
00130 }
00131
00132 //-----------------------------------------------------------------------
00133 static uint16_t OCR1B_Read()
00134 {
00135     uint16_t u16Ret = 0;
00136
00137     u16Ret = (stCPU.pstRAM->stRegisters.OCR1BH << 8 ) |
00138              stCPU.pstRAM->stRegisters.OCR1BL;
00139     return u16Ret;
00140 }
00141
00142 //-----------------------------------------------------------------------
00143 static uint16_t ICR1_Read()
00144 {
00145     uint16_t u16Ret = 0;
00146
00147     u16Ret = (stCPU.pstRAM->stRegisters.ICR1H << 8 ) |
00148              stCPU.pstRAM->stRegisters.ICR1L;
00149     return u16Ret;
00150 }
00151
00152 //-----------------------------------------------------------------------
00153 static bool Timer16_Is_TOIE1_Enabled()
00154 {
00155     return (stCPU.pstRAM->stRegisters.TIMSK1.TOIE1 == 1);
00156 }
00157
00158 //-----------------------------------------------------------------------
00159 static bool Timer16_Is_OCIE1A_Enabled()
00160 {
00161     return (stCPU.pstRAM->stRegisters.TIMSK1.OCIE1A == 1);
00162 }
00163
00164 //-----------------------------------------------------------------------
00165 static bool Timer16_Is_OCIE1B_Enabled()
00166 {
00167     return (stCPU.pstRAM->stRegisters.TIMSK1.OCIE1B == 1);
00168 }
00169
00170 //-----------------------------------------------------------------------
00171 static bool Timer16_Is_ICIE1_Enabled()
00172 {
00173     return (stCPU.pstRAM->stRegisters.TIMSK1.ICIE1 == 1);
00174 }
00175
00176 //-----------------------------------------------------------------------
00177 static void OV1_Ack(  uint8_t ucVector_)
00178 {
00179     stCPU.pstRAM->stRegisters.TIFR1.TOV1 = 0;
00180 }
00181
00182 //-----------------------------------------------------------------------
00183 static void IC1_Ack(  uint8_t ucVector_)
00184 {
```

```
00185     stCPU.pstRAM->stRegisters.TIFR1.ICF1 = 0;
00186 }
00187
00188 //------------------------------------------------------------------------------
00189 static void COMP1A_Ack(  uint8_t ucVector_)
00190 {
00191     static uint64_t lastcycles = 0;
00192    // printf("COMP1A - Ack'd: %d delta\n", stCPU.u64CycleCount - lastcycles);
00193     lastcycles = stCPU.u64CycleCount;
00194
00195     stCPU.pstRAM->stRegisters.TIFR1.OCF1A = 0;
00196 }
00197
00198 //------------------------------------------------------------------------------
00199 static void COMP1B_Ack(  uint8_t ucVector_)
00200 {
00201     stCPU.pstRAM->stRegisters.TIFR1.OCF1B = 0;
00202 }
00203
00204 //------------------------------------------------------------------------------
00205 static void Timer16_Init(void *context_ )
00206 {
00207     DEBUG_PRINT(stderr, "Timer16 Init\n");
00208
00209     CPU_RegisterInterruptCallback( OV1_Ack, 0x0D);
00210     CPU_RegisterInterruptCallback( IC1_Ack, 0x0A);
00211     CPU_RegisterInterruptCallback( COMP1A_Ack, 0x0B);
00212     CPU_RegisterInterruptCallback( COMP1B_Ack, 0x0C);
00213 }
00214
00215 //------------------------------------------------------------------------------
00216 static void Timer16_Read(void *context_, uint8_t ucAddr_, uint8_t *pucValue_ )
00217 {
00218     DEBUG_PRINT(stderr, "Timer16 Read: 0x%02x\n", ucAddr_);
00219     *pucValue_ = stCPU.pstRAM->au8RAM[ ucAddr_ ];
00220 }
00221
00222 //------------------------------------------------------------------------------
00223 static void TCCR1A_Write( uint8_t ucAddr_, uint8_t ucValue_)
00224 {
00225     // Update the waveform generator mode (WGM11:10) bits.
00226     uint8_t u8WGMBits = ucValue_ & 0x03; // WGM11 and 10 are in bits 0,1
00227     uint8_t u8WGMTemp = (uint8_t)eWGM;
00228     u8WGMTemp &= ~(0x03);
00229     u8WGMTemp |= u8WGMBits;
00230     eWGM = (WaveformGeneratorMode_t)u8WGMTemp;
00231
00232     // Update the memory-mapped register.
00233     stCPU.pstRAM->stRegisters.TCCR1A.r = ucValue_ & 0xF3;
00234 }
00235
00236 //------------------------------------------------------------------------------
00237 static void TCCR1B_Write( uint8_t ucAddr_, uint8_t ucValue_)
00238 {
00239     // Update the waveform generator mode (WGM13:12) bits.
00240     uint8_t u8WGMBits = (ucValue_ >> 1) & 0x0C; // WGM13 and 12 are in register bits 3,4
00241     uint8_t u8WGMTemp = (uint8_t)eWGM;
00242     u8WGMTemp &= ~(0x0C);
00243     u8WGMTemp |= u8WGMBits;
00244     eWGM = (WaveformGeneratorMode_t)u8WGMTemp;
00245
00246     // Update the clock-select bits
00247     uint8_t u8ClockSource = ucValue_ & 0x07; // clock select is last 3 bits in reg
00248     eClockSource = (ClockSource_t)u8ClockSource;
00249     switch (eClockSource)
00250     {
00251     case CLK_SRC_DIV_1:
00252         u16DivCycles = 1;
00253         break;
00254     case CLK_SRC_DIV_8:
00255         u16DivCycles = 8;
00256         break;
00257     case CLK_SRC_DIV_64:
00258         u16DivCycles = 64;
00259         break;
00260     case CLK_SRC_DIV_256:
00261         u16DivCycles = 256;
00262         break;
00263     case CLK_SRC_DIV_1024:
00264         u16DivCycles = 1024;
00265         break;
00266     default:
00267         u16DivCycles = 0;
00268         break;
00269     }
00270
00271     // Update the memory-mapped register.
```

```
00272     stCPU.pstRAM->stRegisters.TCCR1B.r = ucValue_ & 0xDF; // Bit 5 is read-only
00273 }
00274
00275 //-----------------------------------------------------------------------------
00276 static void TCCR1C_Write( uint8_t ucAddr_, uint8_t ucValue_)
00277 {
00278     stCPU.pstRAM->stRegisters.TCCR1C.r = ucValue_;
00279 }
00280
00281 //-----------------------------------------------------------------------------
00282 static void TCNT1L_Write( uint8_t ucAddr_, uint8_t ucValue_)
00283 {
00284     // Writing the low-word forces the high-word to be stored from the internal
00285     // temp register... which is why the high byte must be written first.
00286     stCPU.pstRAM->stRegisters.TCNT1L = ucValue_;
00287     stCPU.pstRAM->stRegisters.TCNT1H = u8Temp;
00288 }
00289 //-----------------------------------------------------------------------------
00290 static void TCNT1H_Write( uint8_t ucAddr_, uint8_t ucValue_)
00291 {
00292     u8Temp = ucValue_;
00293 }
00294 //-----------------------------------------------------------------------------
00295 static void ICR1L_Write( uint8_t ucAddr_, uint8_t ucValue_)
00296 {
00297     // Writing the low-word forces the high-word to be stored from the internal
00298     // temp register... which is why the high byte must be written first.
00299     stCPU.pstRAM->stRegisters.ICR1L = ucValue_;
00300     stCPU.pstRAM->stRegisters.ICR1H = u8Temp;
00301 }
00302 //-----------------------------------------------------------------------------
00303 static void ICR1H_Write( uint8_t ucAddr_, uint8_t ucValue_)
00304 {
00305     u8Temp = ucValue_;
00306 }
00307
00308 //-----------------------------------------------------------------------------
00309 static void OCR1AL_Write( uint8_t ucAddr_, uint8_t ucValue_)
00310 {
00311     // Writing the low-word forces the high-word to be stored from the internal
00312     // temp register... which is why the high byte must be written first.
00313     stCPU.pstRAM->stRegisters.OCR1AL = ucValue_;
00314     stCPU.pstRAM->stRegisters.OCR1AH = u8Temp;
00315 }
00316
00317 //-----------------------------------------------------------------------------
00318 static void OCR1AH_Write( uint8_t ucAddr_, uint8_t ucValue_)
00319 {
00320     u8Temp = ucValue_;
00321 }
00322
00323 //-----------------------------------------------------------------------------
00324 static void OCR1BL_Write( uint8_t ucAddr_, uint8_t ucValue_)
00325 {
00326     // Writing the low-word forces the high-word to be stored from the internal
00327     // temp register... which is why the high byte must be written first.
00328     stCPU.pstRAM->stRegisters.OCR1BL = ucValue_;
00329     stCPU.pstRAM->stRegisters.OCR1BH = u8Temp;
00330 }
00331
00332 //-----------------------------------------------------------------------------
00333 static void OCR1BH_Write( uint8_t ucAddr_, uint8_t ucValue_)
00334 {
00335     u8Temp = ucValue_;
00336 }
00337
00338 //-----------------------------------------------------------------------------
00339 static void Timer16_IntFlagUpdate(void)
00340 {
00341     if (stCPU.pstRAM->stRegisters.TIMSK1.TOIE1 == 1)
00342     {
00343         if (stCPU.pstRAM->stRegisters.TIFR1.TOV1 == 1)
00344         {
00345             DEBUG_PRINT(" TOV1 Interrupt Candidate\n" );
00346             AVR_InterruptCandidate(0x0D);
00347         }
00348         else
00349         {
00350             AVR_ClearCandidate(0x0D);
00351         }
00352     }
00353
00354     if (stCPU.pstRAM->stRegisters.TIMSK1.OCIE1A == 1)
00355     {
00356         if (stCPU.pstRAM->stRegisters.TIFR1.OCF1A == 1)
00357         {
00358             DEBUG_PRINT(" OCF1A Interrupt Candidate\n" );
```

```
00359             AVR_InterruptCandidate(0x0B);
00360         }
00361         else
00362         {
00363             AVR_ClearCandidate(0x0B);
00364         }
00365     }
00366
00367     if (stCPU.pstRAM->stRegisters.TIMSK1.OCIE1B == 1)
00368     {
00369         if (stCPU.pstRAM->stRegisters.TIFR1.OCF1B == 1)
00370         {
00371             DEBUG_PRINT(" OCF1B Interrupt Candidate\n" );
00372             AVR_InterruptCandidate(0x0C);
00373         }
00374         else
00375         {
00376             AVR_ClearCandidate(0x0C);
00377         }
00378     }
00379
00380     if (stCPU.pstRAM->stRegisters.TIMSK1.ICIE1 == 1)
00381     {
00382         if (stCPU.pstRAM->stRegisters.TIFR1.ICF1 == 1)
00383         {
00384             DEBUG_PRINT(" ICF1 Interrupt Candidate\n" );
00385             AVR_InterruptCandidate(0x0A);
00386         }
00387         else
00388         {
00389             AVR_ClearCandidate(0x0A);
00390         }
00391     }
00392 }
00393
00394 //-------------------------------------------------------------------------
00395 // TIFR & TMSK
00396 static void Timer16b_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
00397 {
00398     stCPU.pstRAM->au8RAM[ucAddr_] = ucValue_;
00399     Timer16_IntFlagUpdate();
00400 }
00401
00402 //-------------------------------------------------------------------------
00403 static void Timer16_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
00404 {
00405     switch (ucAddr_)
00406     {
00407     case 0x80:  //TCCR1A
00408         TCCR1A_Write(ucAddr_, ucValue_);
00409         break;
00410     case 0x81:  //TCCR1B
00411         TCCR1B_Write(ucAddr_, ucValue_);
00412         break;
00413     case 0x82:  //TCCR1C
00414         TCCR1C_Write(ucAddr_, ucValue_);
00415         break;
00416     case 0x83:  // Reserved
00417         break;
00418     case 0x84:  // TCNT1L
00419         TCNT1L_Write(ucAddr_, ucValue_);
00420         break;
00421     case 0x85:  // TCNT1H
00422         TCNT1H_Write(ucAddr_, ucValue_);
00423         break;
00424     case 0x86:  // ICR1L
00425         ICR1L_Write(ucAddr_, ucValue_);
00426         break;
00427     case 0x87:  // ICR1H
00428         ICR1H_Write(ucAddr_, ucValue_);
00429         break;
00430     case 0x88:  // OCR1AL
00431         OCR1AL_Write(ucAddr_, ucValue_);
00432         break;
00433     case 0x89:  // OCR1AH
00434         OCR1AH_Write(ucAddr_, ucValue_);
00435         break;
00436     case 0x8A:  // OCR1BL
00437         OCR1BL_Write(ucAddr_, ucValue_);
00438         break;
00439     case 0x8B:  // OCR1BH
00440         OCR1BH_Write(ucAddr_, ucValue_);
00441         break;
00442     default:
00443         break;
00444     }
00445 }
```

```
00446
00447 //-------------------------------------------------------------------------
00448 static void Timer16_Clock(void *context_ )
00449 {
00450     if (eClockSource == CLK_SRC_OFF)
00451     {
00452         return;
00453     }
00454
00455     // Handle clock division logic
00456     bool bUpdateTimer = false;
00457     switch (eClockSource)
00458     {
00459     case CLK_SRC_DIV_1:
00460     case CLK_SRC_DIV_8:
00461     case CLK_SRC_DIV_64:
00462     case CLK_SRC_DIV_256:
00463     case CLK_SRC_DIV_1024:
00464     {
00465         // Decrement the clock-divide value
00466         if (u16DivRemain)
00467         {
00468             //DEBUG_PRINT(" %d ticks remain\n", u16DivRemain);
00469             u16DivRemain--;
00470         }
00471
00472         if (!u16DivRemain)
00473         {
00474             // clock-divider count hits zero, reset and trigger an update.
00475             //DEBUG_PRINT(" expire and reset\n");
00476             if (u16DivCycles)
00477             {
00478                 u16DivRemain = u16DivCycles;
00479                 bUpdateTimer = true;
00480             }
00481         }
00482     }
00483         break;
00484     default:
00485         break;
00486     }
00487
00488
00489
00490     if (bUpdateTimer)
00491     {
00492         // Handle event flags on timer updates
00493         bool bOVF   = false;
00494         bool bCTCA  = false;
00495         bool bCTCB  = false;
00496         bool bICR   = false;
00497         bool bIntr  = false;
00498
00499         //DEBUG_PRINT( " WGM Mode %d\n", eWGM );
00500         switch (eWGM)
00501         {
00502         case WGM_NORMAL:
00503         {
00504             DEBUG_PRINT(" Update Normal\n");
00505             TCNT1_Increment();
00506             if (TCNT1_Read() == 0)
00507             {
00508                 bOVF = true;
00509             }
00510         }
00511             break;
00512         case WGM_CTC_OCR:
00513         {
00514             DEBUG_PRINT(" Update CTC\n");
00515             TCNT1_Increment();
00516             if (TCNT1_Read() == 0)
00517             {
00518                 bOVF = true;
00519             }
00520             else
00521             {
00522                 bool bClearTCNT1 = false;
00523                 if (TCNT1_Read() == OCR1A_Read())
00524                 {
00525                     DEBUG_PRINT(" CTC1A Match\n" );
00526                     bCTCA = true;
00527                     bClearTCNT1 = true;
00528                 }
00529                 if (TCNT1_Read() == ICR1_Read())
00530                 {
00531                     DEBUG_PRINT(" ICR1 Match\n" );
00532                     bICR = true;
00533                     bClearTCNT1 = true;
```

```
00534                 }
00535                 if (bClearTCNT1)
00536                 {
00537                     TCNT1_Clear();
00538                 }
00539             }
00540         }
00541             break;
00542         default:
00543             break;
00544         }
00545
00546         // Set interrupt flags if an appropriate transition has taken place
00547         if (bOVF)
00548         {
00549             DEBUG_PRINT(" TOV1 Set\n" );
00550             stCPU.pstRAM->stRegisters.TIFR1.TOV1 = 1;
00551             bIntr = true;
00552         }
00553         if (bCTCA)
00554         {
00555             DEBUG_PRINT(" OCF1A Set\n" );
00556             stCPU.pstRAM->stRegisters.TIFR1.OCF1A = 1;
00557             bIntr = true;
00558         }
00559         if (bCTCB)
00560         {
00561             DEBUG_PRINT(" OCF1B Set\n" );
00562             stCPU.pstRAM->stRegisters.TIFR1.OCF1B = 1;
00563             bIntr = true;
00564         }
00565         if (bICR)
00566         {
00567             DEBUG_PRINT(" ICF1 Set\n" );
00568             stCPU.pstRAM->stRegisters.TIFR1.ICF1 = 1;
00569             bIntr = true;
00570         }
00571
00572         if (bIntr)
00573         {
00574             Timer16_IntFlagUpdate();
00575         }
00576     }
00577 }
00578
00579 //-------------------------------------------------------------------------
00580 AVRPeripheral stTimer16 =
00581 {
00582     Timer16_Init,
00583     Timer16_Read,
00584     Timer16_Write,
00585     Timer16_Clock,
00586     0,
00587     0x80,
00588     0x8B
00589 };
00590
00591 //-------------------------------------------------------------------------
00592 AVRPeripheral stTimer16a =
00593 {
00594     0,
00595     Timer16_Read,
00596     Timer16b_Write,
00597     0,
00598     0,
00599     0x36,
00600     0x36
00601 };
00602
00603 //-------------------------------------------------------------------------
00604 AVRPeripheral stTimer16b =
00605 {
00606     0,
00607     Timer16_Read,
00608     Timer16b_Write,
00609     0,
00610     0,
00611     0x6F,
00612     0x6F
00613 };
```

4.71 mega_timer16.h File Reference

ATMega 16-bit timer implementation.

```
#include "avr_peripheral.h"
```

Variables

- AVRPeripheral **stTimer16**
- AVRPeripheral **stTimer16a**
- AVRPeripheral **stTimer16b**

4.71.1 Detailed Description

ATMega 16-bit timer implementation.

Definition in file mega_timer16.h.

4.72 mega_timer16.h

```
00001 /*************************************************************************
00002  *      (      (                       (      |
00003  *    )\ )   )\ )     (              )\ )    |
00004  *   (()/( (()/(     )\      (    (  (()/(    | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\   /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \    \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\    \_/   |_|_\    |
00010  *                                          | "Yeah, it does Arduino..."
00011  * ----------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *************************************************************************/
00021 #ifndef __MEGA_TIMER16_H__
00022 #define __MEGA_TIMER16_H__
00023
00024 #include "avr_peripheral.h"
00025
00026 extern AVRPeripheral stTimer16;
00027 extern AVRPeripheral stTimer16a;
00028 extern AVRPeripheral stTimer16b;
00029
00030 #endif //__MEGA_EINT_H__
```

4.73 mega_timer8.c File Reference

ATMega 8-bit timer implementation.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "avr_cpu.h"
#include "avr_peripheral.h"
#include "avr_periphregs.h"
#include "avr_interrupt.h"
```

Macros

- #define **DEBUG_PRINT**(...)

Enumerations

- enum ClockSource_t {
  CLK_SRC_OFF, **CLK_SRC_DIV_1**, **CLK_SRC_DIV_8**, **CLK_SRC_DIV_64**,
  CLK_SRC_DIV_256, **CLK_SRC_DIV_1024**, **CLK_SRC_T1_FALL**, **CLK_SRC_T1_RISE**,
  CLK_SRC_OFF, **CLK_SRC_DIV_1**, **CLK_SRC_DIV_8**, **CLK_SRC_DIV_64**,
  CLK_SRC_DIV_256, **CLK_SRC_DIV_1024**, **CLK_SRC_T1_FALL**, **CLK_SRC_T1_RISE** }
- enum **WaveformGeneratorMode_t** {
  WGM_NORMAL, **WGM_PWM_PC_8BIT**, **WGM_PWM_PC_9BIT**, **WGM_PWM_PC_10BIT**,
  WGM_CTC_OCR, **WGM_PWM_8BIT**, **WGM_PWM_9BIT**, **WGM_PWM_10BIT**,
  WGM_PWM_PC_FC_ICR, **WGM_PWM_PC_FC_OCR**, **WGM_PWM_PC_ICR**, **WGM_PWM_PC_OCR**,
  WGM_CTC_ICR, **WGM_RESERVED**, **WGM_FAST_PWM_ICR**, **WGM_FAST_PWM_OCR**,
  WGM_NORMAL, **WGM_PWM_PC_FF**, **WGM_CTC_OCR**, **WGM_FAST_PWM_FF**,
  WGM_RESERVED_1, **WGM_PWM_PC_OCR**, **WGM_RESERVED_2**, **WGM_FAST_PWM_OCR** }
- enum **CompareOutputMode_t** {
  COM_NORMAL, **COM_TOGGLE_MATCH**, **COM_CLEAR_MATCH**, **COM_SET_MATCH**,
  COM_NORMAL, **COM_TOGGLE_MATCH**, **COM_CLEAR_MATCH**, **COM_SET_MATCH** }

Functions

- static void **TCNT0_Increment** ()
- static uint8_t **TCNT0_Read** ()
- static void **TCNT0_Clear** ()
- static uint8_t **OCR0A_Read** ()
- static uint8_t **OCR0B_Read** ()
- static bool **Timer8_Is_TOIE0_Enabled** ()
- static bool **Timer8_Is_OCIE0A_Enabled** ()
- static bool **Timer8_Is_OCIE1B_Enabled** ()
- static void **OV0_Ack** (uint8_t ucVector_)
- static void **COMP0A_Ack** (uint8_t ucVector_)
- static void **COMP0B_Ack** (uint8_t ucVector_)
- static void **Timer8_Init** (void *context_)
- static void **Timer8_Read** (void *context_, uint8_t ucAddr_, uint8_t *pucValue_)
- static void **TCCR0A_Write** (uint8_t ucAddr_, uint8_t ucValue_)
- static void **TCCR0B_Write** (uint8_t ucAddr_, uint8_t ucValue_)
- static void **TCNT0_Write** (uint8_t ucAddr_, uint8_t ucValue_)
- static void **OCR0A_Write** (uint8_t ucAddr_, uint8_t ucValue_)
- static void **OCR0B_Write** (uint8_t ucAddr_, uint8_t ucValue_)
- static void **Timer8_IntFlagUpdate** (void)
- static void **Timer8b_Write** (void *context_, uint8_t ucAddr_, uint8_t ucValue_)
- static void **Timer8_Write** (void *context_, uint8_t ucAddr_, uint8_t ucValue_)
- static void Timer8_Clock (void *context_)

Variables

- static uint16_t **u16DivCycles** = 0
- static uint16_t **u16DivRemain** = 0
- static ClockSource_t **eClockSource** = CLK_SRC_OFF
- static WaveformGeneratorMode_t **eWGM** = WGM_NORMAL
- static CompareOutputMode_t **eCOM1A** = COM_NORMAL
- static CompareOutputMode_t **eCOM1B** = COM_NORMAL
- static uint8_t **u8Temp**
- static uint16_t **u8Count**
- AVRPeripheral **stTimer8**
- AVRPeripheral **stTimer8a**
- AVRPeripheral **stTimer8b**

4.73.1 Detailed Description

ATMega 8-bit timer implementation.

Definition in file mega_timer8.c.

4.73.2 Enumeration Type Documentation

4.73.2.1 enum ClockSource_t

! This implementation only tracks the basic timer/capture/compare functionality of the peripheral, to match what's used in Mark3. Future considerations, TBD.

Definition at line 38 of file mega_timer8.c.

4.73.3 Function Documentation

4.73.3.1 static void Timer8_Clock (void * *context_*) [static]

! ToDo - Handle external timer generated events.

Definition at line 315 of file mega_timer8.c.

4.73.4 Variable Documentation

4.73.4.1 AVRPeripheral stTimer8

Initial value:

```
=
{
    Timer8_Init,
    Timer8_Read,
    Timer8_Write,
    Timer8_Clock,
    0,
    0x44,
    0x48
}
```

Definition at line 428 of file mega_timer8.c.

4.73.4.2 AVRPeripheral stTimer8a

Initial value:

```
=
{
    0,
    Timer8_Read,
    Timer8b_Write,
    0,
    0,
    0x35,
    0x35
}
```

Definition at line 441 of file mega_timer8.c.

4.73.4.3 AVRPeripheral stTimer8b

Initial value:

```
=
{
    0,
    Timer8_Read,
    Timer8b_Write,
    0,
    0,
    0x6E,
    0x6E
}
```

Definition at line 453 of file mega_timer8.c.

4.74 mega_timer8.c

```
00001 /*************************************************************************
00002  *      (     (                        (     |
00003  *    )\ )  )\ )     (              )\ )    |
00004  *   (()/( (()/(     )\      (   (  (()/(    | -- [ Funkenstein ] ------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))       | -- [ Litle ] -------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+-------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *************************************************************************/
00021 #include <stdio.h>
00022 #include <stdlib.h>
00023 #include <string.h>
00024 #include "avr_cpu.h"
00025 #include "avr_peripheral.h"
00026 #include "avr_periphregs.h"
00027 #include "avr_interrupt.h"
00028
00029 #define DEBUG_PRINT(...)
00030
00031 //-----------------------------------------------------------------------------
00035 //-----------------------------------------------------------------------------
00036
00037 //-----------------------------------------------------------------------------
00038 typedef enum
00039 {
00040     CLK_SRC_OFF,
00041     CLK_SRC_DIV_1,
00042     CLK_SRC_DIV_8,
00043     CLK_SRC_DIV_64,
00044     CLK_SRC_DIV_256,
00045     CLK_SRC_DIV_1024,
00046     CLK_SRC_T1_FALL,
00047     CLK_SRC_T1_RISE
00048 } ClockSource_t;
00049
00050 //-----------------------------------------------------------------------------
00051 typedef enum
00052 {
00053     WGM_NORMAL,
00054     WGM_PWM_PC_FF,
00055     WGM_CTC_OCR,
00056     WGM_FAST_PWM_FF,
00057     WGM_RESERVED_1, // Not a valid mode
00058     WGM_PWM_PC_OCR,
00059     WGM_RESERVED_2, // Not a valid mode
00060     WGM_FAST_PWM_OCR
00061 } WaveformGeneratorMode_t;
00062
00063 //-----------------------------------------------------------------------------
00064 typedef enum
00065 {
00066     COM_NORMAL,         // OCA
00067     COM_TOGGLE_MATCH,   // Toggle on match
00068     COM_CLEAR_MATCH,
00069     COM_SET_MATCH
00070 } CompareOutputMode_t;
```

```
00071
00072 //-------------------------------------------------------------------------
00073 static uint16_t u16DivCycles = 0;
00074 static uint16_t u16DivRemain = 0;
00075 static ClockSource_t eClockSource   = CLK_SRC_OFF;
00076 static WaveformGeneratorMode_t eWGM = WGM_NORMAL;
00077 static CompareOutputMode_t eCOM1A = COM_NORMAL;
00078 static CompareOutputMode_t eCOM1B = COM_NORMAL;
00079
00080 //-------------------------------------------------------------------------
00081 static uint8_t  u8Temp;  // The 8-bit temporary register used in 16-bit register accesses
00082 static uint16_t u8Count; // Internal 16-bit count register
00083
00084 //-------------------------------------------------------------------------
00085 static void TCNT0_Increment()
00086 {
00087     stCPU.pstRAM->stRegisters.TCNT0++;
00088 }
00089
00090 //-------------------------------------------------------------------------
00091 static uint8_t TCNT0_Read()
00092 {
00093     return stCPU.pstRAM->stRegisters.TCNT0;
00094 }
00095
00096 //-------------------------------------------------------------------------
00097 static void TCNT0_Clear()
00098 {
00099     stCPU.pstRAM->stRegisters.TCNT0 = 0;
00100 }
00101
00102 //-------------------------------------------------------------------------
00103 static uint8_t OCR0A_Read()
00104 {
00105     return stCPU.pstRAM->stRegisters.OCR0A;
00106 }
00107
00108 //-------------------------------------------------------------------------
00109 static uint8_t OCR0B_Read()
00110 {
00111     return stCPU.pstRAM->stRegisters.OCR0B;
00112 }
00113
00114 //-------------------------------------------------------------------------
00115 static bool Timer8_Is_TOIE0_Enabled()
00116 {
00117     return (stCPU.pstRAM->stRegisters.TIMSK0.TOIE0 == 1);
00118 }
00119
00120 //-------------------------------------------------------------------------
00121 static bool Timer8_Is_OCIE0A_Enabled()
00122 {
00123     return (stCPU.pstRAM->stRegisters.TIMSK0.OCIE0A == 1);
00124 }
00125
00126 //-------------------------------------------------------------------------
00127 static bool Timer8_Is_OCIE1B_Enabled()
00128 {
00129     return (stCPU.pstRAM->stRegisters.TIMSK0.OCIE0B == 1);
00130 }
00131
00132 //-------------------------------------------------------------------------
00133 static void OV0_Ack(  uint8_t ucVector_)
00134 {
00135     static uint64_t lastcycles = 0;
00136     stCPU.pstRAM->stRegisters.TIFR0.TOV0 = 0;
00137    // printf("OV0 - Ack'd: %d delta\n", stCPU.u64CycleCount - lastcycles);
00138     lastcycles = stCPU.u64CycleCount;
00139 }
00140
00141 //-------------------------------------------------------------------------
00142 static void COMP0A_Ack(  uint8_t ucVector_)
00143 {
00144     stCPU.pstRAM->stRegisters.TIFR0.OCF0A = 0;
00145 }
00146
00147 //-------------------------------------------------------------------------
00148 static void COMP0B_Ack(  uint8_t ucVector_)
00149 {
00150     stCPU.pstRAM->stRegisters.TIFR0.OCF0B = 0;
00151 }
00152
00153 //-------------------------------------------------------------------------
00154 static void Timer8_Init(void *context_ )
00155 {
00156     DEBUG_PRINT( "Timer8 Init\n");
00157     CPU_RegisterInterruptCallback( OV0_Ack, 0x10);
```

```
00158     CPU_RegisterInterruptCallback( COMP0A_Ack, 0x0E);
00159     CPU_RegisterInterruptCallback( COMP0B_Ack, 0x0F);
00160 }
00161
00162 //------------------------------------------------------------------------
00163 static void Timer8_Read(void *context_, uint8_t ucAddr_, uint8_t *pucValue_ )
00164 {
00165     DEBUG_PRINT( "Timer8 Read: 0x%02x\n", ucAddr_);
00166     *pucValue_ = stCPU.pstRAM->au8RAM[ ucAddr_ ];
00167 }
00168
00169 //------------------------------------------------------------------------
00170 static void TCCR0A_Write( uint8_t ucAddr_, uint8_t ucValue_)
00171 {
00172     // Update the waveform generator mode (WGM1:0) bits.
00173     uint8_t u8WGMBits = ucValue_ & 0x03; // WGM1 and 0 are in bits 0,1
00174     uint8_t u8WGMTemp = (uint8_t)eWGM;
00175     u8WGMTemp &= ~(0x03);
00176     u8WGMTemp |= u8WGMBits;
00177     eWGM = (WaveformGeneratorMode_t)u8WGMTemp;
00178
00179     // Update the memory-mapped register.
00180     stCPU.pstRAM->stRegisters.TCCR0A.r = ucValue_ & 0xF3;
00181 }
00182
00183 //------------------------------------------------------------------------
00184 static void TCCR0B_Write( uint8_t ucAddr_, uint8_t ucValue_)
00185 {
00186     // Update the waveform generator mode (WGM2) bit
00187     uint8_t u8WGMBits = (ucValue_ >> 1) & 0x04; // WGM2 is in bit 3 of the register
00188     uint8_t u8WGMTemp = (uint8_t)eWGM;
00189     u8WGMTemp &= ~(0x04);
00190     u8WGMTemp |= u8WGMBits;
00191     eWGM = (WaveformGeneratorMode_t)u8WGMTemp;
00192
00193     // Update the clock-select bits
00194     uint8_t u8ClockSource = ucValue_ & 0x07; // clock select is last 3 bits in reg
00195     eClockSource = (ClockSource_t)u8ClockSource;
00196     switch (eClockSource)
00197     {
00198     case CLK_SRC_DIV_1:
00199         u16DivCycles = 1;
00200         break;
00201     case CLK_SRC_DIV_8:
00202         u16DivCycles = 8;
00203         break;
00204     case CLK_SRC_DIV_64:
00205         u16DivCycles = 64;
00206         break;
00207     case CLK_SRC_DIV_256:
00208         u16DivCycles = 256;
00209         break;
00210     case CLK_SRC_DIV_1024:
00211         u16DivCycles = 1024;
00212         break;
00213     default:
00214         u16DivCycles = 0;
00215         break;
00216     }
00217     DEBUG_PRINT(" ClockSource = %d, %d cycles\n", eClockSource, u16DivCycles);
00218     // Update the memory-mapped register.
00219     stCPU.pstRAM->stRegisters.TCCR0B.r = ucValue_ & 0xCF; // Bit 5&6 are read-only
00220 }
00221
00222 //------------------------------------------------------------------------
00223 static void TCNT0_Write( uint8_t ucAddr_, uint8_t ucValue_)
00224 {
00225     stCPU.pstRAM->stRegisters.TCNT0 = ucValue_;
00226 }
00227
00228 //------------------------------------------------------------------------
00229 static void OCR0A_Write( uint8_t ucAddr_, uint8_t ucValue_)
00230 {
00231     stCPU.pstRAM->stRegisters.OCR0A = ucValue_;
00232 }
00233
00234 //------------------------------------------------------------------------
00235 static void OCR0B_Write( uint8_t ucAddr_, uint8_t ucValue_)
00236 {
00237     stCPU.pstRAM->stRegisters.OCR0B = ucValue_;
00238 }
00239
00240 //------------------------------------------------------------------------
00241 static void Timer8_IntFlagUpdate(void)
00242 {
00243     if (stCPU.pstRAM->stRegisters.TIMSK0.TOIE0 == 1)
00244     {
```

```
00245         if (stCPU.pstRAM->stRegisters.TIFR0.TOV0 == 1)
00246         {
00247             DEBUG_PRINT(" TOV0 Interrupt Candidate\n" );
00248             AVR_InterruptCandidate(0x10);
00249         }
00250         else
00251         {
00252             AVR_ClearCandidate(0x10);
00253         }
00254     }
00255     if (stCPU.pstRAM->stRegisters.TIMSK0.OCIE0A == 1)
00256     {
00257         if (stCPU.pstRAM->stRegisters.TIFR0.OCF0A == 1)
00258         {
00259             DEBUG_PRINT(" OCF0A Interrupt Candidate\n" );
00260             AVR_InterruptCandidate(0x0E);
00261         }
00262         else
00263         {
00264             AVR_ClearCandidate(0x0E);
00265         }
00266     }
00267     if (stCPU.pstRAM->stRegisters.TIMSK0.OCIE0B == 1)
00268     {
00269         if (stCPU.pstRAM->stRegisters.TIFR0.OCF0B == 1)
00270         {
00271             DEBUG_PRINT(" OCF0B Interrupt Candidate\n" );
00272             AVR_InterruptCandidate(0x0F);
00273         }
00274         else
00275         {
00276             AVR_ClearCandidate(0x0F);
00277         }
00278     }
00279 }
00280
00281 //-------------------------------------------------------------------------
00282 static void Timer8b_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
00283 {
00284     stCPU.pstRAM->au8RAM[ucAddr_] = ucValue_;
00285     Timer8_IntFlagUpdate();
00286 }
00287
00288 //-------------------------------------------------------------------------
00289 static void Timer8_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
00290 {
00291     DEBUG_PRINT("Timer8_Write: %d=%d\n", ucAddr_, ucValue_);
00292     switch (ucAddr_)
00293     {
00294     case 0x44:  //TCCR1A
00295         TCCR0A_Write(ucAddr_, ucValue_);
00296         break;
00297     case 0x45:  //TCCR1B
00298         TCCR0B_Write(ucAddr_, ucValue_);
00299         break;
00300     case 0x46:  // TCNT0
00301         TCNT0_Write(ucAddr_, ucValue_);
00302         break;
00303     case 0x47:  // OCR0A
00304         OCR0A_Write(ucAddr_, ucValue_);
00305         break;
00306     case 0x48:  // OCR0B
00307         OCR0B_Write(ucAddr_, ucValue_);
00308         break;
00309     default:
00310         break;
00311     }
00312 }
00313
00314 //-------------------------------------------------------------------------
00315 static void Timer8_Clock(void *context_ )
00316 {
00317     if (eClockSource == CLK_SRC_OFF)
00318     {
00319         return;
00320     }
00321
00322     // Handle clock division logic
00323     bool bUpdateTimer = false;
00324     switch (eClockSource)
00325     {
00326     case CLK_SRC_DIV_1:
00327     case CLK_SRC_DIV_8:
00328     case CLK_SRC_DIV_64:
00329     case CLK_SRC_DIV_256:
00330     case CLK_SRC_DIV_1024:
00331     {
```

```
00332         // Decrement the clock-divide value
00333         if (u16DivRemain)
00334         {
00335             //DEBUG_PRINT(" %d ticks remain\n", u16DivRemain);
00336             u16DivRemain--;
00337         }
00338 
00339         if (!u16DivRemain)
00340         {
00341             // clock-divider count hits zero, reset and trigger an update.
00342             DEBUG_PRINT(" expire and reset\n");
00343             if (u16DivCycles)
00344             {
00345                 u16DivRemain = u16DivCycles;
00346                 bUpdateTimer = true;
00347             }
00348         }
00349     }
00350         break;
00351     default:
00353         break;
00354     }
00355 
00356 
00357     if (bUpdateTimer)
00358     {
00359         // Handle event flags on timer updates
00360         bool bOVF   = false;
00361         bool bCTCA  = false;
00362         bool bCTCB  = false;
00363         bool bIntr  = false;
00364 
00365         switch (eWGM)
00366         {
00367         case WGM_NORMAL:
00368         {
00369             DEBUG_PRINT(" Update Normal, TCNT = %d\n", TCNT0_Read());
00370             TCNT0_Increment();
00371             if (TCNT0_Read() == 0)
00372             {
00373                 bOVF = true;
00374             }
00375         }
00376             break;
00377         case WGM_CTC_OCR:
00378         {
00379             DEBUG_PRINT(" Update CTC\n");
00380             TCNT0_Increment();
00381             if (TCNT0_Read() == 0)
00382             {
00383                 bOVF = true;
00384             }
00385             else
00386             {
00387                 if (TCNT0_Read() == OCR0A_Read())
00388                 {
00389                     DEBUG_PRINT(" CTC0A Match\n" );
00390                     bCTCA = true;
00391                     TCNT0_Clear();
00392                 }
00393             }
00394         }
00395             break;
00396         default:
00397             break;
00398         }
00399 
00400         // Set interrupt flags if an appropriate transition has taken place
00401         if (bOVF)
00402         {
00403             DEBUG_PRINT(" TOV0 Set\n" );
00404             stCPU.pstRAM->stRegisters.TIFR0.TOV0 = 1;
00405             bIntr = true;
00406         }
00407         if (bCTCA)
00408         {
00409             DEBUG_PRINT(" OCF0A Set\n" );
00410             stCPU.pstRAM->stRegisters.TIFR0.OCF0A = 1;
00411             bIntr = true;
00412         }
00413         if (bCTCB)
00414         {
00415             DEBUG_PRINT(" OCF0B Set\n" );
00416             stCPU.pstRAM->stRegisters.TIFR0.OCF0B = 1;
00417             bIntr = true;
00418         }
00419 
```

```
00420         if (bIntr)
00421         {
00422             Timer8_IntFlagUpdate();
00423         }
00424     }
00425 }
00426
00427 //-------------------------------------------------------------------------
00428 AVRPeripheral stTimer8 =
00429 {
00430     Timer8_Init,
00431     Timer8_Read,
00432     Timer8_Write,
00433     Timer8_Clock,
00434     0,
00435     0x44,
00436     0x48
00437 };
00438
00439
00440 //-------------------------------------------------------------------------
00441 AVRPeripheral stTimer8a =
00442 {
00443     0,
00444     Timer8_Read,
00445     Timer8b_Write,
00446     0,
00447     0,
00448     0x35,
00449     0x35
00450 };
00451
00452 //-------------------------------------------------------------------------
00453 AVRPeripheral stTimer8b =
00454 {
00455     0,
00456     Timer8_Read,
00457     Timer8b_Write,
00458     0,
00459     0,
00460     0x6E,
00461     0x6E
00462 };
```

4.75 mega_timer8.h File Reference

ATMega 8-bit timer implementation.

```
#include "avr_peripheral.h"
```

Variables

- AVRPeripheral **stTimer8**
- AVRPeripheral **stTimer8a**
- AVRPeripheral **stTimer8b**

4.75.1 Detailed Description

ATMega 8-bit timer implementation.

Definition in file mega_timer8.h.

4.76 mega_timer8.h

```
00001 /*************************************************************************
00002  *      (     (                        (     |
00003  *    )\ )  )\ )     (              )\ )    |
00004  *   (()/( (()/(     )\     (   (  (()/(    | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] -------------------
```

```
00006  *    (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *    | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *    | __| | |__    / _ \    \ V /  |   /   | -- [ Runtime ] -----------------
00009  *    |_|   |____| /_/ \_\    \_/   |_|_\    |
00010  *                                            | "Yeah, it does Arduino..."
00011  * ---------------------------------------+--------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  ************************************************************************/
00021 #ifndef __MEGA_TIMER8_H__
00022 #define __MEGA_TIMER8_H__
00023
00024 #include "avr_peripheral.h"
00025
00026 extern AVRPeripheral stTimer8;
00027 extern AVRPeripheral stTimer8a;
00028 extern AVRPeripheral stTimer8b;
00029
00030 #endif //__MEGA_EINT_H__
```

4.77 mega_uart.c File Reference

Implements an atmega UART plugin.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "avr_cpu.h"
#include "avr_peripheral.h"
#include "avr_periphregs.h"
#include "avr_interrupt.h"
```

Macros

- #define DEBUG_PRINT(...)

  Plugin must interface with the following registers:

Functions

- static void **Echo_Tx** ()
- static void **Echo_Rx** ()
- static bool **UART_IsRxEnabled** (void)
- static bool **UART_IsTxEnabled** (void)
- static bool **UART_IsTxIntEnabled** (void)
- static bool **UART_IsDREIntEnabled** (void)
- static bool **UART_IsRxIntEnabled** (void)
- static bool **UART_IsDoubleSpeed** ()
- static void **UART_SetDoubleSpeed** ()
- static void **UART_SetEmpty** (void)
- static void **UART_ClearEmpty** (void)
- static bool **UART_IsEmpty** (void)
- static bool **UART_IsTxComplete** (void)
- static void **UART_TxComplete** (void)
- static bool **UART_IsRxComplete** (void)
- static void **UART_RxComplete** (void)
- static void **TXC0_Callback** (uint8_t ucVector_)
- static void **UART_Init** (void *context_)
- static void **UART_Read** (void *context_, uint8_t ucAddr_, uint8_t *pucValue_)
- static void **UART_WriteBaudReg** ()

- static void **UART_WriteDataReg** ()
- static void **UART_WriteUCSR0A** (uint8_t u8Value_)
- static void **UART_UpdateInterruptFlags** (void)
- static void **UART_WriteUCSR0B** (uint8_t u8Value_)
- static void **UART_WriteUCSR0C** (uint8_t u8Value_)
- static void **UART_Write** (void *context_, uint8_t ucAddr_, uint8_t ucValue_)
- static void **UART_TxClock** (void *context_)
- static void **UART_RxClock** (void *context_)
- static void **UART_Clock** (void *context_)

Variables

- static bool **bUDR_Empty** = true
- static bool **bTSR_Empty** = true
- static uint8_t **RXB** = 0
- static uint8_t **TXB** = 0
- static uint8_t **TSR** = 0
- static uint8_t **RSR** = 0
- static uint32_t **u32BaudTicks** = 0
- static uint32_t **u32TxTicksRemaining** = 0
- static uint32_t **u32RxTicksRemaining** = 0
- AVRPeripheral **stUART**

4.77.1 Detailed Description

Implements an atmega UART plugin.

Definition in file mega_uart.c.

4.77.2 Macro Definition Documentation

4.77.2.1 #define DEBUG_PRINT(...)

Plugin must interface with the following registers:

UDRn UCSRnA UCSRnB UCSRnC UBBRnL UBBRnH

Definition at line 42 of file mega_uart.c.

4.77.3 Variable Documentation

4.77.3.1 AVRPeripheral stUART

Initial value:

```
=
{
    UART_Init,
    UART_Read,
    UART_Write,
    UART_Clock,
    0,
    0xC0,
    0xC6
}
```

Definition at line 433 of file mega_uart.c.

4.78 mega_uart.c

```
00001 /*****************************************************************************
00002  *      (     (                          (      |
00003  *    )\ )  )\ )    (              )\ )   |
00004  *   (()/( (()/(    )\      (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ----------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] ------------------
00008  *   | __| | |__    / _ \   \ V /  |   /   | -- [ Runtime ] ------------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\   |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00033 #include <stdio.h>
00034 #include <stdlib.h>
00035 #include <string.h>
00036 #include "avr_cpu.h"
00037 #include "avr_peripheral.h"
00038 #include "avr_periphregs.h"
00039 #include "avr_interrupt.h"
00040
00041 #if 1
00042 #define DEBUG_PRINT(...)
00043 #else
00044 #define DEBUG_PRINT printf
00045 #endif
00046
00047 //-----------------------------------------------------------------------------
00048 static bool bUDR_Empty = true;
00049 static bool bTSR_Empty = true;
00050
00051 static uint8_t RXB = 0; // receive buffer
00052 static uint8_t TXB = 0; // transmit buffer
00053 static uint8_t TSR = 0; // transmit shift register.
00054 static uint8_t RSR = 0; // receive shift register.
00055
00056 static uint32_t u32BaudTicks = 0;
00057 static uint32_t u32TxTicksRemaining = 0;
00058 static uint32_t u32RxTicksRemaining = 0;
00059
00060 //-----------------------------------------------------------------------------
00061 static void Echo_Tx()
00062 {
00063     printf("%c", TSR);
00064 }
00065
00066 //-----------------------------------------------------------------------------
00067 static void Echo_Rx()
00068 {
00069     printf("%c", RSR);
00070 }
00071
00072 //-----------------------------------------------------------------------------
00073 static bool UART_IsRxEnabled( void )
00074 {
00075     //DEBUG_PRINT( "RxEnabled\n");
00076     return (stCPU.pstRAM->stRegisters.UCSR0B.RXEN0 == 1);
00077 }
00078
00079 //-----------------------------------------------------------------------------
00080 static bool UART_IsTxEnabled( void )
00081 {
00082     //DEBUG_PRINT( "TxEnabled\n");
00083     return (stCPU.pstRAM->stRegisters.UCSR0B.TXEN0 == 1);
00084 }
00085
00086 //-----------------------------------------------------------------------------
00087 static bool UART_IsTxIntEnabled( void )
00088 {
00089     return (stCPU.pstRAM->stRegisters.UCSR0B.TXCIE0 == 1);
00090 }
00091
00092 //-----------------------------------------------------------------------------
00093 static bool UART_IsDREIntEnabled( void )
00094 {
00095     return (stCPU.pstRAM->stRegisters.UCSR0B.UDRIE0 == 1);
00096 }
00097
00098 //-----------------------------------------------------------------------------
00099 static bool UART_IsRxIntEnabled( void )
00100 {
00101     return (stCPU.pstRAM->stRegisters.UCSR0B.RXCIE0 == 1);
00102 }
```

```
00103
00104 //-----------------------------------------------------------------------------
00105 static bool UART_IsDoubleSpeed()
00106 {
00107     return (stCPU.pstRAM->stRegisters.UCSR0A.U2X0 == 1);
00108 }
00109
00110 //-----------------------------------------------------------------------------
00111 static void UART_SetDoubleSpeed()
00112 {
00113     stCPU.pstRAM->stRegisters.UCSR0A.U2X0 = 1;
00114 }
00115
00116 //-----------------------------------------------------------------------------
00117 static void UART_SetEmpty( void )
00118 {
00119     stCPU.pstRAM->stRegisters.UCSR0A.UDRE0 = 1;
00120 }
00121
00122 //-----------------------------------------------------------------------------
00123 static void UART_ClearEmpty( void )
00124 {
00125     stCPU.pstRAM->stRegisters.UCSR0A.UDRE0 = 0;
00126 }
00127
00128 //-----------------------------------------------------------------------------
00129 static bool UART_IsEmpty( void )
00130 {
00131     return (stCPU.pstRAM->stRegisters.UCSR0A.UDRE0 == 1);
00132 }
00133
00134 //-----------------------------------------------------------------------------
00135 static bool UART_IsTxComplete( void )
00136 {
00137     return (stCPU.pstRAM->stRegisters.UCSR0A.TXC0 == 1);
00138 }
00139
00140 //-----------------------------------------------------------------------------
00141 static void UART_TxComplete( void )
00142 {
00143     stCPU.pstRAM->stRegisters.UCSR0A.TXC0 = 1;
00144 }
00145
00146 //-----------------------------------------------------------------------------
00147 static bool UART_IsRxComplete( void )
00148 {
00149     return (stCPU.pstRAM->stRegisters.UCSR0A.RXC0 == 1);
00150 }
00151
00152 //-----------------------------------------------------------------------------
00153 static void UART_RxComplete( void )
00154 {
00155     stCPU.pstRAM->stRegisters.UCSR0A.RXC0 = 1;
00156 }
00157
00158 //-----------------------------------------------------------------------------
00159 static void TXC0_Callback(  uint8_t ucVector_ )
00160 {
00161     // On TX Complete interrupt, automatically clear the TXC0 flag.
00162     stCPU.pstRAM->stRegisters.UCSR0A.TXC0 = 0;
00163 }
00164
00165 //-----------------------------------------------------------------------------
00166 static void UART_Init(void *context_ )
00167 {
00168     DEBUG_PRINT("UART Init\n");
00169     stCPU.pstRAM->stRegisters.UCSR0A.UDRE0 = 1;
00170
00171     CPU_RegisterInterruptCallback( TXC0_Callback, 0x14); // TX Complete
00172 }
00173
00174 //-----------------------------------------------------------------------------
00175 static void UART_Read(void *context_, uint8_t ucAddr_, uint8_t *pucValue_ )
00176 {
00177     DEBUG_PRINT( "UART Read: 0x%02x\n", ucAddr_);
00178     *pucValue_ = stCPU.pstRAM->au8RAM[ ucAddr_ ];
00179     switch (ucAddr_)
00180     {
00181         case 0xC6: // UDR0
00182             stCPU.pstRAM->stRegisters.UCSR0A.RXC0 = 0;
00183             break;
00184         default:
00185             break;
00186     }
00187 }
00188
00189 //-----------------------------------------------------------------------------
```

```
00190 static void UART_WriteBaudReg()
00191 {
00192     DEBUG_PRINT( "WriteBaud\n");
00193     uint16_t u16Baud =  (uint16_t)(stCPU.pstRAM->stRegisters.UBRR0L) |
00194                         ((uint16_t)(stCPU.pstRAM->stRegisters.UBRR0H) << 8);
00195
00196     u32BaudTicks = u16Baud;
00197 }
00198
00199 //-------------------------------------------------------------------------
00200 static void UART_WriteDataReg()
00201 {
00202     DEBUG_PRINT("UART Write UDR...\n");
00203     if (UART_IsTxEnabled())
00204     {
00205         DEBUG_PRINT("Enabled...\n");
00206         // Only set the baud timer if the UART is idle
00207         if (!u32TxTicksRemaining)
00208         {
00209             u32TxTicksRemaining = u32BaudTicks;
00210             if (UART_IsDoubleSpeed())
00211             {
00212                 u32TxTicksRemaining >>= 1;
00213             }
00214         }
00215
00216         // If the shift register is empty, load it immediately
00217         if (bTSR_Empty)
00218         {
00219             TSR = stCPU.pstRAM->stRegisters.UDR0;
00220             TXB = 0;
00221             bTSR_Empty = false;
00222             bUDR_Empty = true;
00223             UART_SetEmpty();
00224
00225             if (UART_IsDREIntEnabled())
00226             {
00227                 DEBUG_PRINT("DRE Interrupt\n");
00228                 AVR_InterruptCandidate( 0x13 );
00229             }
00230         }
00231         else
00232         {
00233             TXB = stCPU.pstRAM->stRegisters.UDR0;
00234             bTSR_Empty = false;
00235             bUDR_Empty = false;
00236             UART_ClearEmpty();
00237         }
00238     }
00239     else
00240     {
00241         DEBUG_PRINT("Disabled...\n");
00242     }
00243 }
00244
00245 //-------------------------------------------------------------------------
00246 static void UART_WriteUCSR0A( uint8_t u8Value_)
00247 {
00248     DEBUG_PRINT("UART Write UCSR0A...\n");
00249     uint8_t u8Reg = stCPU.pstRAM->stRegisters.UCSR0A.r;
00250     if (u8Value_ & 0x40) // TXC was set explicitly -- clear it in the SR.
00251     {
00252         u8Reg &= ~0x40;
00253     }
00254     u8Reg &= ~(0xBC);
00255
00256     stCPU.pstRAM->stRegisters.UCSR0A.r |= u8Reg;
00257
00258 }
00259
00260 //-------------------------------------------------------------------------
00261 static void UART_UpdateInterruptFlags(void)
00262 {
00263     //DEBUG_PRINT("Check UART Interrupts\n");
00264     if (UART_IsTxIntEnabled())
00265     {
00266         if (UART_IsTxComplete())
00267         {
00268             DEBUG_PRINT("TXC Interrupt\n");
00269             AVR_InterruptCandidate( 0x14 );
00270         }
00271         else
00272         {
00273             AVR_ClearCandidate( 0x14 );
00274         }
00275     }
00276     if (UART_IsDREIntEnabled())
```

```
00277     {
00278         if( UART_IsEmpty())
00279         {
00280             DEBUG_PRINT("DRE Interrupt\n");
00281             AVR_InterruptCandidate( 0x13 );
00282         }
00283         else
00284         {
00285             AVR_ClearCandidate( 0x13 );
00286         }
00287     }
00288     if (UART_IsRxIntEnabled())
00289     {
00290         if (UART_IsRxComplete())
00291         {
00292             printf("RXC Interrupt\n");
00293             AVR_InterruptCandidate( 0x12 );
00294         }
00295         else
00296         {
00297             AVR_ClearCandidate( 0x12 );
00298         }
00299     }
00300 }
00301
00302 //-------------------------------------------------------------------------
00303 static void UART_WriteUCSR0B( uint8_t u8Value_)
00304 {
00305     DEBUG_PRINT("Write UCRS0B\n");
00306     stCPU.pstRAM->stRegisters.UCSR0B.r = u8Value_;
00307     UART_UpdateInterruptFlags();
00308 }
00309
00310 //-------------------------------------------------------------------------
00311 static void UART_WriteUCSR0C( uint8_t u8Value_)
00312 {
00313     DEBUG_PRINT("Write UCRS0C\n");
00314     stCPU.pstRAM->stRegisters.UCSR0C.r == u8Value_;
00315 }
00316
00317 //-------------------------------------------------------------------------
00318 static void UART_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
00319 {
00320     DEBUG_PRINT("UART Write: %2X=%2X\n", ucAddr_, ucValue_ );
00321     switch (ucAddr_)
00322     {
00323     case 0xC0:  //UCSR0A
00324         UART_WriteUCSR0A( ucValue_ );
00325         break;
00326     case 0xC1:  //UCSR0B
00327         UART_WriteUCSR0B( ucValue_ );
00328         break;
00329     case 0xC2:  //UCSR0C
00330         UART_WriteUCSR0C( ucValue_ );
00331         break;
00332     case 0xC3:  // NA.
00333         break;
00334     case 0xC4:  //UBRR0L
00335     case 0xC5:  //UBRR0H
00336         DEBUG_PRINT("Write UBRR0x\n");
00337         stCPU.pstRAM->au8RAM[ ucAddr_ ] = ucValue_;
00338         UART_WriteBaudReg();
00339         break;
00340     case 0xC6:  //UDR0
00341         DEBUG_PRINT("Write UDR0\n");
00342         stCPU.pstRAM->au8RAM[ ucAddr_ ] = ucValue_;
00343         UART_WriteDataReg();
00344         break;
00345     default:
00346         break;
00347     }
00348 }
00349
00350 //-------------------------------------------------------------------------
00351 static void UART_TxClock(void *context_ )
00352 {
00353     //DEBUG_PRINT("TX clock...\n");
00354     if (UART_IsTxEnabled() && u32TxTicksRemaining)
00355     {
00356         DEBUG_PRINT("Countdown %d ticks remain\n", u32TxTicksRemaining);
00357         u32TxTicksRemaining--;
00358         if (!u32TxTicksRemaining)
00359         {
00360             // Local echo of the freshly "shifted out" data to the terminal
00361             Echo_Tx();
00362
00363             // If there's something queued in the TXB, reload the TSR
```

```
00364             // register, flag the UDR as empty, and TSR as full.
00365             if (!bUDR_Empty)
00366             {
00367                 TSR = TXB;
00368                 TXB = 0;
00369                 bUDR_Empty = true;
00370                 bTSR_Empty = false;
00371 
00372                 UART_SetEmpty();
00373 
00374                 if (UART_IsDREIntEnabled())
00375                 {
00376                     DEBUG_PRINT("DRE Interrupt\n");
00377                     AVR_InterruptCandidate( 0x13 );
00378                 }
00379             }
00380             // Nothing pending in the TXB?  Flag the TSR as empty, and
00381             // set the "Transmit complete" flag in the register.
00382             else
00383             {
00384                 TXB = 0;
00385                 TSR = 0;
00386                 bTSR_Empty = true;
00387 
00388                 UART_TxComplete();
00389                 if (UART_IsTxIntEnabled())
00390                 {
00391                     DEBUG_PRINT("TXC Interrupt\n");
00392                     AVR_InterruptCandidate( 0x14 );
00393                 }
00394             }
00395         }
00396     }
00397 }
00398 
00399 //-----------------------------------------------------------------------------
00400 static void UART_RxClock(void *context_ )
00401 {
00402     if (UART_IsRxEnabled() && u32RxTicksRemaining)
00403     {
00404         u32RxTicksRemaining--;
00405         if (!u32RxTicksRemaining)
00406         {
00407             // Local echo of the freshly "shifted in" data to the terminal
00408             Echo_Rx();
00409 
00410             // Move data from receive shift register into the receive buffer
00411             RXB = RSR;
00412             RSR = 0;
00413 
00414             // Set the RX Complete flag
00415             UART_RxComplete();
00416             if (UART_IsRxIntEnabled())
00417             {
00418                 DEBUG_PRINT("RXC Interrupt\n");
00419                 AVR_InterruptCandidate( 0x12 );
00420             }
00421         }
00422     }
00423 }
00424 //-----------------------------------------------------------------------------
00425 static void UART_Clock(void *context_ )
00426 {
00427     // Handle Rx and TX clocks.
00428     UART_TxClock(context_);
00429     UART_RxClock(context_);
00430 }
00431 
00432 //-----------------------------------------------------------------------------
00433 AVRPeripheral stUART =
00434 {
00435     UART_Init,
00436     UART_Read,
00437     UART_Write,
00438     UART_Clock,
00439     0,
00440     0xC0,
00441     0xC6
00442 };
```

4.79 mega_uart.h File Reference

ATMega UART implementation.

```
#include "avr_peripheral.h"
```

Variables

- AVRPeripheral **stUART**

4.79.1 Detailed Description

ATMega UART implementation.

Definition in file mega_uart.h.

4.80 mega_uart.h

```
00001 /*************************************************************************
00002  *      (     (                       (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\     (   (  (()/(   | -- [ Funkenstein ] ------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] --------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \    \ V /  |   /   | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\    \_/   |_|_\   |
00010  *                                         | "Yeah, it does Arduino..."
00011  * ----------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *************************************************************************/
00021 #ifndef __MEGA_UART_H__
00022 #define __MEGA_UART_H__
00023
00024 #include "avr_peripheral.h"
00025
00026 extern AVRPeripheral stUART;
00027
00028 #endif //__MEGA_UART_H__
```

4.81 options.c File Reference

Module for managing command-line options.

```
#include "emu_config.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdint.h>
```

Data Structures

- struct Option_t

  Local data structure used to define a command-line option.

Enumerations

- enum OptionIndex_t {
  OPTION_VARIANT, **OPTION_FREQ**, **OPTION_HEXFILE**, **OPTION_DEBUG**,
  OPTION_SILENT, **OPTION_DISASM**, **OPTION_TRACE**, OPTION_NUM }

  Enumerated type specifcying the known command-line options accepted by flAVR.

Functions

- static void Options_SetDefaults (void)

  Options_SetDefaults.
- const char * Options_GetByName (const char *szAttribute_)

  Options_GetByName.
- static uint16_t Options_ParseElement (int start_, int argc_, char **argv_)

  Options_ParseElement.
- static void Options_Parse (int argc_, char **argv_)

  Options_Parse.
- void Options_Init (int argc_, char **argv_)

  Options_Init.

Variables

- static Option_t astAttributes [OPTION_NUM]

  Table of available commandline options.

4.81.1 Detailed Description

Module for managing command-line options.

Definition in file options.c.

4.81.2 Enumeration Type Documentation

4.81.2.1 enum OptionIndex_t

Enumerated type specifcying the known command-line options accepted by flAVR.

Enumerator

OPTION_NUM Total count of command-line options supported.

Definition at line 43 of file options.c.

4.81.3 Function Documentation

4.81.3.1 const char* Options_GetByName (const char * *szAttribute_*)

Options_GetByName.

Return the parameter value associated with an option attribute.

Parameters

*szAttribute_*	Name of the attribute to look up

Returns

Pointer to the attribute string, or NULL if attribute is invalid, or parameter has not been set.

Definition at line 86 of file options.c.

4.81.3.2 void Options_Init (int *argc_*, char ** *argv_*)

Options_Init.

Initialize command-line options for the emulator based on argc/argv input.

Parameters

*argc_*	argc, passed in from main
*argv_*	argv, passed in from main

Definition at line 181 of file options.c.

4.81.3.3 static void Options_Parse (int *argc_*, char ** *argv_*) [static]

Options_Parse.

Parse the commandline optins, seeding the array of known parameters with the values specified by the user on the commandline

Parameters

*argc_*	Number of arguments
*argv_*	Argument vector, passed from main().

Definition at line 170 of file options.c.

4.81.3.4 static uint16_t Options_ParseElement (int *start_*, int *argc_*, char ** *argv_*) [static]

Options_ParseElement.

Parse out the next commandline option, starting with argv[start_]. Modifies the values stored in the local astAttributes table.

Parameters

*start_*	Starting index
*argc_*	Total number of arguments
*argv_*	Command-line argument vector

Returns

The next index to process

Definition at line 113 of file options.c.

4.81.3.5 static void Options_SetDefaults (void) [static]

Options_SetDefaults.

Set certain options to default implicit values, in case none are specific from the commandline.

Definition at line 80 of file options.c.

4.81.4 Variable Documentation

4.81.4.1 Option_t astAttributes[OPTION_NUM] [static]

Initial value:

```
=
{
    {"--variant", NULL, false },
    {"--freq", NULL, false },
    {"--hexfile", NULL, false },
    {"--debug", NULL, true },
    {"--silent", NULL, true },
    {"--disasm", NULL, true },
    {"--trace", NULL, true }
}
```

Table of available commandline options.

Order must match enumeration defined above.

Definition at line 61 of file options.c.

4.82 options.c

```
00001 /*****************************************************************************
00002  *      (      (                       (      |
00003  *    )\ )   )\ )     (              )\ )    |
00004  *   (()/(  (()/(     )\      (    (  (()/(    | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ----------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] ------------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] ------------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                         | "Yeah, it does Arduino..."
00011  * ----------------------------------------+-------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #include "emu_config.h"
00022 #include <stdio.h>
00023 #include <string.h>
00024 #include <stdlib.h>
00025 #include <stdint.h>
00026
00027 //-----------------------------------------------------------------------------
00031 typedef struct
00032 {
00033     const char *szAttribute;
00034     char *szParameter;
00035     bool bStandalone;
00036 } Option_t;
00037
00038 //-----------------------------------------------------------------------------
00043 typedef enum
00044 {
00045     OPTION_VARIANT,
00046     OPTION_FREQ,
00047     OPTION_HEXFILE,
00048     OPTION_DEBUG,
00049     OPTION_SILENT,
00050     OPTION_DISASM,
00051     OPTION_TRACE,
00052 //-- New options go here ^^^
00053     OPTION_NUM
00054 } OptionIndex_t;
00055
00056 //-----------------------------------------------------------------------------
00061 static Option_t astAttributes[OPTION_NUM] =
00062 {
00063     {"--variant", NULL, false },
00064     {"--freq", NULL, false },
00065     {"--hexfile", NULL, false },
00066     {"--debug", NULL, true },
00067     {"--silent", NULL, true },
00068     {"--disasm", NULL, true },
00069     {"--trace", NULL, true }
00070 };
00071
00072 //-----------------------------------------------------------------------------
00080 static void Options_SetDefaults( void )
00081 {
00082     astAttributes[ OPTION_VARIANT ].szParameter  = strdup( "atmega328p" );
00083     astAttributes[ OPTION_FREQ ].szParameter     = strdup( "16000000" );
00084 }
00085 //-----------------------------------------------------------------------------
00086 const char *Options_GetByName (const char *szAttribute_)
00087 {
00088     uint16_t j;
00089
00090     // linear search for the correct option value.
00091     for (j = 0; j < OPTION_NUM; j++)
00092     {
00093         if (0 == strcmp(astAttributes[j].szAttribute, szAttribute_))
00094         {
00095             return (const char*)astAttributes[j].szParameter;
00096         }
00097     }
00098     return NULL;
00099 }
```

```
00100
00101 //-----------------------------------------------------------------------------
00113 static uint16_t Options_ParseElement( int start_, int argc_, char **argv_ )
00114 {
00115     // Parse out specific option parameter data for a given option attribute
00116     uint16_t i = start_;
00117     uint16_t j;
00118
00119     while (i < argc_)
00120     {
00121         // linear search for the correct option value.
00122         for (j = 0; j < OPTION_NUM; j++)
00123         {
00124             if (0 == strcmp(astAttributes[j].szAttribute, argv_[i]))
00125             {
00126                 // Match - is the option stand-alone, or does it take a parameter?
00127                 if (astAttributes[j].bStandalone)
00128                 {
00129                     // Standalone argument, auto-seed a "1" value for the parameter to
00130                     // indicate that the option was set on the commandline
00131                     astAttributes[j].szParameter = strdup("1");
00132                     return 1;
00133                 }
00134
00135                 // ensure the user provided a parameter for this attribute
00136                 if (i + 1 >= argc_)
00137                 {
00138                     fprintf( stderr, "Error: Paramter expected for attribute %s", argv_[i] );
00139                     exit(-1);
00140                 }
00141                 // Check to see if a parameter has already been set; if so, free the existing value
00142                 if (NULL != astAttributes[j].szParameter)
00143                 {
00144                     free(astAttributes[j].szParameter );
00145                 }
00146                 // fprintf( stderr, "Match: argv[i]=%s, argv[i+1]=%s\n", argv_[i], argv_[i+1] );
00147                 astAttributes[j].szParameter = strdup(argv_[i+1]);
00148             }
00149         }
00150         // Read attribute + parameter combo, 2 tokens
00151         return 2;
00152     }
00153
00154     // Unknown option - 1 token
00155     fprintf( stderr, "WARN: Invalid option \"%s\"", argv_[i] );
00156
00157     return 1;
00158 }
00159
00160 //-----------------------------------------------------------------------------
00170 static void Options_Parse(int argc_, char **argv_ )
00171 {
00172     uint16_t i = 1;
00173     while (i < argc_)
00174     {
00175         // Parse out token from the command line array.
00176         i += Options_ParseElement( i, argc_, argv_ );
00177     }
00178 }
00179
00180 //-----------------------------------------------------------------------------
00181 void Options_Init( int argc_, char **argv_ )
00182 {
00183     Options_SetDefaults();
00184     Options_Parse( argc_, argv_ );
00185 }
00186 //-----------------------------------------------------------------------------
```

4.83 trace_buffer.c File Reference

Implements a circular buffer containing a history of recently executed instructions, along with core register context for each.

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "trace_buffer.h"
#include "emu_config.h"
#include "avr_disasm.h"
#include "avr_op_decode.h"
```

Functions

- void TraceBuffer_Init (TraceBuffer_t *pstTraceBuffer_)

  TraceBuffer_Init Initialize a tracebuffer prior to use.
- void TraceBuffer_StoreFromCPU (TraceBuffer_t *pstTraceBuffer_)

  TraceBuffer_StoreFromCPU Store a trace element in the tracebuffer at its current head index.
- void TraceBuffer_LoadElement (TraceBuffer_t *pstTraceBuffer_, TraceElement_t *pstElement_, uint32_t u32-Element_)

  TraceBuffer_LoadElement Load an element from the tracebuffer into a a specified.
- void TraceBuffer_PrintElement (TraceElement_t *pstElement_, TracePrintFormat_t eFormat_)

  TraceBuffer_PrintElement Print a single element from a tracebuffer to standard output.
- void TraceBuffer_Print (TraceBuffer_t *pstTraceBuffer_, TracePrintFormat_t eFormat_)

  TraceBuffer_Print Print the raw contents of a tracebuffer to standard output.

4.83.1 Detailed Description

Implements a circular buffer containing a history of recently executed instructions, along with core register context for each.

Definition in file trace_buffer.c.

4.83.2 Function Documentation

4.83.2.1 void TraceBuffer_Init (TraceBuffer_t * *pstTraceBuffer_*)

TraceBuffer_Init Initialize a tracebuffer prior to use.

Parameters

*pstTraceBuffer_*	Pointer to the tracebuffer to initialize

Definition at line 35 of file trace_buffer.c.

4.83.2.2 void TraceBuffer_LoadElement (TraceBuffer_t * *pstTraceBuffer_*, TraceElement_t * *pstElement_*, uint32_t *u32Element_*)

TraceBuffer_LoadElement Load an element from the tracebuffer into a a specified.

Parameters

*pstTraceBuffer_*	Pointer to a tracebuffer to load from

*pstElement_*	Pointer to a trace element structure to store data into
*u32Element_*	Index of the element in the tracebuffer to read

Definition at line 67 of file trace_buffer.c.

4.83.2.3 void TraceBuffer_Print (TraceBuffer_t * *pstTraceBuffer_*, TracePrintFormat_t *eFormat_*)

TraceBuffer_Print Print the raw contents of a tracebuffer to standard output.

Parameters

*pstTraceBuffer_*	Pointer to the tracebuffer to print
*eFormat_*	Formatting type for the print

Definition at line 118 of file trace_buffer.c.

4.83.2.4 void TraceBuffer_PrintElement (TraceElement_t * *pstElement_*, TracePrintFormat_t *eFormat_*)

TraceBuffer_PrintElement Print a single element from a tracebuffer to standard output.

This prints core registers and addresses.

Parameters

*pstElement_*	Pointer to the trace element to print •
*eFormat_*	Formatting type for the print

Definition at line 75 of file trace_buffer.c.

4.83.2.5 void TraceBuffer_StoreFromCPU (TraceBuffer_t * *pstTraceBuffer_*)

TraceBuffer_StoreFromCPU Store a trace element in the tracebuffer at its current head index.

Parameters

*pstTraceBuffer_*	Pointer to the tracebuffer to store into

Definition at line 41 of file trace_buffer.c.

4.84 trace_buffer.c

```
00001 /*****************************************************************************
00002  *      (     (                       (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(    )\     (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\   |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00023 #include <stdint.h>
00024 #include <stdio.h>
00025 #include <stdlib.h>
00026 #include <string.h>
00027
00028 #include "trace_buffer.h"
00029 #include "emu_config.h"
00030
```

```
00031 #include "avr_disasm.h"
00032 #include "avr_op_decode.h"
00033
00034 //-----------------------------------------------------------------------------
00035 void TraceBuffer_Init( TraceBuffer_t *pstTraceBuffer_ )
00036 {
00037     memset( pstTraceBuffer_, 0, sizeof(*pstTraceBuffer_) );
00038 }
00039
00040 //-----------------------------------------------------------------------------
00041 void TraceBuffer_StoreFromCPU( TraceBuffer_t *pstTraceBuffer_  )
00042 {
00043     TraceElement_t *pstTraceElement = &pstTraceBuffer_->astTraceStep[ pstTraceBuffer_->
      u32Index ];
00044
00045     // Manually copy over whatever elements we need to
00046     pstTraceElement->u64Counter    = stCPU.u64InstructionCount;
00047     pstTraceElement->u64CycleCount = stCPU.u64CycleCount;
00048     pstTraceElement->u16PC         = stCPU.u16PC;
00049     pstTraceElement->u16SP         = ((uint16_t)(stCPU.pstRAM->stRegisters.SPH.r) << 8) |
00050                                       (uint16_t)(stCPU.pstRAM->stRegisters.SPL.r);
00051
00052     pstTraceElement->u16OpCode     = stCPU.pu16ROM[ stCPU.u16PC ];
00053     pstTraceElement->u8SR          = stCPU.pstRAM->stRegisters.SREG.r;
00054
00055     // Memcpy the core registers in one chunk
00056     memcpy(&(pstTraceElement->stCoreRegs), &(stCPU.pstRAM->stRegisters.CORE_REGISTERS), sizeof(
      pstTraceElement->stCoreRegs));
00057
00058     // Update the index of the write buffer
00059     pstTraceBuffer_->u32Index++;
00060     if (pstTraceBuffer_->u32Index >= CONFIG_TRACEBUFFER_SIZE)
00061     {
00062         pstTraceBuffer_->u32Index = 0;
00063     }
00064 }
00065
00066 //-----------------------------------------------------------------------------
00067 void TraceBuffer_LoadElement( TraceBuffer_t *pstTraceBuffer_,
      TraceElement_t *pstElement_, uint32_t u32Element_ )
00068 {
00069     TraceElement_t *pstSourceElement = &pstTraceBuffer_->astTraceStep[ pstTraceBuffer_->
      u32Index ];
00070
00071     memcpy(pstElement_, pstSourceElement, sizeof(*pstElement_));
00072 }
00073
00074 //-----------------------------------------------------------------------------
00075 void TraceBuffer_PrintElement( TraceElement_t *pstElement_,
      TracePrintFormat_t eFormat_  )
00076 {
00077     printf( "[%08d] 0x%04X:0x%04X: ",
00078             pstElement_->u64Counter, pstElement_->u16PC, pstElement_->u16OpCode );
00079     if (eFormat_ & TRACE_PRINT_DISASSEMBLY)
00080     {
00081         uint16_t u16TempPC = stCPU.u16PC;
00082         stCPU.u16PC = pstElement_->u16PC;
00083
00084         AVR_Opcode pfOp = AVR_Disasm_Function( pstElement_->u16OpCode );
00085
00086         AVR_Decode( pstElement_->u16OpCode );
00087         pfOp();
00088
00089         stCPU.u16PC = u16TempPC;
00090     }
00091
00092     if (eFormat_ & TRACE_PRINT_COMPACT)
00093     {
00094         printf( "%04X ", pstElement_->u16SP );
00095
00096         int i;
00097         for (i = 0; i < 32; i++)
00098         {
00099             printf( "%02X ", pstElement_->stCoreRegs.r[i] );
00100         }
00101         printf( "\n" );
00102     }
00103     if (eFormat_ & TRACE_PRINT_REGISTERS)
00104     {
00105         uint8_t i;
00106         for (i = 0; i < 32; i++)
00107         {
00108             printf( "[R%02d] = 0x%02X\n", i, pstElement_->stCoreRegs.r[i] );
00109         }
00110         printf("[SP]   = 0x%04X\n", pstElement_->u16SP );
00111         printf("[PC]   = 0x%04X\n", (uint16_t)pstElement_->u16PC );
00112         printf("[SREG]= 0x%02X", pstElement_->u8SR );
```

```
00113         printf( "\n" );
00114     }
00115 }
00116
00117 //-----------------------------------------------------------------------------
00118 void TraceBuffer_Print( TraceBuffer_t *pstTraceBuffer_, TracePrintFormat_t
      eFormat_ )
00119 {
00120     int i;
00121     for (i = pstTraceBuffer_->u32Index; i < CONFIG_TRACEBUFFER_SIZE; i++)
00122     {
00123         TraceBuffer_PrintElement(&pstTraceBuffer_->astTraceStep[i], eFormat_ );
00124     }
00125     for (i = 0; i < pstTraceBuffer_->u32Index; i++)
00126     {
00127         TraceBuffer_PrintElement(&pstTraceBuffer_->astTraceStep[i], eFormat_ );
00128     }
00129 }
```

4.85 trace_buffer.h File Reference

Implements a circular buffer containing a history of recently executed instructions, along with core register context for each.

```
#include <stdint.h>
#include "emu_config.h"
#include "avr_cpu.h"
```

Data Structures

- struct TraceElement_t
- struct TraceBuffer_t

Enumerations

- enum **TracePrintFormat_t** { **TRACE_PRINT_COMPACT** = 1, **TRACE_PRINT_REGISTERS** = 2, **TRACE_PRINT_DISASSEMBLY** = 4 }

Functions

- void TraceBuffer_Init (TraceBuffer_t *pstTraceBuffer_)

  TraceBuffer_Init Initialize a tracebuffer prior to use.
- void TraceBuffer_StoreFromCPU (TraceBuffer_t *pstTraceBuffer_)

  TraceBuffer_StoreFromCPU Store a trace element in the tracebuffer at its current head index.
- void TraceBuffer_LoadElement (TraceBuffer_t *pstTraceBuffer_, TraceElement_t *pstElement_, uint32_t u32Element_)

  TraceBuffer_LoadElement Load an element from the tracebuffer into a a specified.
- void TraceBuffer_PrintElement (TraceElement_t *pstElement_, TracePrintFormat_t eFormat_)

  TraceBuffer_PrintElement Print a single element from a tracebuffer to standard output.
- void TraceBuffer_Print (TraceBuffer_t *pstTraceBuffer_, TracePrintFormat_t eFormat_)

  TraceBuffer_Print Print the raw contents of a tracebuffer to standard output.

4.85.1 Detailed Description

Implements a circular buffer containing a history of recently executed instructions, along with core register context for each.

Definition in file trace_buffer.h.

4.85.2 Function Documentation

4.85.2.1 void TraceBuffer_Init (TraceBuffer_t * *pstTraceBuffer_*)

TraceBuffer_Init Initialize a tracebuffer prior to use.

Parameters

*pstTraceBuffer_*	Pointer to the tracebuffer to initialize

Definition at line 35 of file trace_buffer.c.

4.85.2.2 void TraceBuffer_LoadElement (TraceBuffer_t * *pstTraceBuffer_*, TraceElement_t * *pstElement_*, uint32_t *u32Element_*)

TraceBuffer_LoadElement Load an element from the tracebuffer into a a specified.

Parameters

*pstTraceBuffer_*	Pointer to a tracebuffer to load from
*pstElement_*	Pointer to a trace element structure to store data into
*u32Element_*	Index of the element in the tracebuffer to read

Definition at line 67 of file trace_buffer.c.

4.85.2.3 void TraceBuffer_Print (TraceBuffer_t * *pstTraceBuffer_*, TracePrintFormat_t *eFormat_*)

TraceBuffer_Print Print the raw contents of a tracebuffer to standard output.

Parameters

*pstTraceBuffer_*	Pointer to the tracebuffer to print
*eFormat_*	Formatting type for the print

Definition at line 118 of file trace_buffer.c.

4.85.2.4 void TraceBuffer_PrintElement (TraceElement_t * *pstElement_*, TracePrintFormat_t *eFormat_*)

TraceBuffer_PrintElement Print a single element from a tracebuffer to standard output.

This prints core registers and addresses.

Parameters

*pstElement_*	Pointer to the trace element to print •
*eFormat_*	Formatting type for the print

Definition at line 75 of file trace_buffer.c.

4.85.2.5 void TraceBuffer_StoreFromCPU (TraceBuffer_t * *pstTraceBuffer_*)

TraceBuffer_StoreFromCPU Store a trace element in the tracebuffer at its current head index.

Parameters

*pstTraceBuffer_*	Pointer to the tracebuffer to store into

Definition at line 41 of file trace_buffer.c.

4.86 trace_buffer.h

```
00001 /*****************************************************************************
00002  *      (     (                       (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\     (   (  (()/(   | -- [ Funkenstein ] ------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] -------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_))(_))    | -- [ AVR ] --------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\   |
00010  *                                        | "Yeah, it does Arduino..."
00011  * ---------------------------------------+-------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00023 #ifndef __TRACE_BUFFER_H__
00024 #define __TRACE_BUFFER_H__
00025
00026 #include <stdint.h>
00027
00028 #include "emu_config.h"
00029 #include "avr_cpu.h"
00030
00031 //-----------------------------------------------------------------------------
00032 typedef struct
00033 {
00034     uint64_t    u64Counter;
00035     uint64_t    u64CycleCount;
00036     uint16_t    u16OpCode;
00037     uint16_t    u16PC;
00038     uint16_t    u16SP;
00039     uint8_t     u8SR;
00040
00041     AVR_CoreRegisters stCoreRegs;
00042
00043 } TraceElement_t;
00044
00045 //-----------------------------------------------------------------------------
00046 typedef struct
00047 {
00048     TraceElement_t  astTraceStep[ CONFIG_TRACEBUFFER_SIZE ];
00049     uint32_t        u32Index;
00050 } TraceBuffer_t;
00051
00052
00053 //-----------------------------------------------------------------------------
00054 typedef enum
00055 {
00056     TRACE_PRINT_COMPACT     = 1,
00057     TRACE_PRINT_REGISTERS   = 2,
00058     TRACE_PRINT_DISASSEMBLY = 4
00059 } TracePrintFormat_t;
00060
00061 //-----------------------------------------------------------------------------
00067 void TraceBuffer_Init( TraceBuffer_t *pstTraceBuffer_ );
00068
00069 //-----------------------------------------------------------------------------
00077 void TraceBuffer_StoreFromCPU( TraceBuffer_t *pstTraceBuffer_ );
00078
00079 //-----------------------------------------------------------------------------
00090 void TraceBuffer_LoadElement( TraceBuffer_t *pstTraceBuffer_,
      TraceElement_t *pstElement_, uint32_t u32Element_ );
00091
00092 //-----------------------------------------------------------------------------
00099 void TraceBuffer_PrintElement( TraceElement_t *pstElement_,
      TracePrintFormat_t eFormat_ );
00100
00101 //-----------------------------------------------------------------------------
00110 void TraceBuffer_Print( TraceBuffer_t *pstTraceBuffer_, TracePrintFormat_t
      eFormat_ );
00111
00112 #endif
```

4.87 variant.c File Reference

Module containing a table of device variants supported by flavr.

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "variant.h"
```

Macros

- #define **ADD_CAPABILITY** 0xFE, 0xEF
- #define **IO_REGISTER_RANGE** 0xFD, 0xDF
- #define **AVR_HAS_RAMP_Z** 0x07
- #define **AVR_HAS_EIND** 0x08
- #define **AVR_HAS_UART0** 0x09
- #define **AVR_HAS_UART1** 0x0A
- #define **AVR_HAS_TIMER0_8BIT** 0x0B
- #define **AVR_HAS_TIMER0_16BIT** 0x0C
- #define **AVR_HAS_TIMER1_8BIT** 0x0D
- #define **AVR_HAS_TIMER1_16BIT** 0x0E
- #define **AVR_HAS_TIMER2_8BIT** 0x0F
- #define **AVR_HAS_TIMER2_16BIT** 0x10
- #define **KB** ∗ (1024)

Functions

- const AVR_Variant_t ∗ Variant_GetByName (const char ∗szName_)

  Variant_GetByName.

Variables

- static AVR_Variant_t **astVariants** []

4.87.1 Detailed Description

Module containing a table of device variants supported by flavr.

Definition in file variant.c.

4.87.2 Function Documentation

4.87.2.1 const AVR_Variant_t∗ Variant_GetByName (const char ∗ *szName_*)

Variant_GetByName.

Lookup a processor variant based on its name, and return a pointer to a matching variant string on successful match.

Parameters

*szName_*	String containing a varaint name to check against (i.e. "atmega328p")

Returns

Pointer to a CPU Variant struct on successful match, NULL on failure.

Definition at line 66 of file variant.c.

4.87.3 Variable Documentation

4.87.3.1 AVR_Variant_t astVariants[] [static]

Initial value:

```
=
{
    { "atmega328p",  2 KB,   32 KB, 1 KB,    NULL },
    { "atmega328",   2 KB,   32 KB, 1 KB,    NULL },
    { "atmega168pa", 1 KB,   16 KB, 0.5 KB,  NULL },
    { "atmega168",   1 KB,   16 KB, 0.5 KB,  NULL },
    { "atmega88pa",  1 KB,   8 KB,  0.5 KB,  NULL },
    { "atmega88",    1 KB,   8 KB,  0.5 KB,  NULL },
    { "atmega44pa",  0.5 KB, 4 KB,  0.25 KB, NULL },
    { "atmega44",    0.5 KB, 4 KB,  0.25 KB, NULL },
    { 0 }
}
```

Definition at line 52 of file variant.c.

4.88 variant.c

```
00001 /*****************************************************************************
00002  *      (     (                          (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(     )\      (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\   )\  /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\    |
00010  *                                         | "Yeah, it does Arduino..."
00011  * ----------------------------------------+-------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #include <stdint.h>
00022 #include <stdio.h>
00023 #include <stdlib.h>
00024 #include <string.h>
00025
00026 #include "variant.h"
00027
00028 //-----------------------------------------------------------------------------
00029 #define ADD_CAPABILITY                  0xFE, 0xEF
00030
00031 #define IO_REGISTER_RANGE               0xFD, 0xDF
00032
00033 #define AVR_HAS_RAMP_Z                  0x07
00034 #define AVR_HAS_EIND                    0x08
00035
00036 #define AVR_HAS_UART0                   0x09
00037 #define AVR_HAS_UART1                   0x0A
00038
00039 #define AVR_HAS_TIMER0_8BIT             0x0B
00040 #define AVR_HAS_TIMER0_16BIT            0x0C
00041
00042 #define AVR_HAS_TIMER1_8BIT             0x0D
00043 #define AVR_HAS_TIMER1_16BIT            0x0E
00044
00045 #define AVR_HAS_TIMER2_8BIT             0x0F
00046 #define AVR_HAS_TIMER2_16BIT            0x10
```

```
00047
00048 //------------------------------------------------------------------------------
00049 #define KB  * (1024)
00050
00051 //------------------------------------------------------------------------------
00052 static AVR_Variant_t astVariants[] =
00053 {
00054     { "atmega328p",  2 KB,   32 KB, 1 KB,    NULL },
00055     { "atmega328",   2 KB,   32 KB, 1 KB,    NULL },
00056     { "atmega168pa", 1 KB,   16 KB, 0.5 KB,  NULL },
00057     { "atmega168",   1 KB,   16 KB, 0.5 KB,  NULL },
00058     { "atmega88pa",  1 KB,   8 KB,  0.5 KB,  NULL },
00059     { "atmega88",    1 KB,   8 KB,  0.5 KB,  NULL },
00060     { "atmega44pa",  0.5 KB, 4 KB,  0.25 KB, NULL },
00061     { "atmega44",    0.5 KB, 4 KB,  0.25 KB, NULL },
00062     { 0 }
00063 };
00064
00065 //------------------------------------------------------------------------------
00066 const AVR_Variant_t *Variant_GetByName( const char *szName_ )
00067 {
00068     AVR_Variant_t *pstVariant = astVariants;
00069     while (pstVariant->szName)
00070     {
00071         if (0 == strcmp(pstVariant->szName, szName_ ) )
00072         {
00073             return pstVariant;
00074         }
00075         pstVariant++;
00076     }
00077     return NULL;
00078 }
00079
```

4.89 variant.h File Reference

Module containing a lookup table of device variants supported by flavr.

Data Structures

- struct AVR_Variant_t

  This struct contains the information necessary to effectively describe an AVR Microcontroller variant among the rest of the code.

Macros

- #define **ADD_CAPABILITY** 0xFE, 0xEF
- #define **IO_REGISTER_RANGE** 0xFD, 0xDF
- #define **AVR_HAS_RAMP_Z** 0x07
- #define **AVR_HAS_EIND** 0x08
- #define **AVR_HAS_UART0** 0x09
- #define **AVR_HAS_UART1** 0x0A
- #define **AVR_HAS_TIMER0_8BIT** 0x0B
- #define **AVR_HAS_TIMER0_16BIT** 0x0C
- #define **AVR_HAS_TIMER1_8BIT** 0x0D
- #define **AVR_HAS_TIMER1_16BIT** 0x0E
- #define **AVR_HAS_TIMER2_8BIT** 0x0F
- #define **AVR_HAS_TIMER2_16BIT** 0x10

Functions

- const AVR_Variant_t ∗ Variant_GetByName (const char ∗szName_)

  Variant_GetByName.

4.89.1 Detailed Description

Module containing a lookup table of device variants supported by flavr.

Definition in file variant.h.

4.89.2 Function Documentation

4.89.2.1 const AVR_Variant_t* Variant_GetByName (const char * *szName_*)

Variant_GetByName.

Lookup a processor variant based on its name, and return a pointer to a matching variant string on successful match.

Parameters

*szName_*	String containing a varaint name to check against (i.e. "atmega328p")

Returns

Pointer to a CPU Variant struct on successful match, NULL on failure.

Definition at line 66 of file variant.c.

4.90 variant.h

```
00001 /*****************************************************************************
00002  *      (      (                       (      |
00003  *    )\ )   )\ )     (              )\ )    |
00004  *   (()/(  (()/(     )\      (    (  (()/(    | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\   )\   /(_))      | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ----------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \     | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /     | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\     |
00010  *                                          | "Yeah, it does Arduino..."
00011  * -----------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *****************************************************************************/
00021 #ifndef __VARIANT_H__
00022 #define __VARIANT_H__
00023 //---------------------------------------------------------------------------
00024 #define ADD_CAPABILITY                  0xFE, 0xEF
00025
00026 #define IO_REGISTER_RANGE               0xFD, 0xDF
00027
00028 #define AVR_HAS_RAMP_Z                  0x07
00029 #define AVR_HAS_EIND                    0x08
00030
00031 #define AVR_HAS_UART0                   0x09
00032 #define AVR_HAS_UART1                   0x0A
00033
00034 #define AVR_HAS_TIMER0_8BIT             0x0B
00035 #define AVR_HAS_TIMER0_16BIT            0x0C
00036
00037 #define AVR_HAS_TIMER1_8BIT             0x0D
00038 #define AVR_HAS_TIMER1_16BIT            0x0E
00039
00040 #define AVR_HAS_TIMER2_8BIT             0x0F
00041 #define AVR_HAS_TIMER2_16BIT            0x10
00042
00043 //---------------------------------------------------------------------------
00048 typedef struct
00049 {
00050     const char  *szName;
00051
00052     uint32_t    u32RAMSize;
00053     uint32_t    u32ROMSize;
00054     uint32_t    u32EESize;
00055
00056     const uint8_t *u8Descriptors;
```

```
00057
00058 } AVR_Variant_t;
00059
00060 //-------------------------------------------------------------------------
00072 const AVR_Variant_t *Variant_GetByName( const char *szName_ );
00073
00074 #endif
```

4.91 watchpoint.c File Reference

Implements data watchpoints for debugging running programs based on reads/writes to a given memory address.

```
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include "watchpoint.h"
```

Functions

- void WatchPoint_Insert (uint16_t u16Addr_)

  WatchPoint_Insert.
- void WatchPoint_Delete (uint16_t u16Addr_)

  WatchPoint_Delete.
- bool WatchPoint_EnabledAtAddress (uint16_t u16Addr_)

  WatchPoint_EnabledAtAddress.

4.91.1 Detailed Description

Implements data watchpoints for debugging running programs based on reads/writes to a given memory address.

Definition in file watchpoint.c.

4.91.2 Function Documentation

4.91.2.1 void WatchPoint_Delete (uint16_t *u16Addr_*)

WatchPoint_Delete.

Remove a data watchpoint installed at a specific address. Has no effect if there isn't a watchpoint at the given address.

Parameters

*u16Addr_*	Address to remove data watchpoints from (if any)

Definition at line 57 of file watchpoint.c.

4.91.2.2 bool WatchPoint_EnabledAtAddress (uint16_t *u16Addr_*)

WatchPoint_EnabledAtAddress.

Check to see whether or not a watchpoint is installed at a given address

Parameters

*u16Addr_*	Address to check

Returns

true if watchpoint is installed at the specified adress

Definition at line 97 of file watchpoint.c.

4.91.2.3 void WatchPoint_Insert (uint16_t *u16Addr_*)

WatchPoint_Insert.

Insert a data watchpoint for a given address. Has no effect if a watchpoint already exists at the specified address.

Parameters

*u16Addr_*	Address of the watchpoint.

Definition at line 31 of file watchpoint.c.

4.92 watchpoint.c

```
00001
00002 /*****************************************************************************
00003  *      (      (                       (      |
00004  *    )\ )   )\ )     (              )\ )    |
00005  *   (()/(  (()/(     )\      (    (  (()/(    | -- [ Funkenstein ] -------------
00006  *    /(_)) /(_))((((_)()\   )\  /(_))      | -- [ Litle ] --------------------
00007  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ----------------------
00008  *   | |_  | |     (_)_\(_)\ \ / / | _ \     | -- [ Virtual ] ------------------
00009  *   | __| | |__    / _ \    \ V /  |   /     | -- [ Runtime ] ------------------
00010  *   |_|   |____| /_/ \_\    \_/   |_|_\     |
00011  *                                          | "Yeah, it does Arduino..."
00012  * -----------------------------------------+--------------------------------
00013  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00014  *     See license.txt for details
00015  *****************************************************************************/
00023 #include <stdint.h>
00024 #include <stdbool.h>
00025 #include <stdio.h>
00026 #include <stdlib.h>
00027
00028 #include "watchpoint.h"
00029
00030 //-----------------------------------------------------------------------------
00031 void WatchPoint_Insert( uint16_t u16Addr_ )
00032 {
00033     // Don't add multiple watchpoints at the same address
00034     if (WatchPoint_EnabledAtAddress( u16Addr_ ))
00035     {
00036         return;
00037     }
00038
00039     WatchPoint_t *pstNewWatch = NULL;
00040
00041     pstNewWatch = (WatchPoint_t*)malloc( sizeof(WatchPoint_t) );
00042
00043     pstNewWatch->next = stCPU.pstWatchPoints;
00044     pstNewWatch->prev = NULL;
00045
00046     pstNewWatch->u16Addr = u16Addr_;
00047
00048     if (stCPU.pstWatchPoints)
00049     {
00050         WatchPoint_t *pstTemp = stCPU.pstWatchPoints;
00051         pstTemp->prev = pstNewWatch;
00052     }
00053     stCPU.pstWatchPoints = pstNewWatch;
00054 }
00055
00056 //-----------------------------------------------------------------------------
00057 void WatchPoint_Delete( uint16_t u16Addr_ )
00058 {
```

```
00059     WatchPoint_t *pstTemp = stCPU.pstWatchPoints;
00060
00061     while (pstTemp)
00062     {
00063         if (pstTemp->u16Addr == u16Addr_)
00064         {
00065             // Remove node -- reconnect surrounding elements
00066             WatchPoint_t *pstNext = pstTemp->next;
00067             if (pstNext)
00068             {
00069                 pstNext->prev = pstTemp->prev;
00070             }
00071
00072             WatchPoint_t *pstPrev = pstTemp->prev;
00073             if (pstPrev)
00074             {
00075                 pstPrev->next = pstTemp->next;
00076             }
00077
00078             // Adjust list-head if necessary
00079             if (pstTemp == stCPU.pstWatchPoints)
00080             {
00081                 stCPU.pstWatchPoints = pstNext;
00082             }
00083
00084             // Free the node/iterate to next node.
00085             pstPrev = pstTemp;
00086             pstTemp = pstTemp->next;
00087             free(pstPrev);
00088         }
00089         else
00090         {
00091             pstTemp = pstTemp->next;
00092         }
00093     }
00094 }
00095
00096 //-----------------------------------------------------------------------------
00097 bool WatchPoint_EnabledAtAddress( uint16_t u16Addr_ )
00098 {
00099     WatchPoint_t *pstTemp = stCPU.pstWatchPoints;
00100
00101     while (pstTemp)
00102     {
00103         if (pstTemp->u16Addr == u16Addr_)
00104         {
00105             return true;
00106         }
00107         pstTemp = pstTemp->next;
00108     }
00109     return false;
00110 }
```

4.93 watchpoint.h File Reference

Implements data watchpoints for debugging running programs based on reads/writes to a given memory address.

```
#include <stdint.h>
#include <stdbool.h>
#include "avr_cpu.h"
```

Data Structures

- struct _WatchPoint

Typedefs

- typedef struct _WatchPoint **WatchPoint_t**

Functions

- void WatchPoint_Insert (uint16_t u16Addr_)

  WatchPoint_Insert.
- void WatchPoint_Delete (uint16_t u16Addr_)

  WatchPoint_Delete.
- bool WatchPoint_EnabledAtAddress (uint16_t u16Addr_)

  WatchPoint_EnabledAtAddress.

4.93.1 Detailed Description

Implements data watchpoints for debugging running programs based on reads/writes to a given memory address.

Definition in file watchpoint.h.

4.93.2 Function Documentation

4.93.2.1 void WatchPoint_Delete (uint16_t *u16Addr_*)

WatchPoint_Delete.

Remove a data watchpoint installed at a specific address. Has no effect if there isn't a watchpoint at the given address.

Parameters

*u16Addr_*	Address to remove data watchpoints from (if any)

Definition at line 57 of file watchpoint.c.

4.93.2.2 bool WatchPoint_EnabledAtAddress (uint16_t *u16Addr_*)

WatchPoint_EnabledAtAddress.

Check to see whether or not a watchpoint is installed at a given address

Parameters

*u16Addr_*	Address to check

Returns

true if watchpoint is installed at the specified adress

Definition at line 97 of file watchpoint.c.

4.93.2.3 void WatchPoint_Insert (uint16_t *u16Addr_*)

WatchPoint_Insert.

Insert a data watchpoint for a given address. Has no effect if a watchpoint already exists at the specified address.

Parameters

*u16Addr_*	Address of the watchpoint.

Definition at line 31 of file watchpoint.c.

4.94 watchpoint.h

```
00001 /*************************************************************************
00002  *      (      (                       (      |
00003  *    )\ )  )\ )     (               )\ )    |
00004  *   (()/( (()/(     )\      (   (  (()/(    | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))((((_)()\  )\  /(_))       | -- [ Litle ] --------------------
00006  *   (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ---------------------
00007  *   | |_  | |     (_)_\(_)\ \ / / | _ \     | -- [ Virtual ] -----------------
00008  *   | __| | |__    / _ \   \ V /  |   /     | -- [ Runtime ] -----------------
00009  *   |_|   |____| /_/ \_\   \_/   |_|_\     |
00010  *                                         | "Yeah, it does Arduino..."
00011  * ----------------------------------------+---------------------------------
00012  * (c) Copyright 2014, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  *************************************************************************/
00022 #ifndef __WATCHPOINT_H__
00023 #define __WATCHPOINT_H__
00024
00025 #include <stdint.h>
00026 #include <stdbool.h>
00027
00028 #include "avr_cpu.h"
00029
00030 //-------------------------------------------------------------------------
00031 typedef struct _WatchPoint
00032 {
00033     struct _WatchPoint *next;
00034     struct _WatchPoint *prev;
00035
00036     uint16_t    u16Addr;
00037 } WatchPoint_t;
00038
00039 //-------------------------------------------------------------------------
00048 void WatchPoint_Insert( uint16_t u16Addr_ );
00049
00050 //-------------------------------------------------------------------------
00059 void WatchPoint_Delete( uint16_t u16Addr_ );
00060
00061 //-------------------------------------------------------------------------
00070 bool WatchPoint_EnabledAtAddress( uint16_t u16Addr_ );
00071
00072 #endif
00073
```