flAVR - Funkenstein Little AVR Virtual Runtime

Generated by Doxygen 1.8.7

Mon May 2 2016 22:05:29

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 AddressCoverageTLV_t Struct Reference

Data Fields

- uint32_t **u32CodeAddress**
- uint64_t **u64Hits**
- char szDisasmLine [256]

  Disassembly for the address in question.

3.6.1 Detailed Description

Definition at line 55 of file code_profile.c.

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

- code_profile.c

3.7 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

3.7.1 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

3.8 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]

  bool **bExitOnReset**

  bool **bProfile**

3.8.1 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

3.9 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**
- bool **bExitOnReset**

3.9.1 Detailed Description

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

Definition at line 146 of file avr_cpu.h.

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

- avr_cpu.h

3.10 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

3.10.1 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

3.11 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.

3.11.1 Detailed Description

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

Definition at line 29 of file variant.h.

3.11.2 Field Documentation

3.11.2.1 const char∗ AVR_Variant_t::szName

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

"atmega328p")

Definition at line 31 of file variant.h.

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

- variant.h

3.12 AVRPeripheral Struct Reference

Data Fields

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

3.12.1 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

3.13 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**

3.13.1 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

3.14 Debug_Symbol_t Struct Reference

Data Fields

- Debug_t eType

  Debug symbol type.
- uint32_t u32StartAddr

  Start of the address range held by the symbol.
- uint32_t u32EndAddr

  Last address held by the symbol.
- const char ∗ szName

  Name of the debug symbol.
- uint64_t u64TotalRefs

  Total reference count, used in code profiling.
- uint64_t u64EpochRefs

  Current reference count, used in code profiling.

3.14.1 Detailed Description

Definition at line 36 of file debug_sym.h.

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

- debug_sym.h

3.15 DrawPoint_t Struct Reference

Data Fields

- uint16_t usX

    X coordinate of the pixel.
- uint16_t usY

    Y coordinate of the pixel.
- uint32_t uColor

    Color of the pixel in 5:6:5 format.

3.15.1 Detailed Description

Definition at line 39 of file ka_graphics.c.

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

- ka_graphics.c

3.16 ElfHeader_t Struct Reference

Data Fields

- uint32_t **u32IdentMagicNumber**
- uint8_t **u8IdentFormat**
- uint8_t **u8IdentEndianness**
- uint8_t **u8IdentVersion**
- uint8_t **u8IdentABI**
- uint8_t **u8IdentABIVersion**
- uint8_t **u8Pad1** [7]
- uint16_t **u16Type**
- uint16_t **u16Machine**
- uint32_t **u32Version**
- uint32_t **u32EntryPoint**
- uint32_t **u32PHOffset**
- uint32_t **u32SHOffset**
- uint32_t **u32Flags**
- uint16_t **u16EHSize**
- uint16_t **u16PHSize**
- uint16_t **u16PHNum**
- uint16_t **u16SHSize**
- uint16_t **u16SHNum**
- uint16_t **u16SHIndex**

3.16.1 Detailed Description

Definition at line 72 of file elf_types.h.

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

- elf_types.h

3.17 ElfProgramHeader_t Struct Reference

Data Fields

- uint32_t **u32Type**
- uint32_t **u32Offset**
- uint32_t **u32VirtualAddress**
- uint32_t **u32PhysicalAddress**
- uint32_t **u32FileSize**
- uint32_t **u32MemSize**
- uint32_t **u32Flags**
- uint32_t **u32Alignment**

3.17.1 Detailed Description

Definition at line 122 of file elf_types.h.

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

- elf_types.h

3.18 ElfSectionHeader_t Struct Reference

Data Fields

- uint32_t **u32Name**
- uint32_t **u32Type**
- uint32_t **u32Flags**
- uint32_t **u32Address**
- uint32_t **u32Offset**
- uint32_t **u32Size**
- uint32_t **u32Link**
- uint32_t **u32Info**
- uint32_t **u32Alignment**
- uint32_t **u32EntrySize**

3.18.1 Detailed Description

Definition at line 135 of file elf_types.h.

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

- elf_types.h

3.19 ElfSymbol_t Struct Reference

Data Fields

- uint32_t **u32Name**
- uint32_t **u32Value**
- uint32_t **u32Size**
- uint8_t **u8Info**
- uint8_t **u8Other**
- uint16_t **u16SHIndex**

3.19.1 Detailed Description

Definition at line 150 of file elf_types.h.

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

- elf_types.h

3.20 FunctionCoverageTLV_t Struct Reference

Data Fields

- uint32_t **u32FunctionSize**
- uint32_t **u32AddressesHit**
- char **szSymName** [256]

3.20.1 Detailed Description

Definition at line 47 of file code_profile.c.

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

- code_profile.c

3.21 FunctionProfileTLV_t Struct Reference

Data Fields

- uint64_t **u64CyclesTotal**
- uint64_t **u64CPUCycles**
- char **szSymName** [256]

3.21.1 Detailed Description

Definition at line 39 of file code_profile.c.

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

- code_profile.c

3.22 GDBCommandMap_t Struct Reference

Data Fields

- GDBCommandType_t **eCmd**
- const char * **szToken**
- GDBCommandHandler_t **pfHandler**

3.22.1 Detailed Description

Definition at line 63 of file gdb_rsp.c.

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

- gdb_rsp.c

3.23 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.

3.23.1 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

3.24 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.

3.24.1 Detailed Description

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

Definition at line 52 of file interactive.c.

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

- interactive.c

3.25 Interrupt_Callout_ Struct Reference

Data Fields

- struct Interrupt_Callout_ ∗ pstNext

  Next interrupt callout.
- InterruptCalloutFunc pfCallout

  Callout function.

3.25.1 Detailed Description

Definition at line 29 of file interrupt_callout.c.

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

- interrupt_callout.c

3.26 KernelAwareTrace_t Struct Reference

Data Fields

- uint16_t **u16File**
- uint16_t **u16Line**
- uint16_t **u16Code**
- uint16_t **u16Arg1**
- uint16_t **u16Arg2**

3.26.1 Detailed Description

Definition at line 34 of file ka_trace.c.

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

- ka_trace.c

3.27 Mark3_Context_t Struct Reference

Data Fields

- uint8_t **SPH**
- uint8_t **SPL**
- uint8_t **r** [32]
- uint8_t **SREG**
- uint16_t **PC**

3.27.1 Detailed Description

Definition at line 26 of file ka_thread.h.

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

- ka_thread.h

3.28 Mark3_Thread_Info_t Struct Reference

Data Fields

- Mark3_Thread_t ∗ **pstThread**
- uint8_t **u8ThreadID**
- uint64_t **u64TotalCycles**
- uint64_t **u64EpockCycles**
- bool **bActive**

3.28.1 Detailed Description

Definition at line 84 of file ka_thread.c.

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

- ka_thread.c

3.29 Mark3_Thread_t Struct Reference

Data Fields

- uint16_t u16NextPtr

  Link list pointers.
- uint16_t **u16PrevPtr**
- uint16_t u16StackTopPtr

  Pointer to the top of the thread's stack.
- uint16_t u16StackPtr

  Pointer to the thread's stack.
- uint8_t u8ThreadID

  Thread ID.
- uint8_t u8Priority

  Default priority of the thread.

- uint8_t u8CurPriority

    Current priority of the thread (priority inheritence)
- uint8_t u8ThreadState

    Thread's current state (ready. blocking, etc)
- uint16_t u16StackSize

    Size of the stack (in bytes)
- uint16_t u16CurrentThreadList

    Threadlists.
- uint16_t **u16OwnerThreadList**
- uint16_t u16EntryPoint

    The entry-point function called when the thread starts.
- void ∗ m_pvArg

    Pointer to the argument passed into the thread's entrypoint.
- uint16_t u16Quantum

    Thread quantum (in milliseconds)

3.29.1 Detailed Description

Definition at line 41 of file ka_thread.c.

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

- ka_thread.c

3.30 Mark3ContextSwitch_TLV_t Struct Reference

Data Fields

- uint64_t **u64Timestamp**
- uint16_t **u16StackMargin**
- uint8_t **u8ThreadID**
- uint8_t **u8ThreadPri**

3.30.1 Detailed Description

Definition at line 94 of file ka_thread.c.

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

- ka_thread.c

3.31 Mark3Interrupt_TLV_t Struct Reference

Data Fields

- uint64_t **u64TimeStamp**
- uint8_t **u8Vector**
- bool **bEntry**

3.31.1 Detailed Description

Definition at line 38 of file ka_interrupt.c.

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

- ka_interrupt.c

3.32 Mark3Profile_TLV_t Struct Reference

Data Fields

- uint64_t u64Timestamp

    Timestamp when the profiling print was made.
- uint64_t u64ProfileCount

    Count of profiling events.
- uint64_t u64ProfileTotalCycles

    Total cycles (sum from all profiling events.
- char szName [32]

    Profiling name.

3.32.1 Detailed Description

Definition at line 44 of file ka_profile.c.

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

- ka_profile.c

3.33 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.
- const char * szDescription

    Description string, used for printing valid options.
- char * szParameter

    Parameter string associated with the option.
- bool bStandalone

    Attribute is standalone (no parameter value expected)

3.33.1 Detailed Description

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

Definition at line 31 of file options.c.

3.33.2 Field Documentation

3.33.2.1 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

3.34 Profile_t Struct Reference

Data Fields

- Debug_Symbol_t ∗ pstSym

  Pointer to the debug symbol being profiled at this address.
- uint64_t u64TotalHit

  Total count of hits at this address.
- uint64_t u64EpochHit

  Count of hits at this address in the current epoch.

3.34.1 Detailed Description

Definition at line 31 of file code_profile.c.

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

- code_profile.c

3.35 TLV_t Struct Reference

Data Fields

- FlavrTag_t eTag

  Tag for the object.
- uint16_t u16Len

  Number of bytes that follow in this entry.
- uint8_t au8Data [1]

  Data array (1 or more bytes)

3.35.1 Detailed Description

Definition at line 53 of file tlv_file.h.

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

- tlv_file.h

3.36 TraceBuffer_t Struct Reference

Implements a circular buffer of trace elements, sized according to the compile-time configuration.

`#include <trace_buffer.h>`

Data Fields

- TraceElement_t astTraceStep [CONFIG_TRACEBUFFER_SIZE]

    Array of trace samples.
- uint32_t u32Index

    Current sample index.

3.36.1 Detailed Description

Implements a circular buffer of trace elements, sized according to the compile-time configuration.

Definition at line 53 of file trace_buffer.h.

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

- trace_buffer.h

3.37 TraceElement_t Struct Reference

Struct defining the CPU's running state at each tracebuffer sample point.

`#include <trace_buffer.h>`

Data Fields

- uint64_t u64Counter

    Instruction counter.
- uint64_t u64CycleCount

    CPU Cycle counter.
- uint16_t u16OpCode

    opcode @ trace sample
- uint16_t u16PC

    program counter @ trace sample
- uint16_t u16SP

    stack pointer @ trace sample
- uint8_t u8SR

    status register @ trace sample
- AVR_CoreRegisters stCoreRegs

    core CPU registers @ trace sample

3.37.1 Detailed Description

Struct defining the CPU's running state at each tracebuffer sample point.

Definition at line 35 of file trace_buffer.h.

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

- trace_buffer.h

3.38 Write_Callout_ Struct Reference

Data Fields

- struct Write_Callout_ ∗ pstNext

    Pointer to the next callout.
- uint16_t u16Addr

    Address in RAM to monitor.
- WriteCalloutFunc pfCallout

    Function to call on write.

3.38.1 Detailed Description

Definition at line 31 of file write_callout.c.

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

- write_callout.c

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

```
/*****************************************************************************
 *     (     (                      (     |
 *    )\ )  )\ )    (              )\ )   |
 *   (()/( (()/(    )\     (   (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / /| _ \    | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V / |   /    | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/  |_|_\    |
 *                                        | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __AVR_COREREG_H__
#define __AVR_COREREG_H__

#include <stdint.h>

typedef struct
{
    union
    {
        uint8_t r[32];
        uint16_t r_word[16];
        struct
        {
```

```
            uint16_t r1_0;
            uint16_t r3_2;
            uint16_t r5_4;
            uint16_t r7_6;
            uint16_t r9_8;
            uint16_t r11_10;
            uint16_t r13_12;
            uint16_t r15_14;
            uint16_t r17_16;
            uint16_t r19_18;
            uint16_t r21_20;
            uint16_t r23_22;
            uint16_t r25_24;
            uint16_t r27_26;
            uint16_t r29_28;
            uint16_t r31_30;
        };
        struct
        {
            uint8_t r0;
            uint8_t r1;
            uint8_t r2;
            uint8_t r3;
            uint8_t r4;
            uint8_t r5;
            uint8_t r6;
            uint8_t r7;
            uint8_t r8;
            uint8_t r9;
            uint8_t r10;
            uint8_t r11;
            uint8_t r12;
            uint8_t r13;
            uint8_t r14;
            uint8_t r15;
            uint8_t r16;
            uint8_t r17;
            uint8_t r18;
            uint8_t r19;
            uint8_t r20;
            uint8_t r21;
            uint8_t r22;
            uint8_t r23;
            uint8_t r24;
            uint8_t r25;
            union
            {
                uint16_t X;
                struct
                {
                    uint8_t r26;
                    uint8_t r27;
                };
            };
            union
            {
                uint16_t Y;
                struct
                {
                    uint8_t r28;
                    uint8_t r29;
                };
            };
            union
            {
                uint16_t Z;
                struct
                {
                    uint8_t r30;
                    uint8_t r31;
                };
            };
        };
    };
} AVR_CoreRegisters;

#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 264 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 282 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 2014-15, 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)
```

```
    {
        return 0xFFFF;
    }
    return stCPU.pu16ROM[ stCPU.u16PC ];
}

//---------------------------------------------------------------------------
static void CPU_GetOpCycles( uint16_t OP_ )
{
#if FEATURE_USE_JUMPTABLES
    stCPU.u16ExtraCycles = au8OpCycles[ OP_ ];
#else
    stCPU.u16ExtraCycles = AVR_Opcode_Cycles( OP_ );
#endif
}

//---------------------------------------------------------------------------
static void CPU_GetOpSize( uint16_t OP_ )
{
#if FEATURE_USE_JUMPTABLES
    stCPU.u16ExtraPC = au8OpSizes[ OP_ ];
#else
    stCPU.u16ExtraPC = AVR_Opcode_Size( OP_ );
#endif
}

//---------------------------------------------------------------------------
static void CPU_PeripheralCycle( void )
{
    IO_Clock();
}

//---------------------------------------------------------------------------
void CPU_RunCycle( void )
{
    uint16_t OP;

    if (!stCPU.bAsleep)
    {

        OP = CPU_Fetch();

        // From the first word fetched, figure out how big this opcode is
        // (either 16 or 32-bit)
        CPU_GetOpSize(  OP );

        // Based on the first word fetched, figure out the minimum number of
        // CPU cycles required to execute the instruction fetched.
        CPU_GetOpCycles(  OP );

        // Decode the instruction, load internal registers with appropriate
        // values.
        CPU_Decode(  OP );

        // Execute the instruction that was just decoded
        CPU_Execute(  OP );

        // Update the PC based on the size of the instruction + whatever
        // modifications occurred during the execution cycle.
        stCPU.u16PC += stCPU.u16ExtraPC;

        // Add CPU clock cycles to the global cycle counter based on
        // the minimum instruction time, plus whatever modifiers are applied
        // during execution of the instruction.
        stCPU.u64CycleCount += stCPU.u16ExtraCycles;

        // Cycle-accurate peripheral clocking -- one iteration for each
        // peripheral for each CPU cycle of the instruction.
        // Note that CPU Interrupts are generated in the peripheral
        // phase of the instruction cycle.
        while (stCPU.u16ExtraCycles--)
        {
            CPU_PeripheralCycle();
        }

        // Increment the "total executed instruction counter"
        stCPU.u64InstructionCount++;

    }
    else
    {
        // CPU is asleep, just NOP and wait until we hit an interrupt.
        stCPU.u64CycleCount++;
        CPU_PeripheralCycle();
    }

    // Check to see if there are any pending interrupts - if so, vector
```

```
    // to the appropriate location.  This has no effect if no interrupts
    // are pending
    AVR_Interrupt();
}


#if FEATURE_USE_JUMPTABLES
//---------------------------------------------------------------------------
static void CPU_BuildDecodeTable(void)
{
    uint32_t i;
    for (i = 0; i < 65536; i++)
    {
        astDecoders[i] = AVR_Decoder_Function(i);
    }
}

//---------------------------------------------------------------------------
static void CPU_BuildOpcodeTable(void)
{
    uint32_t i;
    for (i = 0; i < 65536; i++)
    {
        astOpcodes[i] = AVR_Opcode_Function(i);
    }
}

//---------------------------------------------------------------------------
static void CPU_BuildSizeTable(void)
{
    uint32_t i;
    for (i = 0; i < 65536; i++)
    {
        au8OpSizes[i] = AVR_Opcode_Size(i);
    }
}

//---------------------------------------------------------------------------
static void CPU_BuildCycleTable(void)
{
    uint32_t i;
    for (i = 0; i < 65536; i++)
    {
        au8OpCycles[i] = AVR_Opcode_Cycles(i);
    }
}
#endif

//---------------------------------------------------------------------------
void CPU_Init( AVR_CPU_Config_t *pstConfig_ )
{
    memset( &stCPU, 0, sizeof(stCPU));
    pstConfig_->u32RAMSize += 256;

    stCPU.bExitOnReset = pstConfig_->bExitOnReset;

    // Dynamically allocate memory for RAM, ROM, and EEPROM buffers
    stCPU.pu8EEPROM = (uint8_t*)malloc( pstConfig_->u32EESize );
    stCPU.pu16ROM    = (uint16_t*)malloc( pstConfig_->u32ROMSize );
    stCPU.pstRAM     = (AVR_RAM_t*)malloc( pstConfig_->u32RAMSize );

    stCPU.u32ROMSize = pstConfig_->u32ROMSize;
    stCPU.u32RAMSize = pstConfig_->u32RAMSize;
    stCPU.u32EEPROMSize = pstConfig_->u32EESize;

    memset( stCPU.pu8EEPROM, 0, pstConfig_->u32EESize );
    memset( stCPU.pu16ROM, 0, pstConfig_->u32ROMSize );
    memset( stCPU.pstRAM, 0, pstConfig_->u32RAMSize );

    // Set the base stack pointer to top-of-ram.
    uint16_t u16InitialStack = 256 + pstConfig_->u32RAMSize - 1;
    stCPU.pstRAM->stRegisters.SPH.r = (uint8_t)(u16InitialStack >> 8);
    stCPU.pstRAM->stRegisters.SPL.r = (uint8_t)(u16InitialStack & 0xFF);

    // Reset the interrupt priority register
    stCPU.u8IntPriority = 255;

#if FEATURE_USE_JUMPTABLES
    CPU_BuildCycleTable();
    CPU_BuildSizeTable();
    CPU_BuildOpcodeTable();
    CPU_BuildDecodeTable();
#endif
}

//---------------------------------------------------------------------------
void CPU_AddPeriph( AVRPeripheral *pstPeriph_ )
```

```
{
    IO_AddClocker(  pstPeriph_ );

    uint8_t i;
    for (i = pstPeriph_->u8AddrStart; i <= pstPeriph_->u8AddrEnd; i++)
    {
        IO_AddReader(  pstPeriph_, i );
        IO_AddWriter(  pstPeriph_, i );
    }

    if (pstPeriph_->pfInit)
    {
        pstPeriph_->pfInit( pstPeriph_->pvContext );
    }
}

//---------------------------------------------------------------------------
void CPU_RegisterInterruptCallback( InterruptAck pfIntAck_, uint8_t ucVector_
)
{
    if (ucVector_ >= 32)
    {
        return;
    }

    stCPU.apfInterruptCallbacks[ ucVector_ ] = pfIntAck_;
}
```

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 264 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 282 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

```
/***************************************************************************
 *      (      (                       (      |
 *     )\ )   )\ )     (              )\ )    |
 *    (()/(  (()/(     )\     (    (  (()/(    | -- [ Funkenstein ] -------------
 *     /(_))  /(_))((((_)()\  )\  /(_))      | -- [ Litle ] -------------------
 *    (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
 *    | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *    | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
 *    |_|   |____| /_/ \_\   \_/   |_|_\    |
 *                                           | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __AVR_CPU_H__
#define __AVR_CPU_H__

#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"

//---------------------------------------------------------------------------
typedef struct
{
    union
    {
        AVRRegisterFile stRegisters;
        uint8_t au8RAM[ sizeof(AVRRegisterFile) ];
    };
} AVR_RAM_t;

//---------------------------------------------------------------------------
typedef struct
{
    //---------------------------------------------------------------------------
    // Jump tables for peripheral read/write functions.  This implementaton uses
    // a table with function pointer arrays, enabling multiple peripherals to
    // monitor reads/writes at particular addresses efficiently.
    //---------------------------------------------------------------------------
    IOReaderList *apstPeriphReadTable[CONFIG_IO_ADDRESS_BYTES];
    IOWriterList *apstPeriphWriteTable[CONFIG_IO_ADDRESS_BYTES];
    IOClockList  *pstClockList;

    //---------------------------------------------------------------------------
    // List of data watchpoints
    struct _WatchPoint *pstWatchPoints;

    //---------------------------------------------------------------------------
    // List of instruction breakpoints
    struct _BreakPoint *pstBreakPoints;

```

```
    //---------------------------------------------------------------------------
    // Internal CPU Registers (not exposed via IO space)
    uint16_t     u16PC;          // Program counter is not memory mapped, unlike all others

    //---------------------------------------------------------------------------
    // Emulator variables
    uint64_t     u64InstructionCount; // Total Executed instructions
    uint64_t     u64CycleCount; // Cycle Counter
    uint32_t     u32CoreFreq;   // CPU Frequency (Hz)
    uint32_t     u32WDTCount;   // Current watchdog timer count
    uint16_t     u16ExtraPC;    // Offset to add to the PC after executing an instruction
    uint16_t     u16ExtraCycles;// CPU Cycles to add for the current instruction

    bool         bAsleep;       // Whether or not the CPU is sleeping (wake by interrupt)
    //---------------------------------------------------------------------------
    // Temporary registers used for optimizing opcodes - for various addressing modes
    uint16_t    *Rd16;
    uint8_t     *Rd; // Destination register (in some cases, also source)

    uint16_t    *Rr16;
    uint8_t     *Rr; // Source register

    uint16_t    K;   // Constant data
    union
    {
        uint32_t    k;   // Constant address
        int32_t     k_s; // Signed, constant address
    };

    uint8_t     A; // IO location address
    uint8_t     b; // Bit in a register file (3-bits wide)
    uint8_t     s; // BIt in the status register (3-bits wide)
    uint8_t     q; // Displacement for direct addressing (6-bits)

    //---------------------------------------------------------------------------
    // Setting up regions of memory for general-purpose RAM (shared with the
    // IO space from 0-0xFF), ROM/FLASH, and EEPROM.
    //---------------------------------------------------------------------------
    uint16_t     *pu16ROM;
    uint8_t      *pu8EEPROM;
    AVR_RAM_t    *pstRAM;

    uint32_t    u32ROMSize;
    uint32_t    u32EEPROMSize;
    uint32_t    u32RAMSize;

    //---------------------------------------------------------------------------
    uint8_t     u8IntPriority;  // Priority of pending interrupts this cycle
    uint32_t    u32IntFlags;    // Bitmask for the 32 interrupts

    //---------------------------------------------------------------------------
    InterruptAck apfInterruptCallbacks[32]; // Interrupt callbacks

    //---------------------------------------------------------------------------
    bool        bExitOnReset;   // Flag indicating behavior when we jump to 0.  true == exit emulator
    bool        bProfile;       // Flag indicating that CPU is running with active code profiling
} AVR_CPU;


//---------------------------------------------------------------------------
typedef struct
{
    uint32_t u32ROMSize;
    uint32_t u32RAMSize;
    uint32_t u32EESize;
    bool     bExitOnReset;
} AVR_CPU_Config_t;

//---------------------------------------------------------------------------
void CPU_Init( AVR_CPU_Config_t *pstConfig_ );

//---------------------------------------------------------------------------
uint16_t CPU_Fetch( void );

//---------------------------------------------------------------------------
void CPU_RunCycle( void );

//---------------------------------------------------------------------------
void CPU_AddPeriph( AVRPeripheral *pstPeriph_ );

//---------------------------------------------------------------------------
void CPU_RegisterInterruptCallback( InterruptAck pfIntAck_, uint8_t ucVector_
);


extern AVR_CPU stCPU;

```

```
00203 #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 2014-15, 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>
```

```
#include <stdint.h>

//---------------------------------------------------------------------------
#define PRINT_FUNC      printf

#define RAM_DISPLAY_SPAN        (16)
#define ROM_DISPLAY_SPAN        (8)

//---------------------------------------------------------------------------
void print_core_regs( void )
{
    uint8_t i;
    for (i = 0; i < 32; i++)
    {
        PRINT_FUNC( "[R%02d] = 0x%02X\n", i, stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[i] );
    }
    PRINT_FUNC("[SP]   = 0x%02X%02X\n", (uint8_t)stCPU.pstRAM->stRegisters.SPH.r, (uint8_t)stCPU.pstRAM->stRegisters.SPL.r );
    PRINT_FUNC("[PC]   = 0x%04X\n", (uint16_t)stCPU.u16PC );
    PRINT_FUNC("[SREG]= 0x%02X   [", stCPU.pstRAM->stRegisters.SREG.r );

    if (1 == stCPU.pstRAM->stRegisters.SREG.I)
    {
        PRINT_FUNC("I");
    }
    else
    {
        PRINT_FUNC("-");
    }
    if (1 == stCPU.pstRAM->stRegisters.SREG.T)
    {
        PRINT_FUNC("T");
    }
    else
    {
        PRINT_FUNC("-");
    }
    if (1 == stCPU.pstRAM->stRegisters.SREG.H)
    {
        PRINT_FUNC("H");
    }
    else
    {
        PRINT_FUNC("-");
    }
    if (1 == stCPU.pstRAM->stRegisters.SREG.S)
    {
        PRINT_FUNC("S");
    }
    else
    {
        PRINT_FUNC("-");
    }
    if (1 == stCPU.pstRAM->stRegisters.SREG.V)
    {
        PRINT_FUNC("V");
    }
    else
    {
        PRINT_FUNC("-");
    }
    if (1 == stCPU.pstRAM->stRegisters.SREG.N)
    {
        PRINT_FUNC("N");
    }
    else
    {
        PRINT_FUNC("-");
    }
    if (1 == stCPU.pstRAM->stRegisters.SREG.Z)
    {
        PRINT_FUNC("Z");
    }
    else
    {
        PRINT_FUNC("-");
    }
    if (1 == stCPU.pstRAM->stRegisters.SREG.C)
    {
        PRINT_FUNC("C");
    }
    else
    {
        PRINT_FUNC("-");
    }
    PRINT_FUNC("]\n");
}
```

```
//---------------------------------------------------------------------------
void print_io_reg( uint8_t u8Addr_ )
{
    PRINT_FUNC( "[IO%02X]= 0x%02X\n", u8Addr_, stCPU.pstRAM->au8RAM[u8Addr_] );
}

//---------------------------------------------------------------------------
void print_io_reg_with_name( uint8_t u8Addr_, const char *szName_ )
{
    PRINT_FUNC( "[%s]= 0x%02X\n", szName_, stCPU.pstRAM->au8RAM[u8Addr_] );
}

//---------------------------------------------------------------------------
void print_ram( uint16_t u16Start_, uint16_t u16Span_ )
{
    uint16_t i, j;

    while (u16Span_)
    {
        // Print the current memory address
        PRINT_FUNC( "[0x%04X]", u16Start_ );
        if (u16Span_ < RAM_DISPLAY_SPAN)
        {
            j = u16Span_;
        }
        else
        {
            j = RAM_DISPLAY_SPAN;
        }

        // Print a divider, followed by the ASCII codes for each char
        PRINT_FUNC( "|" );
        for (i = 0; i < j; i++)
        {
            uint8_t u8Char = stCPU.pstRAM->au8RAM[u16Start_ + i];
            if (u8Char < 32)
            {
                u8Char = '.';
            }

            PRINT_FUNC( " %c", u8Char );
        }
        i = j;
        while (i < RAM_DISPLAY_SPAN)
        {
            PRINT_FUNC("   ");
            i++;
        }

        // Print a divider, followed by the HEX code for each char
        PRINT_FUNC( "|" );
        for (i = 0; i < j; i++)
        {
            PRINT_FUNC( " %02X", stCPU.pstRAM->au8RAM[u16Start_ + i]);
        }

        if (u16Span_ < RAM_DISPLAY_SPAN)
        {
            u16Span_ = 0;
        }
        else
        {
            u16Span_ -= RAM_DISPLAY_SPAN;
        }
        u16Start_ += RAM_DISPLAY_SPAN;
        PRINT_FUNC( "\n" );
    }
}

//---------------------------------------------------------------------------
void print_rom( uint16_t u16Start_, uint16_t u16Span_ )
{
    uint16_t i, j;

    while (u16Span_)
    {
        // Print the current memory address
        PRINT_FUNC( "[0x%04X]", u16Start_ );
        if (u16Span_ < ROM_DISPLAY_SPAN)
        {
            j = u16Span_;
        }
        else
        {
            j = ROM_DISPLAY_SPAN;
        }
```

```

        // Print a divider, followed by the ASCII codes for each char
        PRINT_FUNC( "|" );
        for (i = 0; i < j; i++)
        {
            uint16_t u16Val = stCPU.pu16ROM[u16Start_ + i];
            uint8_t u8High = u16Val >> 8;
            uint8_t u8Low = u16Val & 0x00FF;

            if (u8High < 32)
            {
                u8High = '.';
            }
            if (u8Low < 32)
            {
                u8Low = '.';
            }

            PRINT_FUNC( " %c%c", u8High, u8Low );
        }
        i = j;
        while (i < ROM_DISPLAY_SPAN)
        {
            PRINT_FUNC("  ");
            i++;
        }

        // Print a divider, followed by the HEX code for each char
        PRINT_FUNC( "|" );
        for (i = 0; i < j; i++)
        {
            PRINT_FUNC( " %04X", stCPU.pu16ROM[u16Start_ + i]);
        }

        if (u16Span_ < ROM_DISPLAY_SPAN)
        {
            u16Span_ = 0;
        }
        else
        {
            u16Span_ -= ROM_DISPLAY_SPAN;
        }
        u16Start_ += ROM_DISPLAY_SPAN;
        PRINT_FUNC( "\n" );
    }
}
```

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

```
/*****************************************************************************
 *     (     (                      (     |
 *    )\ )  )\ )    (              )\ )   |
 *   (()/( (()/(    )\     (   (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                        | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __AVR_CPU_PRINT_H__
#define __AVR_CPU_PRINT_H__

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

//---------------------------------------------------------------------------
void print_core_regs( void );

//---------------------------------------------------------------------------
void print_io_reg( uint8_t u8Addr_ );

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

//---------------------------------------------------------------------------
void print_ram( uint16_t u16Start_, uint16_t u16Span_ );

//---------------------------------------------------------------------------
void print_rom( uint16_t u16Start_, uint16_t u16Span_ );

#endif
```

4.11 avr_disasm.c File Reference

AVR Disassembler Implementation.

```
#include <stdint.h>
#include <stdio.h>
#include "emu_config.h"
#include "avr_disasm.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** (char ∗szOutput_)
- static void **AVR_Disasm_ADC** (char ∗szOutput_)
- static void **AVR_Disasm_ADIW** (char ∗szOutput_)
- static void **AVR_Disasm_SUB** (char ∗szOutput_)
- static void **AVR_Disasm_SUBI** (char ∗szOutput_)
- static void **AVR_Disasm_SBC** (char ∗szOutput_)
- static void **AVR_Disasm_SBCI** (char ∗szOutput_)
- static void **AVR_Disasm_SBIW** (char ∗szOutput_)
- static void **AVR_Disasm_AND** (char ∗szOutput_)
- static void **AVR_Disasm_ANDI** (char ∗szOutput_)
- static void **AVR_Disasm_OR** (char ∗szOutput_)
- static void **AVR_Disasm_ORI** (char ∗szOutput_)
- static void **AVR_Disasm_EOR** (char ∗szOutput_)
- static void **AVR_Disasm_COM** (char ∗szOutput_)
- static void **AVR_Disasm_NEG** (char ∗szOutput_)
- static void **AVR_Disasm_SBR** (char ∗szOutput_)
- static void **AVR_Disasm_CBR** (char ∗szOutput_)
- static void **AVR_Disasm_INC** (char ∗szOutput_)
- static void **AVR_Disasm_DEC** (char ∗szOutput_)
- static void **AVR_Disasm_TST** (char ∗szOutput_)
- static void **AVR_Disasm_CLR** (char ∗szOutput_)
- static void **AVR_Disasm_SER** (char ∗szOutput_)
- static void **AVR_Disasm_MUL** (char ∗szOutput_)
- static void **AVR_Disasm_MULS** (char ∗szOutput_)
- static void **AVR_Disasm_MULSU** (char ∗szOutput_)
- static void **AVR_Disasm_FMUL** (char ∗szOutput_)
- static void **AVR_Disasm_FMULS** (char ∗szOutput_)
- static void **AVR_Disasm_FMULSU** (char ∗szOutput_)
- static void **AVR_Disasm_DES** (char ∗szOutput_)
- static void **AVR_Disasm_RJMP** (char ∗szOutput_)
- static void **AVR_Disasm_IJMP** (char ∗szOutput_)
- static void **AVR_Disasm_EIJMP** (char ∗szOutput_)
- static void **AVR_Disasm_JMP** (char ∗szOutput_)
- static void **AVR_Disasm_RCALL** (char ∗szOutput_)
- static void **AVR_Disasm_ICALL** (char ∗szOutput_)
- static void **AVR_Disasm_EICALL** (char ∗szOutput_)
- static void **AVR_Disasm_CALL** (char ∗szOutput_)
- static void **AVR_Disasm_RET** (char ∗szOutput_)
- static void **AVR_Disasm_RETI** (char ∗szOutput_)
- static void **AVR_Disasm_CPSE** (char ∗szOutput_)
- static void **AVR_Disasm_CP** (char ∗szOutput_)
- static void **AVR_Disasm_CPC** (char ∗szOutput_)
- static void **AVR_Disasm_CPI** (char ∗szOutput_)
- static void **AVR_Disasm_SBRC** (char ∗szOutput_)
- static void **AVR_Disasm_SBRS** (char ∗szOutput_)
- static void **AVR_Disasm_SBIC** (char ∗szOutput_)
- static void **AVR_Disasm_SBIS** (char ∗szOutput_)
- static void **AVR_Disasm_BRBS** (char ∗szOutput_)
- static void **AVR_Disasm_BRBC** (char ∗szOutput_)
- static void **AVR_Disasm_BREQ** (char ∗szOutput_)
- static void **AVR_Disasm_BRNE** (char ∗szOutput_)
- static void **AVR_Disasm_BRCS** (char ∗szOutput_)
- static void **AVR_Disasm_BRCC** (char ∗szOutput_)

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

- static void **AVR_Disasm_LSR** (char ∗szOutput_)
- static void **AVR_Disasm_POP** (char ∗szOutput_)
- static void **AVR_Disasm_PUSH** (char ∗szOutput_)
- static void **AVR_Disasm_ROL** (char ∗szOutput_)
- static void **AVR_Disasm_ROR** (char ∗szOutput_)
- static void **AVR_Disasm_ASR** (char ∗szOutput_)
- static void **AVR_Disasm_SWAP** (char ∗szOutput_)
- static void **AVR_Disasm_BSET** (char ∗szOutput_)
- static void **AVR_Disasm_BCLR** (char ∗szOutput_)
- static void **AVR_Disasm_SBI** (char ∗szOutput_)
- static void **AVR_Disasm_CBI** (char ∗szOutput_)
- static void **AVR_Disasm_BST** (char ∗szOutput_)
- static void **AVR_Disasm_BLD** (char ∗szOutput_)
- static void **AVR_Disasm_SEC** (char ∗szOutput_)
- static void **AVR_Disasm_CLC** (char ∗szOutput_)
- static void **AVR_Disasm_SEN** (char ∗szOutput_)
- static void **AVR_Disasm_CLN** (char ∗szOutput_)
- static void **AVR_Disasm_SEZ** (char ∗szOutput_)
- static void **AVR_Disasm_CLZ** (char ∗szOutput_)
- static void **AVR_Disasm_SEI** (char ∗szOutput_)
- static void **AVR_Disasm_CLI** (char ∗szOutput_)
- static void **AVR_Disasm_SES** (char ∗szOutput_)
- static void **AVR_Disasm_CLS** (char ∗szOutput_)
- static void **AVR_Disasm_SEV** (char ∗szOutput_)
- static void **AVR_Disasm_CLV** (char ∗szOutput_)
- static void **AVR_Disasm_SET** (char ∗szOutput_)
- static void **AVR_Disasm_CLT** (char ∗szOutput_)
- static void **AVR_Disasm_SEH** (char ∗szOutput_)
- static void **AVR_Disasm_CLH** (char ∗szOutput_)
- static void **AVR_Disasm_BREAK** (char ∗szOutput_)
- static void **AVR_Disasm_NOP** (char ∗szOutput_)
- static void **AVR_Disasm_SLEEP** (char ∗szOutput_)
- static void **AVR_Disasm_WDR** (char ∗szOutput_)
- static void **AVR_Disasm_XCH** (char ∗szOutput_)
- static void **AVR_Disasm_Unimplemented** (char ∗szOutput_)
- AVR_Disasm 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_Disasm 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 1637 of file avr_disasm.c.

4.12 avr_disasm.c

```
/*****************************************************************************
 *     (     (                      (     |
 *    )\ )  )\ )    (              )\ )   |
 *   (()/( (()/(    )\     (   (   (()/(  | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                        | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#include <stdint.h>
#include <stdio.h>

#include "emu_config.h"

#include "avr_disasm.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"

//---------------------------------------------------------------------------
inline int8_t Signed_From_Unsigned_6( uint8_t u8Signed_ )
{
    int8_t i8Ret = 0;
    if( u8Signed_ & 0x20 )
    {
        //Sign extend...
        i8Ret = (int8_t)(u8Signed_ | 0xC0);
    }
    else
    {
        i8Ret = (int8_t)u8Signed_;
    }
    return i8Ret;
}

//---------------------------------------------------------------------------
inline uint8_t Register_From_Rd( void )
{
    return stCPU.Rd - &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0);
}
 //---------------------------------------------------------------------------
inline uint8_t Register_From_Rr( void )
{
    return stCPU.Rr - &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0);
}

//---------------------------------------------------------------------------
inline uint8_t Register_From_Rd16( void )
{
    return (uint8_t*)(stCPU.Rd16) - &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0);
}

//---------------------------------------------------------------------------
inline uint8_t Register_From_Rr16( void )
{
    return (uint8_t*)(stCPU.Rr16) - &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0);
}

```

```
//------------------------------------------------------------------------
static void AVR_Disasm_ADD( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "add r%d, r%d              \t ; Add: r%d = r%d + r%d\n",
                 u8Rd, u8Rr,
                 u8Rd, u8Rd, u8Rr );
}

//------------------------------------------------------------------------
static void AVR_Disasm_ADC( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "adc r%d, r%d              \t ; Add with carry: r%d = r%d + r%d + C\n",
                 u8Rd, u8Rr,
                 u8Rd, u8Rd, u8Rr );

}

//------------------------------------------------------------------------
static void AVR_Disasm_ADIW( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd16();
    uint8_t u8K = stCPU.K;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "adiw r%d:%d, %d           \t ; Add immediate to word: r%d:%d = r%d:%d + %d \n",
                u8Rd + 1, u8Rd, u8K,
                u8Rd + 1, u8Rd, u8Rd + 1, u8Rd, u8K
                );
}

//------------------------------------------------------------------------
static void AVR_Disasm_SUB( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sub r%d, r%d              \t ; Subtract: r%d = r%d - r%d \n",
                u8Rd, u8Rr,
                u8Rd, u8Rd, u8Rr
                );
}

//------------------------------------------------------------------------
static void AVR_Disasm_SUBI( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8K = stCPU.K;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "subi r%d, %d              \t ; Subtract immediate: r%d = r%d - %d \n",
                u8Rd, u8K,
                u8Rd, u8Rd, u8K
                );
}

//------------------------------------------------------------------------
static void AVR_Disasm_SBC( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sbc r%d, r%d              \t ; Subtract with carry: r%d = r%d - r%d - C \n",
                u8Rd, u8Rr,
                u8Rd, u8Rd, u8Rr
                );
}

//------------------------------------------------------------------------
static void AVR_Disasm_SBCI( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8K = stCPU.K;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sbci r%d, %d              \t ; Subtract immediate with carry: r%d = r%d - %d - C\n
",
                u8Rd, u8K,
```

```
                  u8Rd, u8Rd, u8K
                  );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SBIW( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd16();
    uint8_t u8K = stCPU.K;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sbiw r%d:%d, %d           \t ; Subtract immediate from word: r%d:%d = r%d:%d + %d \n",
                  u8Rd + 1, u8Rd, u8K,
                  u8Rd + 1, u8Rd, u8Rd + 1, u8Rd, u8K
                  );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_AND( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "and r%d, r%d              \t ; Logical AND: r%d = r%d & r%d \n",
                  u8Rd, u8Rr,
                  u8Rd, u8Rd, u8Rr
                  );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ANDI( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8K = stCPU.K;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "andi r%d, %d              \t ; Logical AND with Immediate: r%d = r%d & %d\n",
                  u8Rd, u8K,
                  u8Rd, u8Rd, u8K
                  );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_OR( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "or r%d, r%d               \t ; Logical OR: r%d = r%d | r%d \n",
                  u8Rd, u8Rr,
                  u8Rd, u8Rd, u8Rr
                  );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ORI( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8K = stCPU.K;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ori r%d, %d               \t ; Logical OR with Immediate: r%d = r%d | %d\n",
                  u8Rd, u8K,
                  u8Rd, u8Rd, u8K
                  );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_EOR( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "eor r%d, r%d              \t ; Exclusive OR: r%d = r%d ^ r%d \n",
                  u8Rd, u8Rr,
                  u8Rd, u8Rd, u8Rr
                  );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_COM( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
```

```
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "com r%d                   \t ; One’s complement (bitwise inverse): r%d = 0xFF - r%d\n",
                u8Rd,
                u8Rd, u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_NEG( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "neg r%d                   \t ; Two’s complement (sign swap): r%d = 0x00 - r%d\n",
                u8Rd,
                u8Rd, u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SBR( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8K = stCPU.K;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sbr r%d, %d               \t ; Set Bits in Register: r%d = r%d | %d\n",
                u8Rd, u8K,
                u8Rd, u8Rd, u8K
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_CBR( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8K = stCPU.K;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "cbr r%d, %d               \t ; Clear Bits in Register: r%d = r%d & (0xFF - %d)\n",
                u8Rd, u8K,
                u8Rd, u8Rd, u8K
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_INC( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "inc r%d                   \t ; Increment Register: r%d = r%d + 1\n",
                u8Rd,
                u8Rd, u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_DEC( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "dec r%d                   \t ; Decrement Register: r%d = r%d - 1\n",
                u8Rd,
                u8Rd, u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_TST( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "tst r%d                   \t ; Test Register for Zero or Negative\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_CLR( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

```

```
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "clr r%d                   \t ; Clear Register\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SER( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ser r%d                   \t ; Set All Bits in Register\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_MUL( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "mul r%d, r%d              \t ; Unsigned Multiply: r1:0 = r%d * r%d\n",
                u8Rd, u8Rr,
                u8Rd, u8Rr );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_MULS( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "muls r%d, r%d             \t ; Signed Multiply: r1:0 = r%d * r%d\n",
                u8Rd, u8Rr,
                u8Rd, u8Rr );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_MULSU( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "mulsu r%d, r%d            \t ; Signed * Unsigned Multiply: r1:0 = r%d * r%d\n",
                u8Rd, u8Rr,
                u8Rd, u8Rr );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_FMUL( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "fmul r%d, r%d             \t ; Fractional Multiply: r1:0 = r%d * r%d\n",
                u8Rd, u8Rr,
                u8Rd, u8Rr );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_FMULS( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "fmuls r%d, r%d            \t ; Signed Fractional Multiply: r1:0 = r%d * r%d\n",
                u8Rd, u8Rr,
                u8Rd, u8Rr );

}

//---------------------------------------------------------------------------
static void AVR_Disasm_FMULSU( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "fmulsu r%d, r%d           \t ; Signed * Unsigned Fractional Multiply: r1:0 = r%d * r%d\n",
```

```
                u8Rd, u8Rr,
                u8Rd, u8Rr );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_DES( char *szOutput_ )
{
    uint8_t u8K = stCPU.K;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "des %d                     \t ; DES Encrypt/Decrypt\n",
            u8K );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_RJMP( char *szOutput_ )
{
    int16_t i16k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "rjmp %d                    \t ; Relative Jump: PC = PC + %d + 1\n",
                i16k, i16k );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_IJMP( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ijmp                       \t ; Indirect Jump: PC = Z\n");
}

//---------------------------------------------------------------------------
static void AVR_Disasm_EIJMP( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "eijmp                      \t ; Extended Indirect Jump: PC(15:0) = Z(15:0),
 PC(21:16) = EIND\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_JMP( char *szOutput_ )
{
    uint32_t u32k = stCPU.k;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "jmp 0x%X                      \t ; Jump to 0x%X \n",
                u32k, u32k );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_RCALL( char *szOutput_ )
{
    int16_t i16k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "rcall %d                   \t ; Relative call to Subroutine: PC = PC +%d + 1\n",
                i16k, i16k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ICALL( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "icall                      \t ; Indirect Jump: PC = Z\n");
}

//---------------------------------------------------------------------------
static void AVR_Disasm_EICALL( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "eicall                     \t ; Extended Indirect Jump: PC(15:0) = Z(15:0),
 PC(21:16) = EIND\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_CALL( char *szOutput_ )
{
    uint32_t u32k = stCPU.k;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "call 0x%X                  \t ; Long Call to Subroutine: PC = 0x%X \n",
                u32k, u32k
                );
}
//---------------------------------------------------------------------------
static void AVR_Disasm_RET( char *szOutput_ )
```

```
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ret                         \t ; Return from subroutine\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_RETI( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "reti                        \t ; Return from interrupt\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_CPSE( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "cpse r%d, r%d               \t ; Compare, Skip Next If r%d = r%d\n",
                u8Rd, u8Rr,
                u8Rd, u8Rr
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_CP( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "cp r%d, r%d                 \t ; Compare: r%d == r%d\n",
                u8Rd, u8Rr,
                u8Rd, u8Rr
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_CPC( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "cpc r%d, r%d                \t ; Compare with carry: r%d == r%d + C\n",
                u8Rd, u8Rr,
                u8Rd, u8Rr
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_CPI( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8K = stCPU.K;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "cpi r%d, %d                 \t ; Compare with Immediate: r%d == %d\n",
                u8Rd, u8K,
                u8Rd, u8K
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SBRC( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8b = stCPU.b;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sbrc r%d, %d                \t ; Skip if Bit (%d) in Register (r%d) Cleared \n",
                u8Rd, u8b,
                u8Rd, u8b
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SBRS( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8b = stCPU.b;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sbrs r%d, %d                \t ; Skip if Bit (%d) in Register (r%d) Set \n",
                u8Rd, u8b,
                u8Rd, u8b
```

```
                 );

}

//---------------------------------------------------------------------------
static void AVR_Disasm_SBIC( char *szOutput_ )
{
    uint8_t u8A = stCPU.A;
    uint8_t u8b = stCPU.b;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sbic %d, %d               \t ; Skip if Bit (%d) in IO Register (r%d) Cleared \n",
                u8A, u8b,
                u8A, u8b
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SBIS( char *szOutput_ )
{
    uint8_t u8A = stCPU.A;
    uint8_t u8b = stCPU.b;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sbis %d, %d               \t ; Skip if Bit (%d) in IO Register (r%d) Set \n",
                u8A, u8b,
                u8A, u8b
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRBS( char *szOutput_ )
{
    uint8_t u8s = stCPU.s;
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brbs %d, %d               \t ; Branch if Bit (%d) in SR set: PC = PC + %d + 1\n",
                u8s, s8k,
                u8s, s8k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRBC( char *szOutput_ )
{
    uint8_t u8s = stCPU.s;
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brbc %d, %d               \t ; Branch if Bit (%d) in SR clear: PC = PC + %d + 1\n",
                u8s, s8k,
                u8s, s8k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BREQ( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "breq %d                   \t ; Branch if zero flag set: PC = PC + %d + 1\n",
                s8k,
                s8k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRNE( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brne %d                   \t ; Branch if zero flag clear: PC = PC + %d + 1\n",
                s8k,
                s8k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRCS( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
```

```
    sprintf( szOutput_, "brcs %d                    \t ; Branch if carry flag set: PC = PC + %d + 1\n",
                s8k,
                s8k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRCC( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brcc %d                    \t ; Branch if carry flag clear: PC = PC + %d + 1\n",
                s8k,
                s8k
                );

}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRSH( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brsh %d                    \t ; Branch if same or higher: PC = PC + %d + 1\n",
                s8k,
                s8k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRLO( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brlo %d                    \t ; Branch if lower: PC = PC + %d + 1\n",
                s8k,
                s8k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRMI( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brmi %d                    \t ; Branch if minus: PC = PC + %d + 1\n",
                s8k,
                s8k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRPL( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brpl %d                    \t ; Branch if plus: PC = PC + %d + 1\n",
                s8k,
                s8k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRGE( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brge %d                    \t ; Branch if greater-or-equal (signed): PC = PC + %d + 1\n",
                s8k,
                s8k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRLT( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brlt %d                    \t ; Branch if less-than (signed): PC = PC + %d + 1\n",
```

```
                s8k,
                s8k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRHS( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brlt %d                   \t ; Branch if half-carry set: PC = PC + %d + 1\n",
                s8k,
                s8k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRHC( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brhc %d                   \t ; Branch if half-carry clear: PC = PC + %d + 1\n",
                s8k,
                s8k
                );

}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRTS( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brts %d                   \t ; Branch if T-flag set: PC = PC + %d + 1\n",
                s8k,
                s8k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRTC( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brtc %d                   \t ; Branch if T-flag clear: PC = PC + %d + 1\n",
                s8k,
                s8k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRVS( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brvs %d                   \t ; Branch if Overflow set: PC = PC + %d + 1\n",
                s8k,
                s8k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRVC( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brvc %d                   \t ; Branch if Overflow clear: PC = PC + %d + 1\n",
                s8k,
                s8k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRIE( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brie %d                   \t ; Branch if Interrupt Enabled: PC = PC + %d + 1\n",
                s8k,
                s8k
```

```
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BRID( char *szOutput_ )
{
    int8_t  s8k = stCPU.k_s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "brid %d                  \t ; Branch if Interrupt Disabled: PC = PC + %d + 1\n",
                s8k,
                s8k
                );

}

//---------------------------------------------------------------------------
static void AVR_Disasm_MOV( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8Rr = Register_From_Rr();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "mov r%d, r%d             \t ; Copy Register: r%d = r%d\n",
                u8Rd, u8Rr,
                u8Rd, u8Rr
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_MOVW( char *szOutput_ )
{
    uint16_t u16Rd = Register_From_Rd16();
    uint16_t u16Rr = Register_From_Rr16();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "movw r%d:r%d, r%d:r%d     \t ; Copy Register (Word): r%d:r%d = r%d:r%d\n",
                u16Rd+1, u16Rd, u16Rr+1, u16Rr,
                u16Rd+1, u16Rd, u16Rr+1, u16Rr
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LDI( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8K = stCPU.K;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ldi r%d, %d               \t ; Load Immediate: r%d = %d\n",
                u8Rd, u8K,
                u8Rd, u8K
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LDS( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint16_t u16k = stCPU.k;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "lds r%d, %d               \t ; Load Direct from Data Space: r%d = (%d)\n",
                u8Rd, u16k,
                u8Rd, u16k
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LD_X_Indirect( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ld r%d, X                 \t ; Load Indirect from Data Space\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LD_X_Indirect_Postinc( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ld r%d, X+                \t ; Load Indirect from Data Space w/Postincrement\n",
                u8Rd
```

```
            );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LD_X_Indirect_Predec( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ld r%d, -X                \t ; Load Indirect from Data Space w/Predecrement\n",
            u8Rd
            );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LD_Y_Indirect( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ld r%d, Y                 \t ; Load Indirect from Data Space\n",
            u8Rd
            );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LD_Y_Indirect_Postinc( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ld r%d, Y+                \t ; Load Indirect from Data Space w/Postincrement\n",
            u8Rd
            );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LD_Y_Indirect_Predec( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ld r%d, -Y                \t ; Load Indirect from Data Space w/Predecrement\n",
            u8Rd
            );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LDD_Y( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8q = stCPU.q;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ldd r%d, Y+%d             \t ; Load Indirect from Data Space (with Displacement)\n",
            u8Rd, u8q
            );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LD_Z_Indirect( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ld r%d, Z                 \t ; Load Indirect from Data Space\n",
            u8Rd
            );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LD_Z_Indirect_Postinc( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ld r%d, Z+                \t ; Load Indirect from Data Space w/Postincrement\n",
            u8Rd
            );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LD_Z_Indirect_Predec( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

```

```
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ld r%d, -Z                \t ; Load Indirect from Data Space w/Predecrement\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LDD_Z( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8q = stCPU.q;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ldd r%d, Z+%d             \t ; Load Indirect from Data Space (with Displacement)\n",
                u8Rd, u8q
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_STS( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint16_t u16k = stCPU.k;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sts %d, r%d               \t ; Store Direct to Data Space: (%d) = r%d\n",
                u16k, u8Rd,
                u16k, u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ST_X_Indirect( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "st X, r%d                 \t ; Store Indirect\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ST_X_Indirect_Postinc( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "st X+, r%d                \t ; Store Indirect w/Postincrement \n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ST_X_Indirect_Predec( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "st -X, r%d                \t ; Store Indirect w/Predecrement\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ST_Y_Indirect( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "st Y, r%d                 \t ; Store Indirect\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ST_Y_Indirect_Postinc( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "st Y+, r%d                \t ; Store Indirect w/Postincrement \n",
                u8Rd
                );
}

```

```
//---------------------------------------------------------------------------
static void AVR_Disasm_ST_Y_Indirect_Predec( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "st -Y, r%d                \t ; Store Indirect w/Predecrement\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_STD_Y( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8q = stCPU.q;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "std Y+%d, r%d             \t ; Store Indirect from Data Space (with Displacement)\n",
                u8q, u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ST_Z_Indirect( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "st Z, r%d                 \t ; Store Indirect\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ST_Z_Indirect_Postinc( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "st Z+, r%d                \t ; Store Indirect w/Postincrement \n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ST_Z_Indirect_Predec( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "st -Z, r%d                \t ; Store Indirect w/Predecrement\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_STD_Z( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8q = stCPU.q;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "std Z+%d, r%d             \t ; Store Indirect from Data Space (with Displacement)\n",
                u8q, u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LPM( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "lpm                       \t ; Load Program Memory: r0 = (Z)\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LPM_Z( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "lpm r%d, Z                \t ; Load Program Memory: r%d = (Z)\n",
                u8Rd,
                u8Rd
                );
```

```
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LPM_Z_Postinc( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "lpm r%d, Z+                 \t ; Load Program Memory with Postincrement: r%d = (Z), Z = Z + 1\n",
                u8Rd,
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ELPM( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "elpm                        \t ; (Extended) Load Program Memory: r0 = (Z)\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ELPM_Z( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "elpm r%d, Z                 \t ; (Extended) Load Program Memory: r%d = (Z)\n",
                u8Rd,
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ELPM_Z_Postinc( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "elpm r%d, Z+                \t ; (Extended) Load Program Memory w/Postincrement: r%d = (Z), Z = Z + 1\n",
                u8Rd,
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SPM( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "spm                         \t ; Store Program Memory\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SPM_Z_Postinc2( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "spm Z+                      \t ; Store Program Memory Z = Z + 2 \n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_IN( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8A = stCPU.A;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "in r%d, %d                  \t ; Load an I/O location to register\n",
                u8Rd,
                u8A
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_OUT( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8A = stCPU.A;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "out %d, r%d                 \t ; Load an I/O location to register\n",
                u8A,
                u8Rd
                );

}
```

```
//---------------------------------------------------------------------------
static void AVR_Disasm_LAC( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "lac Z, r%d                   \t ; Load And Clear\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LAS( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "las Z, r%d                   \t ; Load And Set\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LAT( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "lat Z, r%d                   \t ; Load And Toggle\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LSL( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "lsl r%d                   \t ; Logical shift left r%d by 1 bit\n",
                u8Rd,
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_LSR( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "lsr r%d                   \t ; Logical shift right r%d by 1 bit\n",
                u8Rd,
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_POP( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "pop r%d                   \t ; Pop byte from stack into r%d\n",
                u8Rd,
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_PUSH( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "push r%d                  \t ; Push register r%d to stack\n",
                u8Rd,
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ROL( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

```

```
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "rol r%d                   \t ; Rotate Left through Carry\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ROR( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ror r%d                   \t ; Rotate Right through Carry\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_ASR( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "asr r%d                   \t ; Arithmatic Shift Right\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SWAP( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "swap r%d                  \t ; Swap high/low Nibbles in Register\n",
                u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BSET( char *szOutput_ )
{
    uint8_t u8s = stCPU.s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "bset %d                   \t ; Set bit %d in status register\n",
                u8s,
                u8s
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BCLR( char *szOutput_ )
{
    uint8_t u8s = stCPU.s;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "bclr %d                   \t ; Clear bit %d in status register\n",
                u8s,
                u8s
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SBI( char *szOutput_ )
{
    uint8_t u8b = stCPU.b;
    uint8_t u8A = stCPU.A;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sbi %d, %d                \t ; Set bit in I/O register\n",
                u8A,
                u8b
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_CBI( char *szOutput_ )
{
    uint8_t u8s = stCPU.b;
    uint8_t u8A = stCPU.A;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "cbi %d, %d                \t ; Clear bit in I/O register\n",
                u8A,
                u8s
                );
```

```
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BST( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8b = stCPU.b;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "bst r%d, %d               \t ; Store Bit %d of r%d in the T register\n",
                u8Rd, u8b,
                u8b, u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BLD( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();
    uint8_t u8b = stCPU.b;

    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "bld r%d, %d               \t ; Load the T register into Bit %d of r%d\n",
                u8Rd, u8b,
                u8b, u8Rd
                );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SEC( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sec                       \t ; Set the carry flag in the SR\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_CLC( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "clc                       \t ; Clear the carry flag in the SR\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SEN( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sen                       \t ; Set the negative flag in the SR\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_CLN( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "cln                       \t ; Clear the negative flag in the SR\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SEZ( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sez                       \t ; Set the zero flag in the SR\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_CLZ( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "clz                       \t ; Clear the zero flag in the SR\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SEI( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sei                       \t ; Enable MCU interrupts\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_CLI( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "cli                       \t ; Disable MCU interrupts\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SES( char *szOutput_ )
{
```

```
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "ses                       \t ; Set the sign flag in the SR\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_CLS( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "cls                       \t ; Clear the sign flag in the SR\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SEV( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sev                       \t ; Set the overflow flag in the SR\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_CLV( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "clv                       \t ; Clear the overflow flag in the SR\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SET( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "set                       \t ; Set the T-flag in the SR\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_CLT( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "clt                       \t ; Clear the T-flag in the SR\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SEH( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "seh                       \t ; Set half-carry flag in SR\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_CLH( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "clh                       \t ; Clear half-carry flag in SR\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_BREAK( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "break                     \t ; Halt for debugger\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_NOP( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "nop                       \t ; Do nothing\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_SLEEP( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "sleep                     \t ; Put MCU into sleep mode\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_WDR( char *szOutput_ )
{
    //ruler: 0----5----10---15---20---25---30---35---40" );
    sprintf( szOutput_, "wdr                       \t ; Reset Watchdog Timer\n" );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_XCH( char *szOutput_ )
{
    uint8_t u8Rd = Register_From_Rd();

    //ruler: 0----5----10---15---20---25---30---35---40" );
```

```
    sprintf( szOutput_, "xch Z, r%d                \t ; Exchange registers w/memory\n",
                  u8Rd
                  );
}

//---------------------------------------------------------------------------
static void AVR_Disasm_Unimplemented( char *szOutput_ )
{
    sprintf( szOutput_, ".db 0x%04X ; Data (not an opcode)\n", stCPU.pu16ROM[ stCPU.u16PC ] );
}

//---------------------------------------------------------------------------
AVR_Disasm AVR_Disasm_Function( uint16_t OP_ )
{
    // Special instructions - "static" encoding
    switch (OP_)
    {
    case 0x0000: return AVR_Disasm_NOP;

    case 0x9408: return AVR_Disasm_SEC;
    case 0x9409: return AVR_Disasm_IJMP;
    case 0x9418: return AVR_Disasm_SEZ;
    case 0x9419: return AVR_Disasm_EIJMP;
    case 0x9428: return AVR_Disasm_SEN;
    case 0x9438: return AVR_Disasm_SEV;
    case 0x9448: return AVR_Disasm_SES;
    case 0x9458: return AVR_Disasm_SEH;
    case 0x9468: return AVR_Disasm_SET;
    case 0x9478: return AVR_Disasm_SEI;

    case 0x9488: return AVR_Disasm_CLC;
    case 0x9498: return AVR_Disasm_CLZ;
    case 0x94A8: return AVR_Disasm_CLN;
    case 0x94B8: return AVR_Disasm_CLV;
    case 0x94C8: return AVR_Disasm_CLS;
    case 0x94D8: return AVR_Disasm_CLH;
    case 0x94E8: return AVR_Disasm_CLT;
    case 0x94F8: return AVR_Disasm_CLI;

    case 0x9508: return AVR_Disasm_RET;
    case 0x9509: return AVR_Disasm_ICALL;
    case 0x9518: return AVR_Disasm_RETI;
    case 0x9519: return AVR_Disasm_EICALL;
    case 0x9588: return AVR_Disasm_SLEEP;
    case 0x9598: return AVR_Disasm_BREAK;
    case 0x95A8: return AVR_Disasm_WDR;
    case 0x95C8: return AVR_Disasm_LPM;
    case 0x95D8: return AVR_Disasm_ELPM;
    case 0x95E8: return AVR_Disasm_SPM;
    case 0x95F8: return AVR_Disasm_SPM_Z_Postinc2;
    }

#if 0
    // Note: These disasm handlers are generalized versions of specific mnemonics in the above list.
    // For disassembly, it's probably easier to read the output from the more "spcific" mnemonics, so
    // those are used.  For emulation, using the generalized functions may be more desirable.
    switch( OP_ & 0xFF8F)
    {
    case 0x9408: return AVR_Disasm_BSET;
    case 0x9488: return AVR_Disasm_BCLR;
    }
#endif

    switch (OP_ & 0xFF88)
    {
    case 0x0300: return AVR_Disasm_MULSU;
    case 0x0308: return AVR_Disasm_FMUL;
    case 0x0380: return AVR_Disasm_FMULS;
    case 0x0388: return AVR_Disasm_FMULSU;
    }

    switch (OP_ & 0xFF0F)
    {
    case 0x940B: return AVR_Disasm_DES;
    case 0xEF0F: return AVR_Disasm_SER;
    }

    switch (OP_ & 0xFF00)
    {
    case 0x0100: return AVR_Disasm_MOVW;
    case 0x9600: return AVR_Disasm_ADIW;
    case 0x9700: return AVR_Disasm_SBIW;

    case 0x9800: return AVR_Disasm_CBI;
    case 0x9900: return AVR_Disasm_SBIC;
    case 0x9A00: return AVR_Disasm_SBI;
    case 0x9B00: return AVR_Disasm_SBIS;
```

```
    }

    switch (OP_ & 0xFE0F)
    {
    case 0x8008: return AVR_Disasm_LD_Y_Indirect;
    case 0x8000: return AVR_Disasm_LD_Z_Indirect;
    case 0x8200: return AVR_Disasm_ST_Z_Indirect;
    case 0x8208: return AVR_Disasm_ST_Y_Indirect;

    // -- Single 5-bit register...
    case 0x9000: return AVR_Disasm_LDS;
    case 0x9001: return AVR_Disasm_LD_Z_Indirect_Postinc;
    case 0x9002: return AVR_Disasm_LD_Z_Indirect_Predec;
    case 0x9004: return AVR_Disasm_LPM_Z;
    case 0x9005: return AVR_Disasm_LPM_Z_Postinc;
    case 0x9006: return AVR_Disasm_ELPM_Z;
    case 0x9007: return AVR_Disasm_ELPM_Z_Postinc;
    case 0x9009: return AVR_Disasm_LD_Y_Indirect_Postinc;
    case 0x900A: return AVR_Disasm_LD_Y_Indirect_Predec;
    case 0x900C: return AVR_Disasm_LD_X_Indirect;
    case 0x900D: return AVR_Disasm_LD_X_Indirect_Postinc;
    case 0x900E: return AVR_Disasm_LD_X_Indirect_Predec;
    case 0x900F: return AVR_Disasm_POP;

    case 0x9200: return AVR_Disasm_STS;
    case 0x9201: return AVR_Disasm_ST_Z_Indirect_Postinc;
    case 0x9202: return AVR_Disasm_ST_Z_Indirect_Predec;
    case 0x9204: return AVR_Disasm_XCH;
    case 0x9205: return AVR_Disasm_LAS;
    case 0x9206: return AVR_Disasm_LAC;
    case 0x9207: return AVR_Disasm_LAT;
    case 0x9209: return AVR_Disasm_ST_Y_Indirect_Postinc;
    case 0x920A: return AVR_Disasm_ST_Y_Indirect_Predec;
    case 0x920C: return AVR_Disasm_ST_X_Indirect;
    case 0x920D: return AVR_Disasm_ST_X_Indirect_Postinc;
    case 0x920E: return AVR_Disasm_ST_X_Indirect_Predec;
    case 0x920F: return AVR_Disasm_PUSH;

    // -- One-operand instructions
    case 0x9400: return AVR_Disasm_COM;
    case 0x9401: return AVR_Disasm_NEG;
    case 0x9402: return AVR_Disasm_SWAP;
    case 0x9403: return AVR_Disasm_INC;
    case 0x9405: return AVR_Disasm_ASR;
    case 0x9406: return AVR_Disasm_LSR;
    case 0x9407: return AVR_Disasm_ROR;
    case 0x940A: return AVR_Disasm_DEC;

    }
    switch (OP_ & 0xFE0E)
    {
    case 0x940C: return AVR_Disasm_JMP;
    case 0x940E: return AVR_Disasm_CALL;
    }

    switch (OP_ & 0xFE08)
    {

    // -- BLD/BST Encoding
    case 0xF800: return AVR_Disasm_BLD;
    case 0xFA00: return AVR_Disasm_BST;
    // -- SBRC/SBRS Encoding
    case 0xFC00: return AVR_Disasm_SBRC;
    case 0xFE00: return AVR_Disasm_SBRS;
    }

    switch (OP_ & 0xFC07)
    {
    // -- Conditional branches
    case 0xF000: return AVR_Disasm_BRCS;
    // case 0xF000: return AVR_Disasm_BRLO;            // AKA AVR_Disasm_BRCS;
    case 0xF001: return AVR_Disasm_BREQ;
    case 0xF002: return AVR_Disasm_BRMI;
    case 0xF003: return AVR_Disasm_BRVS;
    case 0xF004: return AVR_Disasm_BRLT;
    case 0xF006: return AVR_Disasm_BRTS;
    case 0xF007: return AVR_Disasm_BRIE;
    case 0xF400: return AVR_Disasm_BRCC;
    // case 0xF400: return AVR_Disasm_BRSH;            // AKA AVR_Disasm_BRCC;
    case 0xF401: return AVR_Disasm_BRNE;
    case 0xF402: return AVR_Disasm_BRPL;
    case 0xF403: return AVR_Disasm_BRVC;
    case 0xF404: return AVR_Disasm_BRGE;
    case 0xF405: return AVR_Disasm_BRHC;
    case 0xF406: return AVR_Disasm_BRTC;
    case 0xF407: return AVR_Disasm_BRID;
    }
```

```
    switch (OP_ & 0xFC00)
    {
    // -- 4-bit register pair
    case 0x0200: return AVR_Disasm_MULS;

    // -- 5-bit register pairs --
    case 0x0400: return AVR_Disasm_CPC;
    case 0x0800: return AVR_Disasm_SBC;
    case 0x0C00: return AVR_Disasm_ADD;
    // case 0x0C00: return AVR_Disasm_LSL; (!! Implemented with: " add rd, rd"
    case 0x1000: return AVR_Disasm_CPSE;
    case 0x1300: return AVR_Disasm_ROL;
    case 0x1400: return AVR_Disasm_CP;
    case 0x1C00: return AVR_Disasm_ADC;
    case 0x1800: return AVR_Disasm_SUB;
    case 0x2000: return AVR_Disasm_AND;
    // case 0x2000: return AVR_Disasm_TST; (!! Implemented with: " and rd, rd"
    case 0x2400: return AVR_Disasm_EOR;
    case 0x2C00: return AVR_Disasm_MOV;
    case 0x2800: return AVR_Disasm_OR;

    // -- 5-bit register pairs -- Destination = R1:R0
    case 0x9C00: return AVR_Disasm_MUL;
    }

    switch (OP_ & 0xF800)
    {
    case  0xB800: return AVR_Disasm_OUT;
    case  0xB000: return AVR_Disasm_IN;
    }

    switch (OP_ & 0xF000)
    {
    // -- Register immediate --
    case 0x3000: return AVR_Disasm_CPI;
    case 0x4000: return AVR_Disasm_SBCI;
    case 0x5000: return AVR_Disasm_SUBI;
    case 0x6000: return AVR_Disasm_ORI;// return AVR_Disasm_SBR;
    case 0x7000: return AVR_Disasm_ANDI;

    //-- 12-bit immediate
    case 0xC000: return AVR_Disasm_RJMP;
    case 0xD000: return AVR_Disasm_RCALL;

    // -- Register immediate
    case 0xE000: return AVR_Disasm_LDI;
    }

    switch (OP_ & 0xD208)
    {
    // -- 7-bit signed offset
    case 0x8000: return AVR_Disasm_LDD_Z;
    case 0x8008: return AVR_Disasm_LDD_Y;
    case 0x8200: return AVR_Disasm_STD_Z;
    case 0x8208: return AVR_Disasm_STD_Y;
    }

    return AVR_Disasm_Unimplemented;
}

```

4.13 avr_disasm.h File Reference

AVR Disassembler Implementation.

`#include "avr_opcodes.h"`

Typedefs

- typedef void(∗ **AVR_Disasm**)(char ∗szOutput_)

Functions

- AVR_Disasm 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_Disasm 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 1637 of file avr_disasm.c.

4.14 avr_disasm.h

```
/*****************************************************************************
 *     (     (                      (     |
 *    )\ )  )\ )    (             )\ )   |
 *   (()/( (()/(    )\     (   ( (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))    | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))   | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \  | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /  | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\  |
 *                                       | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __AVR_DISASM_H__
#define __AVR_DISASM_H__

#include "avr_opcodes.h"

//---------------------------------------------------------------------------
// Format opcode function for disassembly
typedef void (*AVR_Disasm)( char *szOutput_ );

//---------------------------------------------------------------------------
AVR_Disasm 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"
#include "interrupt_callout.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 59 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 67 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 47 of file avr_interrupt.c.

4.16 avr_interrupt.c

```c
/*****************************************************************************
 *     (     (                      (    |
 *    )\ )  )\ )    (              )\ )  |
 *   (()/( (()/(    )\     (   (  (()/(  | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))    | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))   | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \  | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /  | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\  |
 *                                       | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#include <stdint.h>
#include "emu_config.h"
#include "avr_cpu.h"
#include "interrupt_callout.h"

//---------------------------------------------------------------------------
static void AVR_NextInterrupt(void)
{
    uint32_t i = 0x80000000;
    uint32_t j = 31;
    while (i)
    {
        if ((stCPU.u32IntFlags & i) == i)
        {
            stCPU.u8IntPriority = j;
            return;
        }
        i >>= 1;
        j--;
    }

    stCPU.u8IntPriority = 255;
    stCPU.u32IntFlags = 0;
}

//---------------------------------------------------------------------------
void AVR_InterruptCandidate( uint8_t u8Vector_ )
{
    // Interrupts are prioritized by index -- lower == higher priority.
    // Candidate is the lowest
    if (u8Vector_ < stCPU.u8IntPriority)
    {
        stCPU.u8IntPriority = u8Vector_;
    }
    stCPU.u32IntFlags |= (1 << u8Vector_);
}

//---------------------------------------------------------------------------
void AVR_ClearCandidate( uint8_t u8Vector_ )
{
    stCPU.u32IntFlags &= ~(1 << u8Vector_ );
    AVR_NextInterrupt();
}


//---------------------------------------------------------------------------
void AVR_Interrupt( void )
{
    // First - check to see if there's an interrupt pending.
    if (stCPU.u8IntPriority == 255 || stCPU.pstRAM->stRegisters.SREG.I == 0)
    {
        return; // no interrupt pending
    }

    // Push the current PC to stack.
    uint16_t u16SP = (((uint16_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
                     (((uint16_t)stCPU.pstRAM->stRegisters.SPL.r));

    uint16_t u16StoredPC = stCPU.u16PC;

    stCPU.pstRAM->au8RAM[ u16SP ]     = (uint8_t)(u16StoredPC & 0x00FF);
    stCPU.pstRAM->au8RAM[ u16SP - 1 ] = (uint8_t)(u16StoredPC >> 8);

    // Stack is post-decremented
    u16SP -= 2;

    // Store the new SP.
    stCPU.pstRAM->stRegisters.SPH.r = (u16SP >> 8);
    stCPU.pstRAM->stRegisters.SPL.r = (u16SP & 0x00FF);

```

```
    // Read the new PC from the vector table
    uint16_t u16NewPC = (uint16_t)(stCPU.u8IntPriority * 2);

    // Set the new PC
    stCPU.u16PC = u16NewPC;
    stCPU.u16ExtraPC = 0;

    // Clear the "I" (global interrupt enabled) register in the SR
    stCPU.pstRAM->stRegisters.SREG.I = 0;

    // Run the interrupt-acknowledge callback associated with this vector
    uint8_t u8Pri = stCPU.u8IntPriority;
    if (u8Pri < 32 && stCPU.apfInterruptCallbacks[ u8Pri ])
    {
        stCPU.apfInterruptCallbacks[ u8Pri ]( u8Pri );
    }

    // Reset the CPU interrupt priority
    stCPU.u32IntFlags &= ~(1 << u8Pri);
    AVR_NextInterrupt();

    // Run the generic interrupt callout routine
    InterruptCallout_Run( true, u8Pri );

    // Clear any sleep-mode flags currently set
    stCPU.bAsleep = false;
}
```

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 59 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 67 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 47 of file avr_interrupt.c.

4.18 avr_interrupt.h

```
/*****************************************************************************
 *     (     (                      (    |
 *    )\ )  )\ )    (              )\ ) |
 *   (()/( (()/(    )\     (   (  (()/( | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))   | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))   | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \  | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /  | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\  |
 *                                       | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __AVR_INTERRUPT_H__
#define __AVR_INTERRUPT_H__

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

//---------------------------------------------------------------------------
void AVR_InterruptCandidate( uint8_t u8Vector_ );

//---------------------------------------------------------------------------
void AVR_ClearCandidate( uint8_t u8Vector_ );

//---------------------------------------------------------------------------
void AVR_Interrupt( void );

#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 2014-15, 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     {
```

```
        return;
    }

    node->next = stCPU.apstPeriphReadTable[addr_];
    node->pfReader = pstPeriph_->pfRead;
    node->pvContext = pstPeriph_->pvContext;

    stCPU.apstPeriphReadTable[addr_] = node;
}

//---------------------------------------------------------------------------
void IO_AddWriter( AVRPeripheral *pstPeriph_, uint8_t addr_)
{
    IOWriterList *node = NULL;

    node = (IOWriterList*)malloc(sizeof(*node));
    if (!node)
    {
        return;
    }

    node->next = stCPU.apstPeriphWriteTable[addr_];
    node->pfWriter = pstPeriph_->pfWrite;
    node->pvContext = pstPeriph_->pvContext;

    stCPU.apstPeriphWriteTable[addr_] = node;
}

//---------------------------------------------------------------------------
void IO_AddClocker( AVRPeripheral *pstPeriph_ )
{
    IOClockList *node = NULL;

    node = (IOClockList*)malloc(sizeof(*node));
    if (!node)
    {
        return;
    }

    node->next = stCPU.pstClockList;
    node->pfClock = pstPeriph_->pfClock;
    node->pvContext = pstPeriph_->pvContext;

    stCPU.pstClockList = node;
}

//---------------------------------------------------------------------------
void IO_Write( uint8_t addr_, uint8_t value_ )
{
    IOWriterList *node = stCPU.apstPeriphWriteTable[addr_];
    while (node)
    {
        if (node->pfWriter)
        {
            node->pfWriter( node->pvContext, addr_, value_ );
        }
        node = node->next;
    }
}

//---------------------------------------------------------------------------
void IO_Read( uint8_t addr_, uint8_t *value_ )
{
    IOReaderList *node = stCPU.apstPeriphReadTable[addr_];
    while (node)
    {
        if (node->pfReader)
        {
            node->pfReader( node->pvContext, addr_, value_ );
        }
        node = node->next;
    }
}

//---------------------------------------------------------------------------
void IO_Clock( void )
{
    IOClockList *node = stCPU.pstClockList;
    while (node)
    {
        if (node->pfClock)
        {
            node->pfClock( node->pvContext );
        }
        node = node->next;
    }
}
```

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

```
/****************************************************************************
 *     (     (                       (     |
 *    )\ )  )\ )     (              )\ )   |
 *   (()/( (()/(     )\      (   (  (()/(  | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] --------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ----------------------
 *   | |_  | |     (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__    / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                         | "Yeah, it does Arduino..."
 * ----------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __AVR_IO_H__
#define __AVR_IO_H__
```

```

#include "avr_peripheral.h"

//---------------------------------------------------------------------------
typedef struct _IOReaderList
{
    struct _IOReaderList *next;
    void *pvContext;
    PeriphRead pfReader;
} IOReaderList;

//---------------------------------------------------------------------------
typedef struct _IOWriterList
{
    struct _IOWriterList *next;
    void *pvContext;
    PeriphWrite pfWriter;
} IOWriterList;

//---------------------------------------------------------------------------
typedef struct _IOClockList
{
    struct _IOClockList *next;
    void *pvContext;
    PeriphClock pfClock;
} IOClockList;

//---------------------------------------------------------------------------
void IO_AddReader( AVRPeripheral *pstPeriph_, uint8_t addr_);

//--------------------------------------------------------------------------
void IO_AddWriter( AVRPeripheral *pstPeriph_, uint8_t addr_);

//--------------------------------------------------------------------------
void IO_AddClocker( AVRPeripheral *pstPeriph_ );

//--------------------------------------------------------------------------
void IO_Write( uint8_t addr_, uint8_t value_ );

//---------------------------------------------------------------------------
void IO_Read( uint8_t addr_, uint8_t *value_ );

//---------------------------------------------------------------------------
void IO_Clock( void );

#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"
#include "elf_types.h"
#include "elf_process.h"
#include "elf_print.h"
#include "debug_sym.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.
- static void **AVR_Load_ELF_Symbols** (const uint8_t ∗pau8Buffer_)

- bool AVR_Load_ELF (const char ∗szFilePath_)

  AVR_Load_ELF Load an elf 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_ELF (const char ∗ *szFilePath_*)

AVR_Load_ELF Load an elf file, specified by path, into the flash memory of the CPU object.

Will also pre-seed RAM according to the contents of the ELF, if found.

Parameters

*szFilePath_*	Pointer to the elf-file path

Returns

true if the elf file load operation succes

Definition at line 142 of file avr_loader.c.

4.23.2.2 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 54 of file avr_loader.c.

4.24 avr_loader.c

```
/***************************************************************************
 *     (     (                      (    |
 *   )\ )  )\ )    (             )\ )   |
 *   (()/( (()/(    )\      (   (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                       | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#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"

#include "elf_types.h"
#include "elf_process.h"
#include "elf_print.h"

#include "debug_sym.h"

//---------------------------------------------------------------------------
static void AVR_Copy_Record( HEX_Record_t *pstHex_)
{
    uint16_t u16Data;
    uint16_t i;
    for (i = 0; i < pstHex_->u8ByteCount; i += 2)
    {
        u16Data = pstHex_->u8Data[i+1];
        u16Data <<= 8;
        u16Data |= pstHex_->u8Data[i];

        stCPU.pu16ROM[(pstHex_->u16Address + i) >> 1] = u16Data;
    }
}

//---------------------------------------------------------------------------
bool AVR_Load_HEX( const char *szFilePath_)
{
    HEX_Record_t stRecord;
    uint32_t u32Addr = 0;
    int fd = -1;

    if (!szFilePath_)
    {
        fprintf(stderr, "No programming file specified\n");
        return false;
    }

    fd = open(szFilePath_, O_RDONLY);

    if (-1 == fd)
    {
        fprintf(stderr, "Unable to open file\n");
        return false;
    }

    bool rc = true;

    while (rc)
    {
        rc = HEX_Read_Record(fd, &stRecord);
        if (RECORD_EOF == stRecord.u8RecordType)
        {
            break;
        }
        if (RECORD_DATA == stRecord.u8RecordType)
        {
            AVR_Copy_Record(&stRecord);
        }
    }

cleanup:
    close(fd);
    return rc;
}

//---------------------------------------------------------------------------
static void AVR_Load_ELF_Symbols( const uint8_t *pau8Buffer_ )
{
    // Get a pointer to the section header for the symbol table
    uint32_t u32Offset = ELF_GetSymbolTableOffset( pau8Buffer_ );
    if (u32Offset == 0)
    {
        printf( "No debug symbol, bailing\n");
        return;
    }
    ElfSectionHeader_t *pstSymHeader = (ElfSectionHeader_t*)(&pau8Buffer_[u32Offset]);

    // Get a pointer to the section header for the symbol table's strings
    u32Offset = ELF_GetSymbolStringTableOffset( pau8Buffer_ );
    if (u32Offset == 0)
    {
        printf( "No debug symbol strings, bailing\n");
        return;
    }
```

```
    ElfSectionHeader_t *pstStrHeader = (ElfSectionHeader_t*)(&pau8Buffer_[u32Offset]);

    // Iterate through the symbol table section, printing out the details of each.
    uint32_t u32SymOffset = pstSymHeader->u32Offset;
    ElfSymbol_t *pstSymbol = (ElfSymbol_t*)(&pau8Buffer_[u32SymOffset]);

    while (u32SymOffset < (pstSymHeader->u32Offset + pstSymHeader->u32Size))
    {
        uint8_t u8Type = pstSymbol->u8Info & 0x0F;
        if (u8Type == 2)
        {
            // Note that elf file uses byte addressing, and we use 16-bit word addressing
            Symbol_Add_Func( &pau8Buffer_[pstSymbol->u32Name + pstStrHeader->u32Offset],
                             pstSymbol->u32Value >> 1,
                             pstSymbol->u32Size >> 1);
        }
        else if (u8Type == 1)
        {
            // The elf files use 0x0080XXXX as an offset for dat objects.  Mask here
            Symbol_Add_Obj( &pau8Buffer_[pstSymbol->u32Name + pstStrHeader->u32Offset],
                             pstSymbol->u32Value & 0x0000FFFF,
                             pstSymbol->u32Size );
        }
        u32SymOffset += pstSymHeader->u32EntrySize;
        pstSymbol = (ElfSymbol_t*)(&pau8Buffer_[u32SymOffset]);
    }
}

//---------------------------------------------------------------------------
bool AVR_Load_ELF( const char *szFilePath_)
{
    uint8_t *pu8Buffer;

    // Load the ELF Binary from into a newly-created local buffer
    if (0 != ELF_LoadFromFile(&pu8Buffer, szFilePath_))
    {
        return false;
    }

    // Loaded ELF successfully, load program sections into AVR memory.
    ElfHeader_t *pstHeader = (ElfHeader_t*)(pu8Buffer);
    uint32_t     u32Offset = pstHeader->u32PHOffset;
    uint32_t     u32MaxOffset = pstHeader->u32PHOffset
                             + (pstHeader->u16PHNum * pstHeader->u16PHSize);

    // Iterate through every program header section in the elf-file
    while (u32Offset < u32MaxOffset)
    {
        ElfProgramHeader_t *pstPHeader = (ElfProgramHeader_t*)(&pu8Buffer[u32Offset]);

        // RAM encoded in ELF file using addresses >= 0x00800000
        if (pstPHeader->u32PhysicalAddress >= 0x00800000)
        {
            // Clear range in segment
            memset( &(stCPU.pstRAM->au8RAM[pstPHeader->u32PhysicalAddress & 0x0000FFFF]),
                    0,
                    pstPHeader->u32MemSize );
            // Copy program segment from ELF into CPU RAM
            memcpy( &(stCPU.pstRAM->au8RAM[pstPHeader->u32PhysicalAddress & 0x0000FFFF]),
                    &pu8Buffer[pstPHeader->u32Offset],
                    pstPHeader->u32FileSize );
        }
        else
        {
            // Clear range in segment
            memset( &(stCPU.pu16ROM[pstPHeader->u32PhysicalAddress >> 1]),
                    0,
                    pstPHeader->u32MemSize );

            // Copy program segment from ELF into CPU Flash
            memcpy( &(stCPU.pu16ROM[pstPHeader->u32PhysicalAddress >> 1]),
                    &pu8Buffer[pstPHeader->u32Offset],
                    pstPHeader->u32FileSize );
        }

        // Next Section...
        u32Offset += pstHeader->u16PHSize;
    }

    AVR_Load_ELF_Symbols( pu8Buffer );

    free( pu8Buffer );
    return true;
}
```

4.25 avr_loader.h File Reference

Functions to load intel hex or elf binaries 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.
- bool AVR_Load_ELF (const char *szFilePath_)

  AVR_Load_ELF Load an elf file, specified by path, into the flash memory of the CPU object.

4.25.1 Detailed Description

Functions to load intel hex or elf binaries into a virtual AVR.

Definition in file avr_loader.h.

4.25.2 Function Documentation

4.25.2.1 bool AVR_Load_ELF (const char * *szFilePath_*)

AVR_Load_ELF Load an elf file, specified by path, into the flash memory of the CPU object.

Will also pre-seed RAM according to the contents of the ELF, if found.

Parameters

*szFilePath_*	Pointer to the elf-file path

Returns

true if the elf file load operation succes

Definition at line 142 of file avr_loader.c.

4.25.2.2 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 54 of file avr_loader.c.

4.26 avr_loader.h

```
/*****************************************************************************
 *     (     (                          (     |
 *   )\ )  )\ )    (                 )\ )   |
 *  (()/( (()/(    )\     (   (   (()/(    | -- [ Funkenstein ] -------------
 *   /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] -------------------
 *  (_))_|(_))   )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
 *  | |_  | |    (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *  | __| | |__   / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
 *  |_|   |____| /_/ \_\   \_/   |_|_\    |
 *                                        | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __AVR_LOADER_H__
#define __AVR_LOADER_H__

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

//---------------------------------------------------------------------------
bool AVR_Load_HEX( const char *szFilePath_);

//---------------------------------------------------------------------------
bool AVR_Load_ELF( const char *szFilePath_);
#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

```
/****************************************************************************
 *     (      (                      (      |
 *    )\ )   )\ )     (             )\ )    |
 *   (()/(  (()/(    )\       (   (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_))  /(_))((((_)()\   )\  /(_))      | -- [ Litle ] -------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ---------------------
 *   | |_   | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *   | __|  | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
 *   |_|    |____|  /_/ \_\   \_/   |_|_\    |
 *                                          | "Yeah, it does Arduino..."
 * ---------------------------------------------+------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#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"

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ADD()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ADC()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ADIW()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SUB()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SUBI()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SBC()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SBCI()
```

```
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SBIW()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_AND()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ANDI()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_OR()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ORI()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_EOR()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_COM()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_NEG()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SBR()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_CBR()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_INC()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_DEC()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_TST()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_CLR()
{
    return 1;
}
```

```
//------------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SER()
{
    return 1;
}

//------------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_MUL()
{
    return 2;
}

//------------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_MULS()
{
    return 2;
}

//------------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_MULSU()
{
    return 2;
}

//------------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_FMUL()
{
    return 2;
}

//------------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_FMULS()
{
    return 2;
}

//------------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_FMULSU()
{
    return 2;
}

//------------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_DES()
{
    return 1;
}

//------------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_RJMP()
{
    return 2;
}

//------------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_IJMP()
{
    return 2;
}

//------------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_EIJMP()
{
    return 2;
}

//------------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_JMP()
{
    return 2;
}

//------------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_RCALL()
{
    return 3;
}

//------------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ICALL()
{
    return 3;
}

//------------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_EICALL()
```

```
{
    return 4;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_CALL()
{
    return 4;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_RET()
{
    return 4;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_RETI()
{
    return 4;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_CPSE()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_CP()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_CPC()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_CPI()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SBRC()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SBRS()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SBIC()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SBIS()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRBS()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRBC()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BREQ()
{
    return 1;
}
```

```c

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRNE()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRCS()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRCC()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRSH()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRLO()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRMI()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRPL()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRGE()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRLT()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRHS()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRHC()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRTS()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRTC()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRVS()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRVC()
```

```
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRIE()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BRID()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_MOV()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_MOVW()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LDI()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LDS()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LD_X_Indirect()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LD_X_Indirect_Postinc()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LD_X_Indirect_Predec()
{
    return 3;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LD_Y_Indirect()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LD_Y_Indirect_Postinc()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LD_Y_Indirect_Predec()
{
    return 3;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LDD_Y()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LD_Z_Indirect()
{
    return 1;
}
```

```
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LD_Z_Indirect_Postinc()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LD_Z_Indirect_Predec()
{
    return 3;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LDD_Z()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_STS()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ST_X_Indirect()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ST_X_Indirect_Postinc()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ST_X_Indirect_Predec()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ST_Y_Indirect()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ST_Y_Indirect_Postinc()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ST_Y_Indirect_Predec()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_STD_Y()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ST_Z_Indirect()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ST_Z_Indirect_Postinc()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ST_Z_Indirect_Predec()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_STD_Z()
```

```
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LPM()
{
    return 3;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LPM_Z()
{
    return 3;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LPM_Z_Postinc()
{
    return 3;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ELPM()
{
    return 3;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ELPM_Z()
{
    return 3;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ELPM_Z_Postinc()
{
    return 3;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SPM()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SPM_Z_Postinc2()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_IN()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_OUT()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LAC()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LAS()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LAT()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LSL()
{
    return 1;
}
```

```

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_LSR()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_POP()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_PUSH()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ROL()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ROR()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_ASR()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SWAP()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BSET()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BCLR()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SBI()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_CBI()
{
    return 2;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BST()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BLD()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SEC()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_CLC()
```

```
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SEN()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_CLN()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SEZ()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_CLZ()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SEI()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_CLI()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SES()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_CLS()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SEV()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_CLV()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SET()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_CLT()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SEH()
{
    return 1;
}

//----------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_CLH()
{
    return 1;
}
```

```
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_BREAK()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_NOP()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_SLEEP()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_WDR()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_XCH()
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Cycles_Unimplemented()
{
    return 1;
}

//---------------------------------------------------------------------------
uint8_t AVR_Opcode_Cycles( uint16_t OP_ )
{
    // Special instructions - "static" encoding
    switch (OP_)
    {
    case 0x0000: return AVR_Opcode_Cycles_NOP();

    case 0x9408: return AVR_Opcode_Cycles_SEC();
    case 0x9409: return AVR_Opcode_Cycles_IJMP();
    case 0x9418: return AVR_Opcode_Cycles_SEZ();
    case 0x9419: return AVR_Opcode_Cycles_EIJMP();
    case 0x9428: return AVR_Opcode_Cycles_SEN();
    case 0x9438: return AVR_Opcode_Cycles_SEV();
    case 0x9448: return AVR_Opcode_Cycles_SES();
    case 0x9458: return AVR_Opcode_Cycles_SEH();
    case 0x9468: return AVR_Opcode_Cycles_SET();
    case 0x9478: return AVR_Opcode_Cycles_SEI();

    case 0x9488: return AVR_Opcode_Cycles_CLC();
    case 0x9498: return AVR_Opcode_Cycles_CLZ();
    case 0x94A8: return AVR_Opcode_Cycles_CLN();
    case 0x94B8: return AVR_Opcode_Cycles_CLV();
    case 0x94C8: return AVR_Opcode_Cycles_CLS();
    case 0x94D8: return AVR_Opcode_Cycles_CLH();
    case 0x94E8: return AVR_Opcode_Cycles_CLT();
    case 0x94F8: return AVR_Opcode_Cycles_CLI();

    case 0x9508: return AVR_Opcode_Cycles_RET();
    case 0x9509: return AVR_Opcode_Cycles_ICALL();
    case 0x9518: return AVR_Opcode_Cycles_RETI();
    case 0x9519: return AVR_Opcode_Cycles_EICALL();
    case 0x9588: return AVR_Opcode_Cycles_SLEEP();
    case 0x9598: return AVR_Opcode_Cycles_BREAK();
    case 0x95A8: return AVR_Opcode_Cycles_WDR();
    case 0x95C8: return AVR_Opcode_Cycles_LPM();
    case 0x95D8: return AVR_Opcode_Cycles_ELPM();
    case 0x95E8: return AVR_Opcode_Cycles_SPM();
    case 0x95F8: return AVR_Opcode_Cycles_SPM_Z_Postinc2();
    }

#if 0
    // Note: These disasm handlers are generalized versions of specific mnemonics in the above list.
    // For disassembly, it's probably easier to read the output from the more "spcific" mnemonics, so
    // those are used.  For emulation, using the generalized functions may be more desirable.
    switch( OP_ & 0xFF8F)
    {
    case 0x9408: return AVR_Opcode_Cycles_BSET();
    case 0x9488: return AVR_Opcode_Cycles_BCLR();
    }
```

```
#endif

    switch (OP_ & 0xFF88)
    {
    case 0x0300: return AVR_Opcode_Cycles_MULSU();
    case 0x0308: return AVR_Opcode_Cycles_FMUL();
    case 0x0380: return AVR_Opcode_Cycles_FMULS();
    case 0x0388: return AVR_Opcode_Cycles_FMULSU();
    }

    switch (OP_ & 0xFF0F)
    {
    case 0x940B: return AVR_Opcode_Cycles_DES();
    case 0xEF0F: return AVR_Opcode_Cycles_SER();
    }

    switch (OP_ & 0xFF00)
    {
    case 0x0100: return AVR_Opcode_Cycles_MOVW();
    case 0x9600: return AVR_Opcode_Cycles_ADIW();
    case 0x9700: return AVR_Opcode_Cycles_SBIW();

    case 0x9800: return AVR_Opcode_Cycles_CBI();
    case 0x9900: return AVR_Opcode_Cycles_SBIC();
    case 0x9A00: return AVR_Opcode_Cycles_SBI();
    case 0x9B00: return AVR_Opcode_Cycles_SBIS();
    }

    switch (OP_ & 0xFE0F)
    {
    case 0x8008: return AVR_Opcode_Cycles_LD_Y_Indirect();
    case 0x8000: return AVR_Opcode_Cycles_LD_Z_Indirect();
    case 0x8200: return AVR_Opcode_Cycles_ST_Z_Indirect();
    case 0x8208: return AVR_Opcode_Cycles_ST_Y_Indirect();

    // -- Single 5-bit register...
    case 0x9000: return AVR_Opcode_Cycles_LDS();
    case 0x9001: return AVR_Opcode_Cycles_LD_Z_Indirect_Postinc();
    case 0x9002: return AVR_Opcode_Cycles_LD_Z_Indirect_Predec();
    case 0x9004: return AVR_Opcode_Cycles_LPM_Z();
    case 0x9005: return AVR_Opcode_Cycles_LPM_Z_Postinc();
    case 0x9006: return AVR_Opcode_Cycles_ELPM_Z();
    case 0x9007: return AVR_Opcode_Cycles_ELPM_Z_Postinc();
    case 0x9009: return AVR_Opcode_Cycles_LD_Y_Indirect_Postinc();
    case 0x900A: return AVR_Opcode_Cycles_LD_Y_Indirect_Predec();
    case 0x900C: return AVR_Opcode_Cycles_LD_X_Indirect();
    case 0x900D: return AVR_Opcode_Cycles_LD_X_Indirect_Postinc();
    case 0x900E: return AVR_Opcode_Cycles_LD_X_Indirect_Predec();
    case 0x900F: return AVR_Opcode_Cycles_POP();

    case 0x9200: return AVR_Opcode_Cycles_STS();
    case 0x9201: return AVR_Opcode_Cycles_ST_Z_Indirect_Postinc();
    case 0x9202: return AVR_Opcode_Cycles_ST_Z_Indirect_Predec();
    case 0x9204: return AVR_Opcode_Cycles_XCH();
    case 0x9205: return AVR_Opcode_Cycles_LAS();
    case 0x9206: return AVR_Opcode_Cycles_LAC();
    case 0x9207: return AVR_Opcode_Cycles_LAT();
    case 0x9209: return AVR_Opcode_Cycles_ST_Y_Indirect_Postinc();
    case 0x920A: return AVR_Opcode_Cycles_ST_Y_Indirect_Predec();
    case 0x920C: return AVR_Opcode_Cycles_ST_X_Indirect();
    case 0x920D: return AVR_Opcode_Cycles_ST_X_Indirect_Postinc();
    case 0x920E: return AVR_Opcode_Cycles_ST_X_Indirect_Predec();
    case 0x920F: return AVR_Opcode_Cycles_PUSH();

    // -- One-operand instructions
    case 0x9400: return AVR_Opcode_Cycles_COM();
    case 0x9401: return AVR_Opcode_Cycles_NEG();
    case 0x9402: return AVR_Opcode_Cycles_SWAP();
    case 0x9403: return AVR_Opcode_Cycles_INC();
    case 0x9405: return AVR_Opcode_Cycles_ASR();
    case 0x9406: return AVR_Opcode_Cycles_LSR();
    case 0x9407: return AVR_Opcode_Cycles_ROR();
    case 0x940A: return AVR_Opcode_Cycles_DEC();

    }
    switch (OP_ & 0xFE0E)
    {
    case 0x940C: return AVR_Opcode_Cycles_JMP();
    case 0x940E: return AVR_Opcode_Cycles_CALL();
    }

    switch (OP_ & 0xFE08)
    {

    // -- BLD/BST Encoding
    case 0xF800: return AVR_Opcode_Cycles_BLD();
    case 0xFA00: return AVR_Opcode_Cycles_BST();
```

```
    // -- SBRC/SBRS Encoding
    case 0xFC00: return AVR_Opcode_Cycles_SBRC();
    case 0xFE00: return AVR_Opcode_Cycles_SBRS();
    }

    switch (OP_ & 0xFC07)
    {
    // -- Conditional branches
    case 0xF000: return AVR_Opcode_Cycles_BRCS();
    // case 0xF000: return AVR_Opcode_Cycles_BRLO();            // AKA AVR_Opcode_Cycles_BRCS();
    case 0xF001: return AVR_Opcode_Cycles_BREQ();
    case 0xF002: return AVR_Opcode_Cycles_BRMI();
    case 0xF003: return AVR_Opcode_Cycles_BRVS();
    case 0xF004: return AVR_Opcode_Cycles_BRLT();
    case 0xF006: return AVR_Opcode_Cycles_BRTS();
    case 0xF007: return AVR_Opcode_Cycles_BRIE();
    case 0xF400: return AVR_Opcode_Cycles_BRCC();
    // case 0xF400: return AVR_Opcode_Cycles_BRSH();            // AKA AVR_Opcode_Cycles_BRCC();
    case 0xF401: return AVR_Opcode_Cycles_BRNE();
    case 0xF402: return AVR_Opcode_Cycles_BRPL();
    case 0xF403: return AVR_Opcode_Cycles_BRVC();
    case 0xF404: return AVR_Opcode_Cycles_BRGE();
    case 0xF405: return AVR_Opcode_Cycles_BRHC();
    case 0xF406: return AVR_Opcode_Cycles_BRTC();
    case 0xF407: return AVR_Opcode_Cycles_BRID();
    }

    switch (OP_ & 0xFC00)
    {
    // -- 4-bit register pair
    case 0x0200: return AVR_Opcode_Cycles_MULS();

    // -- 5-bit register pairs --
    case 0x0400: return AVR_Opcode_Cycles_CPC();
    case 0x0800: return AVR_Opcode_Cycles_SBC();
    case 0x0C00: return AVR_Opcode_Cycles_ADD();
    // case 0x0C00: return AVR_Opcode_Cycles_LSL(); (!! Implemented with: " add rd, rd"
    case 0x1000: return AVR_Opcode_Cycles_CPSE();
    case 0x1300: return AVR_Opcode_Cycles_ROL();
    case 0x1400: return AVR_Opcode_Cycles_CP();
    case 0x1C00: return AVR_Opcode_Cycles_ADC();
    case 0x1800: return AVR_Opcode_Cycles_SUB();
    case 0x2000: return AVR_Opcode_Cycles_AND();
    // case 0x2000: return AVR_Opcode_Cycles_TST(); (!! Implemented with: " and rd, rd"
    case 0x2400: return AVR_Opcode_Cycles_EOR();
    case 0x2C00: return AVR_Opcode_Cycles_MOV();
    case 0x2800: return AVR_Opcode_Cycles_OR();

    // -- 5-bit register pairs -- Destination = R1:R0
    case 0x9C00: return AVR_Opcode_Cycles_MUL();
    }

    switch (OP_ & 0xF800)
    {
    case  0xB800: return AVR_Opcode_Cycles_OUT();
    case  0xB000: return AVR_Opcode_Cycles_IN();
    }

    switch (OP_ & 0xF000)
    {
    // -- Register immediate --
    case 0x3000: return AVR_Opcode_Cycles_CPI();
    case 0x4000: return AVR_Opcode_Cycles_SBCI();
    case 0x5000: return AVR_Opcode_Cycles_SUBI();
    case 0x6000: return AVR_Opcode_Cycles_ORI();
    case 0x7000: return AVR_Opcode_Cycles_ANDI();

    //-- 12-bit immediate
    case 0xC000: return AVR_Opcode_Cycles_RJMP();
    case 0xD000: return AVR_Opcode_Cycles_RCALL();

    // -- Register immediate
    case 0xE000: return AVR_Opcode_Cycles_LDI();
    }

    switch (OP_ & 0xD208)
    {
    // -- 7-bit signed offset
    case 0x8000: return AVR_Opcode_Cycles_LDD_Z();
    case 0x8008: return AVR_Opcode_Cycles_LDD_Y();
    case 0x8200: return AVR_Opcode_Cycles_STD_Z();
    case 0x8208: return AVR_Opcode_Cycles_STD_Y();
    }

    return AVR_Opcode_Cycles_Unimplemented();
}

```

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

```
/****************************************************************************
 *     (      (                       (      |
 *    )\ )   )\ )     (              )\ )    |
 *   (()/(  (()/(    )\      (    (  (()/(    | -- [ Funkenstein ] -------------
 *    /(_))  /(_))((((_)()\   )\  /(_))      | -- [ Litle ] -------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ---------------------
 *   | |_   | |    (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *   | __|  | |__   / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
 *   |_|    |____| /_/ \_\   \_/   |_|_\    |
 *                                          | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __AVR_OP_CYCLES_H__
#define __AVR_OP_CYCLES_H__

#include <stdint.h>

//---------------------------------------------------------------------------
uint8_t AVR_Opcode_Cycles( uint16_t OP_ );

#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

!MOS Verified

Definition at line 251 of file avr_op_decode.c.

4.32 avr_op_decode.c

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

```
 *     (      (                      (      |
 *    )\ )   )\ )     (                )\ )   |
 *   (()/(  (()/(    )\      (    (   (()/(   | -- [ Funkenstein ] -------------
 *    /(_))  /(_))((((_)()\   )\   /(_))      | -- [ Litle ] -------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))       | -- [ AVR ] ---------------------
 *   | |_   | |    (_)_\(_)\ \ / / | _ \      | -- [ Virtual ] -----------------
 *   | __|  | |__   / _ \   \ V /  |   /      | -- [ Runtime ] -----------------
 *   |_|    |____| /_/ \_\   \_/   |_|_\      |
 *                                            | "Yeah, it does Arduino..."
 * -------------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#include <stdint.h>

#include "emu_config.h"

#include "avr_op_decode.h"

//---------------------------------------------------------------------------
static void AVR_Decoder_NOP( uint16_t OP_)
{
    // Nothing to do here...
}
//---------------------------------------------------------------------------
static void AVR_Decoder_Register_Pair_4bit( uint16_t OP_)
{
    uint8_t Rr = (OP_ & 0x000F);
    uint8_t Rd = ((OP_ & 0x00F0) >> 4);

    stCPU.Rr16 = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r_word[Rr]);
    stCPU.Rd16 = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r_word[Rd]);
}
//---------------------------------------------------------------------------
static void AVR_Decoder_Register_Pair_3bit( uint16_t OP_)
{
    uint8_t Rr = (OP_ & 0x0007) + 16;
    uint8_t Rd = ((OP_ & 0x0070) >> 4) + 16;

    stCPU.Rr = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rr]);
    stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
}
//---------------------------------------------------------------------------
static void AVR_Decoder_Register_Pair_5bit( uint16_t OP_)
{
    uint8_t Rr = (OP_ & 0x000F) | ((OP_ & 0x0200) >> 5);
    uint8_t Rd = (OP_ & 0x01F0) >> 4;

    stCPU.Rr = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rr]);
    stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
}
//---------------------------------------------------------------------------
static void AVR_Decoder_Register_Immediate( uint16_t OP_)
{
    uint8_t K = (OP_ & 0x000F) | ((OP_ & 0x0F00) >> 4);
    uint8_t Rd = ((OP_ & 0x00F0) >> 4) + 16;

    stCPU.K = K;
    stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
}
//---------------------------------------------------------------------------
static void AVR_Decoder_LDST_YZ_k( uint16_t OP_)
{
    uint8_t q = (OP_ & 0x0007) |             // Awkward encoding... see manual for details.
                ((OP_ & 0x0C00) >> (7)) |
                ((OP_ & 0x2000) >> (8));

    uint8_t Rd = (OP_ & 0x01F0) >> 4;

    stCPU.q = q;
    stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
}
//---------------------------------------------------------------------------
static void AVR_Decoder_LDST( uint16_t OP_)
{
    uint8_t Rd = (OP_ & 0x01F0) >> 4;

    stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
}
//---------------------------------------------------------------------------
static void AVR_Decoder_LDS_STS( uint16_t OP_)
{
    uint8_t Rd = (OP_ & 0x01F0) >> 4;

    stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
    stCPU.K = stCPU.pu16ROM[ stCPU.u16PC + 1 ];
}
```

```
//---------------------------------------------------------------------------
static void AVR_Decoder_Register_Single( uint16_t OP_)
{
    uint8_t Rd = (OP_ & 0x01F0) >> 4;

    stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
}
//---------------------------------------------------------------------------
static void AVR_Decoder_Register_SC( uint16_t OP_)
{
    uint8_t b = (OP_ & 0x0070) >> 4;

    stCPU.b = b;
}
//---------------------------------------------------------------------------
static void AVR_Decoder_Misc( uint16_t OP_)
{
    // Nothing to do here.
}
//---------------------------------------------------------------------------
static void AVR_Decoder_Indirect_Jump( uint16_t OP_)
{
    // Nothing to do here.
}
//---------------------------------------------------------------------------
static void AVR_Decoder_DEC_Rd( uint16_t OP_)
{
    uint8_t Rd = (OP_ & 0x01F0) >> 4;

    stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
}
//---------------------------------------------------------------------------
static void AVR_Decoder_DES_round_4( uint16_t OP_)
{
    uint8_t K = (OP_ & 0x00F0) >> 4;
    stCPU.K = K;
}
//---------------------------------------------------------------------------
static void AVR_Decoder_JMP_CALL_22( uint16_t OP_)
{
    uint16_t op = stCPU.pu16ROM[ stCPU.u16PC + 1 ];
    uint32_t k = op;
    k |= (((OP_ & 0x0001) | (OP_ & 0x01F0) >> 3) << 16);

    stCPU.k = k;

    // These are 2-cycle instructions.  Clock the CPU here, since we're fetching
    // the second word of data for this opcode here.
    IO_Clock();
}
//---------------------------------------------------------------------------
static void AVR_Decoder_ADIW_SBIW_6( uint16_t OP_)
{
    uint8_t K = (OP_ & 0x000F) | ((OP_ & 0x00C0) >> 2);
    uint8_t Rd16 = (((OP_ & 0x0030) >> 4) * 2) + 24;

    stCPU.K = K;
    stCPU.Rd16 = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r_word[Rd16 >> 1]);
}
//---------------------------------------------------------------------------
static void AVR_Decoder_IO_Bit( uint16_t OP_)
{
    uint8_t b = (OP_ & 0x0007);
    uint8_t A = (OP_ & 0x00F8) >> 3;

    stCPU.b = b;
    stCPU.A = A;
}
//---------------------------------------------------------------------------
static void AVR_Decoder_MUL( uint16_t OP_)
{
    uint8_t Rr = (OP_ & 0x000F) | ((OP_ & 0x0200) >> 5);
    uint8_t Rd = (OP_ & 0x01F0) >> 4;

    stCPU.Rr = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rr]);
    stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
}
//---------------------------------------------------------------------------
static void AVR_Decoder_IO_In_Out( uint16_t OP_)
{
    uint8_t A = (OP_ & 0x000F) | ((OP_ & 0x0600) >> 5);
    uint8_t Rd = (OP_ & 0x01F0) >> 4;

    stCPU.A = A;
    stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
}
//---------------------------------------------------------------------------
```

```
static void AVR_Decoder_Relative_Jump( uint16_t OP_)
{
    // NB: -2K <= k <= 2K
    uint16_t k = (OP_ & 0x0FFF);

    // Check for sign bit in 12-bit value...
    if (k & 0x0800)
    {
        stCPU.k_s = (int32_t)((~k & 0x07FF) + 1) * -1;
    }
    else
    {
        stCPU.k_s = (int32_t)k;
    }
}
//---------------------------------------------------------------------------
static void AVR_Decoder_LDI( uint16_t OP_)
{
    uint8_t K = (OP_ & 0x000F) | ((OP_ & 0x0F00) >> 4);
    uint8_t Rd = ((OP_ & 0x00F0) >> 4) + 16;

    stCPU.K = K;
    stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
}
//---------------------------------------------------------------------------
static void AVR_Decoder_Conditional_Branch( uint16_t OP_)
{
    // NB: -64 <= k <= 63
    uint8_t b = (OP_ & 0x0007);
    uint8_t k = ((OP_ & 0x03F8) >> 3);

    stCPU.b = b;

    // Check for sign bit in 7-bit value...
    if (k & 0x40)
    {
        // Convert to signed 32-bit integer... probably a cleaner way
        // of doing this, but I'm tired.
        stCPU.k_s = (int32_t)((~k & 0x3F) + 1) * -1;
    }
    else
    {
        stCPU.k_s = (int32_t)k;
    }
}
//---------------------------------------------------------------------------
static void AVR_Decoder_BLD_BST( uint16_t OP_)
{
    uint8_t b = (OP_ & 0x0007);
    uint8_t Rd = ((OP_ & 0x01F0) >> 4);

    stCPU.b = b;
    stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
}

//---------------------------------------------------------------------------
static void AVR_Decoder_SBRC_SBRS( uint16_t OP_)
{
    uint8_t b = (OP_ & 0x0007);
    uint8_t Rd = ((OP_ & 0x01F0) >> 4);

    stCPU.b = b;
    stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
}

//---------------------------------------------------------------------------
AVR_Decoder AVR_Decoder_Function( uint16_t OP_ )
{
    if (( OP_ & 0xFF0F) == 0x9408 )
    {
        // SEx/CLx status register clear/set bit.
        return AVR_Decoder_Register_SC;
    }
    else if (( OP_ & 0xFF0F) == 0x9508 )
    {
        // Miscellaneous instruction
        return AVR_Decoder_Misc;
    }
    else if (( OP_ & 0xFF0F) == 0x940B )
    {
        // Des round k
        return AVR_Decoder_DES_round_4;
    }
    else if ( (( OP_ & 0xFF00 ) == 0x0100 ) ||
              (( OP_ & 0xFF00 ) == 0x0200 ) )
    {
        // Register pair 4bit (MOVW, MULS)
```

```
        return  AVR_Decoder_Register_Pair_4bit;
    }
    else if (( OP_ & 0xFF00 ) == 0x0300 )
    {
        // 3-bit register pair (R16->R23) - (FMUL, FMULS, FMULSU, MULSU)
        return AVR_Decoder_Register_Pair_3bit;
    }
    else if (( OP_ & 0xFF00 ) <= 0x2F00 )
    {
        // Register pair 5bit
        return AVR_Decoder_Register_Pair_5bit;
    }
    else if (( OP_ & 0xFF00) <= 0x7F00 )
    {
        // Register immediate
        return AVR_Decoder_Register_Immediate;
    }
    else if (( OP_ & 0xFEEF) == 0x9409 )
    {
        // Indirect Jump/call
        return AVR_Decoder_Indirect_Jump;
    }
    else if (( OP_ & 0xFE08) == 0x9400 )
    {
        // 1-operand instructions.
        return AVR_Decoder_Register_Single;
    }
    else if (( OP_ & 0xFE0F) == 0x940A )
    {
        // Dec Rd
        return AVR_Decoder_DEC_Rd;
    }
    else if (( OP_ & 0xFE0C) == 0x940C )
    {
        // Jmp/call abs22
        return AVR_Decoder_JMP_CALL_22;
    }
    else if (( OP_ & 0xFE00) == 0x9600 )
    {
        // ADIW/SBIW Rp
        return AVR_Decoder_ADIW_SBIW_6;
    }
    else if (( OP_ & 0xFC0F) == 0x9000 )
    {
        // LDS/STS
        return AVR_Decoder_LDS_STS;
    }
    else if (( OP_ & 0xFC00) == 0x9000 )
    {
        // LD/ST other
        return AVR_Decoder_LDST;
    }
    else if (( OP_ & 0xFC00) == 0x9800 )
    {
        // IO Space bit operations
        return AVR_Decoder_IO_Bit;
    }
    else if (( OP_ & 0xFC00) == 0x9C00 )
    {
        // MUL unsigned R1:R0 = Rr x Rd
        return AVR_Decoder_MUL;
    }
    else if (( OP_ & 0xFC00) == 0xF800 )
    {
        // BLD/BST register bit to STATUS.T
        return AVR_Decoder_BLD_BST;
    }
    else if (( OP_ & 0xFC00) == 0xFC00 )
    {
        // SBRC/SBRS
        return AVR_Decoder_SBRC_SBRS;
    }
    else if (( OP_ & 0xF800) == 0xF000 )
    {
        // Conditional branch
        return AVR_Decoder_Conditional_Branch;
    }
    else if (( OP_ & 0xF000) == 0xE000 )
    {
        // LDI Rh, K
        return AVR_Decoder_LDI;
    }
    else if (( OP_ & 0xF000) == 0xB000 )
    {
        // IO space IN/OUT operations
        return AVR_Decoder_IO_In_Out;
    }
```

```
    else if (( OP_ & 0xE000) == 0xC000 )
    {
        // RElative Jump/Call
        return AVR_Decoder_Relative_Jump;
    }
    else if (( OP_ & 0xD000) == 0x8000 )
    {
        // LDD/STD to Z+kY+k
        return AVR_Decoder_LDST_YZ_k;
    }
    else if ( OP_ == 0 )
    {
        return AVR_Decoder_NOP;
    }
    return AVR_Decoder_NOP;
}

//---------------------------------------------------------------------------
void AVR_Decode( uint16_t OP_ )
{
    AVR_Decoder myDecoder;
    myDecoder = AVR_Decoder_Function(OP_);
    myDecoder( OP_);
}
```

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

!MOS Verified

Definition at line 251 of file avr_op_decode.c.

4.34 avr_op_decode.h

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

```
 *     (      (                        (     |
 *    )\ )   )\ )     (               )\ )   |
 *   (()/(  (()/(    )\      (    (   (()/(   | -- [ Funkenstein ] -------------
 *    /(_))  /(_))((((_)()\   )\   /(_))      | -- [ Litle ] -------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ---------------------
 *   | |_   | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *   | __|  | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
 *   |_|    |____|  /_/ \_\   \_/   |_|_\    |
 *                                           | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __AVR_OP_DECODE_H__
#define __AVR_OP_DECODE_H__

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

//---------------------------------------------------------------------------
// Format decoder function jump table
typedef void (*AVR_Decoder)( uint16_t OP_);

//---------------------------------------------------------------------------
AVR_Decoder AVR_Decoder_Function( uint16_t OP_ );

//---------------------------------------------------------------------------
void AVR_Decode( uint16_t OP_ );

#endif

```

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 2014-15, 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 {
```

```
    return 1;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_Register_Immediate( uint16_t OP_)
{
    return 1;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_LDST_YZ_k( uint16_t OP_)
{
    return 1;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_LDST( uint16_t OP_)
{
    return 1;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_LDS_STS( uint16_t OP_)
{
    return 2;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_Register_Single( uint16_t OP_)
{
    return 1;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_Register_SC( uint16_t OP_)
{
    return 1;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_Misc( uint16_t OP_)
{
    return 1;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_Indirect_Jump( uint16_t OP_)
{
    return 1;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_DEC_Rd( uint16_t OP_)
{
    return 1;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_DES_round_4( uint16_t OP_)
{
    return 1;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_JMP_CALL_22( uint16_t OP_)
{
    return 2;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_ADIW_SBIW_6( uint16_t OP_)
{
    return 1;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_IO_Bit( uint16_t OP_)
{
    return 1;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_MUL( uint16_t OP_)
{
    return 1;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_IO_In_Out( uint16_t OP_)
{
    return 1;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_Relative_Jump( uint16_t OP_)
{
    return 1;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_LDI( uint16_t OP_)
{
    return 1;
}
```

```
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_Conditional_Branch( uint16_t OP_)
{
    return 1;
}
//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_BLD_BST( uint16_t OP_)
{
    return 1;
}

//---------------------------------------------------------------------------
static uint8_t AVR_Opcode_Size_SBRC_SBRS( uint16_t OP_)
{
    return 1;
}

//---------------------------------------------------------------------------
uint8_t AVR_Opcode_Size( uint16_t OP_ )
{
    if (( OP_ & 0xFF0F) == 0x9408 )
    {
        // SEx/CLx status register clear/set bit.
        return AVR_Opcode_Size_Register_SC( OP_ );
    }
    else if (( OP_ & 0xFF0F) == 0x9508 )
    {
        // Miscellaneous instruction
        return AVR_Opcode_Size_Misc( OP_ );
    }
    else if (( OP_ & 0xFF0F) == 0x940B )
    {
        // Des round k
        return AVR_Opcode_Size_DES_round_4( OP_ );
    }
    else if ( (( OP_ & 0xFF00 ) == 0x0100 ) ||
              (( OP_ & 0xFF00 ) == 0x0200 ) )
    {
        // Register pair 4bit (MOVW, MULS)
         return AVR_Opcode_Size_Register_Pair_4bit( OP_ );
    }
    else if (( OP_ & 0xFF00 ) == 0x0300 )
    {
        // 3-bit register pair (R16->R23) - (FMUL, FMULS, FMULSU, MULSU)
        return AVR_Opcode_Size_Register_Pair_3bit( OP_ );
    }
    else if (( OP_ & 0xFF00 ) <= 0x4F00 )
    {
        // Register pair 5bit
        return AVR_Opcode_Size_Register_Pair_5bit( OP_ );
    }
    else if (( OP_ & 0xFF00) <= 0x7F00 )
    {
        // Register immediate
        return AVR_Opcode_Size_Register_Immediate( OP_ );
    }
    else if (( OP_ & 0xFEEF) == 0x9409 )
    {
        // Indirect Jump/call
        return AVR_Opcode_Size_Indirect_Jump( OP_ );
    }
    else if (( OP_ & 0xFE08) == 0x9400 )
    {
        // 1-operand instructions.
        return AVR_Opcode_Size_Register_Single( OP_ );
    }
    else if (( OP_ & 0xFE0F) == 0x940A )
    {
        // Dec Rd
        return AVR_Opcode_Size_DEC_Rd( OP_ );
    }
    else if (( OP_ & 0xFE0C) == 0x940C )
    {
        // Jmp/call abs22
        return AVR_Opcode_Size_JMP_CALL_22( OP_ );
    }
    else if (( OP_ & 0xFE00) == 0x9600 )
    {
        // ADIW/SBIW Rp
        return AVR_Opcode_Size_ADIW_SBIW_6( OP_ );
    }
    else if (( OP_ & 0xFC0F) == 0x9000 )
    {
        // LDS/STS
        return AVR_Opcode_Size_LDS_STS( OP_ );
    }
    else if (( OP_ & 0xFC00) == 0x9000 )
```

```
    {
        // LD/ST other
        return AVR_Opcode_Size_LDST( OP_ );
    }
    else if (( OP_ & 0xFC00) == 0x9800 )
    {
        // IO Space bit operations
        return AVR_Opcode_Size_IO_Bit( OP_ );
    }
    else if (( OP_ & 0xFC00) == 0x9C00 )
    {
        // MUL unsigned R1:R0 = Rr x Rd
        return AVR_Opcode_Size_MUL( OP_ );
    }
    else if (( OP_ & 0xFC00) == 0xF800 )
    {
        // BLD/BST register bit to STATUS.T
        return AVR_Opcode_Size_BLD_BST( OP_ );
    }
    else if (( OP_ & 0xFC00) == 0xFC00 )
    {
        // SBRC/SBRS
        return AVR_Opcode_Size_SBRC_SBRS( OP_ );
    }
    else if (( OP_ & 0xF800) == 0xF000 )
    {
        // Conditional branch
        return AVR_Opcode_Size_Conditional_Branch( OP_ );
    }
    else if (( OP_ & 0xF000) == 0xE000 )
    {
        // LDI Rh, K
        return AVR_Opcode_Size_LDI( OP_ );
    }
    else if (( OP_ & 0xF000) == 0xB000 )
    {
        // IO space IN/OUT operations
        return AVR_Opcode_Size_IO_In_Out( OP_ );
    }
    else if (( OP_ & 0xE000) == 0xC000 )
    {
        // RElative Jump/Call
        return AVR_Opcode_Size_Relative_Jump( OP_ );
    }
    else if (( OP_ & 0xD000) == 0x8000 )
    {
        // LDD/STD to Z+kY+k
        return AVR_Opcode_Size_LDST_YZ_k( OP_ );
    }
    else if ( OP_ == 0 )
    {
        return AVR_Opcode_Size_NOP( OP_ );
    }
    return AVR_Opcode_Size_NOP( OP_ );
}
```

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

```
/****************************************************************************
 *     (      (                        (      |
 *    )\ )  )\ )     (               )\ )    |
 *   (()/( (()/(    )\      (   (   (()/(    | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))       | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \     | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /     | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\     |
 *                                           | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __AVR_OP_SIZE__
#define __AVR_OP_SIZE__

#include <stdint.h>

//---------------------------------------------------------------------------
uint8_t AVR_Opcode_Size( uint16_t OP_ );

#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"
#include "interactive.h"
#include "write_callout.h"
#include "interrupt_callout.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 749 of file avr_opcodes.c.

4.39.2.2 static void AVR_Opcode_EICALL (void) [static]

! ToDo - Implement EIND calling!

Definition at line 858 of file avr_opcodes.c.

4.39.2.3 static void AVR_Opcode_EIJMP (void) [static]

ToDo - implement EIND instructions

Definition at line 793 of file avr_opcodes.c.

4.39.2.4 static void AVR_Opcode_ELPM (void) [static]

! ToDo - Add in RAMPZ register.

Definition at line 1484 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 1856 of file avr_opcodes.c.

4.39.2.6 static void AVR_Opcode_SPM (void) [static]

! Implment later...

Definition at line 1535 of file avr_opcodes.c.

4.39.2.7 static void AVR_Opcode_SPM_Z_Postinc2 (void) [static]

! Implement later...

Definition at line 1541 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 2057 of file avr_opcodes.c.

4.40 avr_opcodes.c

```
/***************************************************************************
 *     (      (                                 (      |
 *   )\ )  )\ )     (                    )\ )    |
 *  (()/( (()/(     )\        (     (   (()/(    | -- [ Funkenstein ] -------------
 *   /(_)) /(_))((((_)()\   )\   /(_))      | -- [ Litle ] -------------------
 *  (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ---------------------
 *  | |_  | |     (_)_\(_)\ \ / / | _ \     | -- [ Virtual ] -----------------
 *  | __| | |__    / _ \   \ V /  |   /     | -- [ Runtime ] -----------------
 *  |_|   |____| /_/ \_\   \_/   |_|_\     |
 *                                          | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

#include "avr_cpu_print.h"
#include "emu_config.h"
#include "avr_opcodes.h"
#include "interactive.h"
#include "write_callout.h"
#include "interrupt_callout.h"

//---------------------------------------------------------------------------
#define DEBUG_PRINT(...)

//---------------------------------------------------------------------------
static void AVR_Abort(void)
{
    print_core_regs();
    exit(-1);
}

//---------------------------------------------------------------------------
static void Data_Write( uint16_t u16Addr_, uint8_t u8Val_ )
{
    // Writing to RAM can be a tricky deal, because the address space is shared
    // between RAM, the core registers, and a bunch of peripheral I/O registers.
    DEBUG_PRINT("Write: 0x%04X=%02X\n", u16Addr_, u8Val_ );
    if (!WriteCallout_Run( u16Addr_, u8Val_ ))
    {
        return;
    }

    // Check to see if the write operation falls within the peripheral I/O range
    if (u16Addr_ >= 32 && u16Addr_ <= 255)
    {
        // I/O range - check to see if there's a peripheral installed at this address
        IOWriterList *pstIOWrite = stCPU.apstPeriphWriteTable[ u16Addr_ ];

        // If there is a peripheral or peripherals
        if (pstIOWrite)
        {
            // Iterate through the list of installed peripherals at this address, and
```

```
            // call their write handler
            while (pstIOWrite)
            {
                pstIOWrite->pfWriter( pstIOWrite->pvContext, (uint8_t)u16Addr_, u8Val_ );
                pstIOWrite = pstIOWrite->next;
            }
        }
        // Otherwise, there is no peripheral -- just assume we can treat this as normal RAM.
        else
        {
            stCPU.pstRAM->au8RAM[ u16Addr_ ] = u8Val_;
        }
    }
    else if (u16Addr_ >= (stCPU.u32RAMSize + 256))
    {
        fprintf( stderr, "[Write Abort] RAM Address 0x%04X is out of range!\n", u16Addr_ );
        AVR_Abort();
    }
    // RAM address range - direct write-through.
    else
    {
        stCPU.pstRAM->au8RAM[ u16Addr_ ] = u8Val_;
    }

}

//---------------------------------------------------------------------------
static uint8_t Data_Read( uint16_t u16Addr_)
{
    // Writing to RAM can be a tricky deal, because the address space is shared
    // between RAM, the core registers, and a bunch of peripheral I/O registers.

    // Check to see if the write operation falls within the peripheral I/O range
    DEBUG_PRINT( "Data Read: %04X\n", u16Addr_ );
    if (u16Addr_ >= 32 && u16Addr_ <= 255)
    {
        // I/O range - check to see if there's a peripheral installed at this address
        IOReaderList *pstIORead = stCPU.apstPeriphReadTable[ u16Addr_ ];
        DEBUG_PRINT( "Peripheral Read: 0x%04X\n", u16Addr_ );
        // If there is a peripheral or peripherals
        if (pstIORead)
        {
            DEBUG_PRINT(" Found peripheral\n");
            // Iterate through the list of installed peripherals at this address, and
            // call their read handler
            uint8_t u8Val;
            while (pstIORead)
            {
                pstIORead->pfReader( pstIORead->pvContext,  (uint8_t)u16Addr_, &u8Val);
                pstIORead = pstIORead->next;
            }
            return u8Val;
        }
        // Otherwise, there is no peripheral -- just assume we can treat this as normal RAM.
        else
        {
            DEBUG_PRINT(" No peripheral\n");
            return stCPU.pstRAM->au8RAM[ u16Addr_ ];
        }
    }
    else if (u16Addr_ >= (stCPU.u32RAMSize + 256))
    {
        fprintf( stderr, "[Read Abort] RAM Address 0x%04X is out of range!\n", u16Addr_ );
        AVR_Abort();
    }
    // RAM address range - direct read
    else
    {
        return stCPU.pstRAM->au8RAM[ u16Addr_ ];
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_NOP( void )
{
    // Nop - do nothing.
}

//---------------------------------------------------------------------------
inline void ADD_Half_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
{
    stCPU.pstRAM->stRegisters.SREG.H =
            ( ((Rd_ & Rr_)      & 0x08 )
            | ((Rr_ & (~Result_)) & 0x08 )
            | (((~Result_) & Rd_) & 0x08) ) != false;
}

```

```
//---------------------------------------------------------------------------
inline void ADD_Full_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
{
    stCPU.pstRAM->stRegisters.SREG.C =
            ( ((Rd_ & Rr_)     & 0x80 )
            | ((Rr_ & (~Result_)) & 0x80 )
            | (((~Result_) & Rd_) & 0x80) ) != false;
}

//---------------------------------------------------------------------------
inline void ADD_Overflow_Flag( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
{
    stCPU.pstRAM->stRegisters.SREG.V =
             ( ((Rd_ & Rr_ & ~Result_)  & 0x80 )
             | ((~Rd_ & ~Rr_ & Result_) & 0x80 ) ) != 0;
}

//---------------------------------------------------------------------------
inline void Signed_Flag( void )
{
    unsigned int N = stCPU.pstRAM->stRegisters.SREG.N;
    unsigned int V = stCPU.pstRAM->stRegisters.SREG.V;

    stCPU.pstRAM->stRegisters.SREG.S = N ^ V;
}

//---------------------------------------------------------------------------
inline void R8_Zero_Flag( uint8_t R_ )
{
    stCPU.pstRAM->stRegisters.SREG.Z = (R_ == 0);
}

//---------------------------------------------------------------------------
inline void R8_CPC_Zero_Flag( uint8_t R_ )
{
    stCPU.pstRAM->stRegisters.SREG.Z = (stCPU.pstRAM->stRegisters.SREG.Z && (R_ == 0));
}

//---------------------------------------------------------------------------
inline void R8_Negative_Flag( uint8_t R_ )
{
    stCPU.pstRAM->stRegisters.SREG.N = ((R_ & 0x80) == 0x80);
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ADD( void )
{
    uint8_t u8Result;
    uint8_t u8Rd = *(stCPU.Rd);
    uint8_t u8Rr = *(stCPU.Rr);

    u8Result = u8Rd + u8Rr;
    *(stCPU.Rd) = u8Result;

// ---- Update flags ----
    ADD_Half_Carry(  u8Rd, u8Rr, u8Result );
    ADD_Full_Carry(  u8Rd, u8Rr, u8Result );
    ADD_Overflow_Flag(  u8Rd, u8Rr, u8Result );
    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ADC( void )
{
    uint8_t u8Result;
    uint8_t u8Rd = *(stCPU.Rd);
    uint8_t u8Rr = *(stCPU.Rr);
    uint8_t u8Carry = (stCPU.pstRAM->stRegisters.SREG.C);

    u8Result = u8Rd + u8Rr + u8Carry;
    *(stCPU.Rd) = u8Result;

// ---- Update flags ----
    ADD_Half_Carry(  u8Rd, u8Rr, u8Result );
    ADD_Full_Carry(  u8Rd, u8Rr, u8Result );
    ADD_Overflow_Flag(  u8Rd, u8Rr, u8Result );
    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Signed_Flag();
}

//---------------------------------------------------------------------------
inline void R16_Negative_Flag( uint16_t Result_ )
{
    stCPU.pstRAM->stRegisters.SREG.N =
```

```
            ((Result_ & 0x8000) != 0);
}

//---------------------------------------------------------------------------
inline void R16_Zero_Flag( uint16_t Result_ )
{
    stCPU.pstRAM->stRegisters.SREG.Z =
            (Result_ == 0);
}

//---------------------------------------------------------------------------
inline void ADIW_Overflow_Flag( uint16_t Rd_, uint16_t Result_ )
{
    stCPU.pstRAM->stRegisters.SREG.V =
            (((Rd_ & 0x8000) == 0) && ((Result_ & 0x8000) == 0x8000));
}

//---------------------------------------------------------------------------
inline void ADIW_Carry_Flag( uint16_t Rd_, uint16_t Result_ )
{
    stCPU.pstRAM->stRegisters.SREG.C =
            (((Rd_ & 0x8000) == 0x8000) && ((Result_ & 0x8000) == 0));
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ADIW( void )
{
    uint16_t u16K = (stCPU.K);
    uint16_t u16Rd = *(stCPU.Rd16);
    uint16_t u16Result;

    u16Result = u16Rd + u16K;
    *(stCPU.Rd16) = u16Result;

// ---- Update Flags ----
    ADIW_Carry_Flag(  u16Rd, u16Result);
    ADIW_Overflow_Flag(  u16Rd, u16Result );
    R16_Negative_Flag(  u16Result );
    R16_Zero_Flag(  u16Result );
    Signed_Flag();
}

//---------------------------------------------------------------------------
inline void SUB_Overflow_Flag( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
{
    stCPU.pstRAM->stRegisters.SREG.V =
             ( ((Rd_ & ~Rr_ & ~Result_) & 0x80 )
             | ((~Rd_ & Rr_ & Result_) & 0x80 ) ) != 0;
}
//---------------------------------------------------------------------------
inline void SUB_Half_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
{
    stCPU.pstRAM->stRegisters.SREG.H =
             ( ((~Rd_ & Rr_) & 0x08 )
             | ((Rr_ & Result_) & 0x08 )
             | ((Result_ & ~Rd_) & 0x08 ) ) == 0x08;
}
//---------------------------------------------------------------------------
inline void SUB_Full_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
{
    stCPU.pstRAM->stRegisters.SREG.C =
             ( ((~Rd_ & Rr_) & 0x80 )
             | ((Rr_ & Result_) & 0x80 )
             | ((Result_ & ~Rd_) & 0x80 ) ) == 0x80;
}

//---------------------------------------------------------------------------
static void AVR_Opcode_SUB( void )
{
    uint8_t u8Rd = *stCPU.Rd;
    uint8_t u8Rr = *stCPU.Rr;
    uint8_t u8Result = u8Rd - u8Rr;

    *stCPU.Rd = u8Result;

    //--Flags
    SUB_Half_Carry( u8Rd, u8Rr, u8Result);
    SUB_Full_Carry( u8Rd, u8Rr, u8Result);
    SUB_Overflow_Flag( u8Rd, u8Rr, u8Result);
    R8_Negative_Flag( u8Result);
    R8_Zero_Flag( u8Result);
    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_SUBI( void )
{
```

```
    uint8_t u8Rd = *stCPU.Rd;
    uint8_t u8K = (uint8_t)stCPU.K;
    uint8_t u8Result = u8Rd - u8K;

    *stCPU.Rd = u8Result;

    //--Flags
    SUB_Half_Carry( u8Rd, u8K, u8Result);
    SUB_Full_Carry( u8Rd, u8K, u8Result);
    SUB_Overflow_Flag( u8Rd, u8K, u8Result);
    R8_Negative_Flag( u8Result);
    R8_Zero_Flag( u8Result);
    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_SBC( void )
{
    uint8_t u8Rd = *stCPU.Rd;
    uint8_t u8Rr = *stCPU.Rr;
    uint8_t u8C = stCPU.pstRAM->stRegisters.SREG.C;
    uint8_t u8Result = u8Rd - u8Rr - u8C;

    *stCPU.Rd = u8Result;

    //--Flags
    SUB_Half_Carry( u8Rd, u8Rr, u8Result);
    SUB_Full_Carry( u8Rd, u8Rr, u8Result);
    SUB_Overflow_Flag( u8Rd, u8Rr, u8Result);
    R8_Negative_Flag( u8Result);
    if (u8Result)
    {
        stCPU.pstRAM->stRegisters.SREG.Z = 0;
    }
    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_SBCI( void )
{
    uint8_t u8Rd = *stCPU.Rd;
    uint8_t u8K = (uint8_t)stCPU.K;
    uint8_t u8C = stCPU.pstRAM->stRegisters.SREG.C;
    uint8_t u8Result = u8Rd - u8K - u8C;

    *stCPU.Rd = u8Result;

    //--Flags
    SUB_Half_Carry( u8Rd, u8K, u8Result);
    SUB_Full_Carry( u8Rd, u8K, u8Result);
    SUB_Overflow_Flag( u8Rd, u8K, u8Result);
    R8_Negative_Flag( u8Result);
    if (u8Result)
    {
        stCPU.pstRAM->stRegisters.SREG.Z = 0;
    }
    Signed_Flag();
}


//---------------------------------------------------------------------------
inline void SBIW_Overflow_Flag( uint16_t Rd_, uint16_t Result_)
{
    stCPU.pstRAM->stRegisters.SREG.V =
             ((Rd_ & 0x8000 ) == 0x8000) && ((Result_ & 0x8000) == 0);

}

//---------------------------------------------------------------------------
inline void SBIW_Full_Carry( uint16_t Rd_, uint16_t Result_)
{
    stCPU.pstRAM->stRegisters.SREG.C =
             ((Rd_ & 0x8000 ) == 0) && ((Result_ & 0x8000) == 0x8000);
}

//---------------------------------------------------------------------------
static void AVR_Opcode_SBIW( void )
{
    uint16_t u16Rd = *stCPU.Rd16;
    uint16_t u16Result;

    //fprintf( stderr, "SBIW: RD=[%4X], K=[%2X]\n", u16Rd, stCPU.K );
    u16Result = u16Rd - stCPU.K;

    *stCPU.Rd16 = u16Result;
    //fprintf( stderr, "  Result=[%4X]\n", u16Result );

```

```
    SBIW_Full_Carry( u16Rd, u16Result);
    SBIW_Overflow_Flag( u16Rd, u16Result);
    R16_Negative_Flag( u16Result);
    R16_Zero_Flag( u16Result);
    Signed_Flag();

}

//---------------------------------------------------------------------------
static void AVR_Opcode_AND( void )
{
    uint8_t u8Rd = *stCPU.Rd;
    uint8_t u8Rr = *stCPU.Rr;
    uint8_t u8Result = u8Rd & u8Rr;

    *stCPU.Rd = u8Result;

    //--Update Status registers;
    stCPU.pstRAM->stRegisters.SREG.V = 0;
    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ANDI( void )
{
    uint8_t u8Rd = *stCPU.Rd;
    uint8_t u8Result = u8Rd & (uint8_t)stCPU.K;

    *stCPU.Rd = u8Result;

    //--Update Status registers;
    stCPU.pstRAM->stRegisters.SREG.V = 0;
    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_OR( void )
{
    uint8_t u8Rd = *stCPU.Rd;
    uint8_t u8Rr = *stCPU.Rr;
    uint8_t u8Result = u8Rd | u8Rr;

    *stCPU.Rd = u8Result;

    //--Update Status registers;
    stCPU.pstRAM->stRegisters.SREG.V = 0;
    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ORI( void )
{
    uint8_t u8Rd = *stCPU.Rd;
    uint8_t u8Result = u8Rd |  (uint8_t)stCPU.K;

    *stCPU.Rd = u8Result;

    //--Update Status registers;
    stCPU.pstRAM->stRegisters.SREG.V = 0;
    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_EOR( void )
{
    uint8_t u8Rd = *stCPU.Rd;
    uint8_t u8Rr = *stCPU.Rr;
    uint8_t u8Result = u8Rd ^ u8Rr;

    *stCPU.Rd = u8Result;

    //--Update Status registers;
    stCPU.pstRAM->stRegisters.SREG.V = 0;
    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Signed_Flag();
}

//---------------------------------------------------------------------------
```

```
static void AVR_Opcode_COM( void )
{
    // 1's complement.
    uint8_t u8Result = *stCPU.Rd;
    u8Result = (0xFF - u8Result);

    *stCPU.Rd = u8Result;

    //--Update Status registers;
    stCPU.pstRAM->stRegisters.SREG.V = 0;
    stCPU.pstRAM->stRegisters.SREG.C = 1;
    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Signed_Flag();
}

//---------------------------------------------------------------------------
inline void NEG_Overflow_Flag( uint8_t u8Result_ )
{
    stCPU.pstRAM->stRegisters.SREG.V = (u8Result_ == 0x80);
}

//---------------------------------------------------------------------------
inline void NEG_Carry_Flag( uint8_t u8Result_ )
{
    stCPU.pstRAM->stRegisters.SREG.C = (u8Result_ != 0x00);
}

//---------------------------------------------------------------------------
static void AVR_Opcode_NEG( void )
{
    // 2's complement.
    uint8_t u8Result = *stCPU.Rd;
    u8Result = (0 - u8Result);

    *stCPU.Rd = u8Result;

    //--Update Status registers;
    NEG_Overflow_Flag(  u8Result );
    NEG_Carry_Flag(  u8Result );
    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_SBR( void )
{
    // Set Bits in Register
    uint8_t u8Result = *stCPU.Rd;
    u8Result |= ((uint8_t)stCPU.K);

    *stCPU.Rd = u8Result;

    //--Update Status registers;
    stCPU.pstRAM->stRegisters.SREG.V = 0;
    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_CBR( void )
{
    // Clear Bits in Register
    uint8_t u8Result = *stCPU.Rd;
    u8Result &= ~((uint8_t)stCPU.K);

    *stCPU.Rd = u8Result;

    //--Update Status registers;
    stCPU.pstRAM->stRegisters.SREG.V = 0;
    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Signed_Flag();
}

//---------------------------------------------------------------------------
inline void INC_Overflow_Flag( uint8_t u8Result_ )
{
    stCPU.pstRAM->stRegisters.SREG.V = (u8Result_ == 0x80);
}

//---------------------------------------------------------------------------
static void AVR_Opcode_INC( void )
{
    uint8_t u8Result;
```

```
    u8Result = *stCPU.Rd + 1;

    *stCPU.Rd = u8Result;

    //--Update Status registers;
    INC_Overflow_Flag(  u8Result );
    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Signed_Flag();
}
//---------------------------------------------------------------------------
inline void DEC_Overflow_Flag( uint8_t u8Result_ )
{
    stCPU.pstRAM->stRegisters.SREG.V = (u8Result_ == 0x7F);
}
//---------------------------------------------------------------------------
static void AVR_Opcode_DEC( void )
{
    uint8_t u8Result;
    u8Result = *stCPU.Rd - 1;

    *stCPU.Rd = u8Result;

    //--Update Status registers;
    DEC_Overflow_Flag(  u8Result );
    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_SER( void )
{
    *stCPU.Rd = 0xFF;
}

//---------------------------------------------------------------------------
inline void Mul_Carry_Flag( uint16_t R_ )
{
    stCPU.pstRAM->stRegisters.SREG.C = ((R_  & 0x8000) == 0x8000);
}

//---------------------------------------------------------------------------
inline void Mul_Zero_Flag( uint16_t R_ )
{
    stCPU.pstRAM->stRegisters.SREG.Z = (R_ == 0);
}

//---------------------------------------------------------------------------
static void AVR_Opcode_MUL( void )
{
    uint16_t u16Product;
    uint16_t u16R1;
    uint16_t u16R2;

    u16R1 = *stCPU.Rd;
    u16R2 = *stCPU.Rr;

    u16Product = u16R1 * u16R2;

    stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = u16Product;

    //-- Update Flags --
    Mul_Zero_Flag( u16Product);
    Mul_Carry_Flag( u16Product);
}

//---------------------------------------------------------------------------
static void AVR_Opcode_MULS( void )
{
    int16_t s16Product;
    int16_t s16R1;
    int16_t s16R2;

    s16R1 = (int8_t)*stCPU.Rd;
    s16R2 = (int8_t)*stCPU.Rr;

    s16Product = s16R1 * s16R2;

    stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = (uint16_t)s16Product;

    //-- Update Flags --
    Mul_Zero_Flag( (uint16_t)s16Product);
    Mul_Carry_Flag( (uint16_t)s16Product);
}

//---------------------------------------------------------------------------
```

```
static void AVR_Opcode_MULSU( void )
{
    int16_t s16Product;
    int16_t s16R1;
    uint16_t u16R2;

    s16R1 = (int8_t)*stCPU.Rd;
    u16R2 = *stCPU.Rr;

    s16Product = s16R1 * u16R2;

    stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = (uint16_t)s16Product;

    //-- Update Flags --
    Mul_Zero_Flag( (uint16_t)s16Product);
    Mul_Carry_Flag( (uint16_t)s16Product);
}

//---------------------------------------------------------------------------
static void AVR_Opcode_FMUL( void )
{
    uint16_t u16Product;
    uint16_t u16R1;
    uint16_t u16R2;

    u16R1 = *stCPU.Rd;
    u16R2 = *stCPU.Rr;

    u16Product = u16R1 * u16R2;

    stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = u16Product << 1;

    //-- Update Flags --
    Mul_Zero_Flag( u16Product);
    Mul_Carry_Flag( u16Product);
}

//---------------------------------------------------------------------------
static void AVR_Opcode_FMULS( void )
{
    int16_t s16Product;
    int16_t s16R1;
    int16_t s16R2;

    s16R1 = (int8_t)*stCPU.Rd;
    s16R2 = (int8_t)*stCPU.Rr;

    s16Product = s16R1 * s16R2;

    stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = ((uint16_t)s16Product) << 1;

    //-- Update Flags --
    Mul_Zero_Flag( (uint16_t)s16Product);
    Mul_Carry_Flag( (uint16_t)s16Product);
}

//---------------------------------------------------------------------------
static void AVR_Opcode_FMULSU( void )
{
    int16_t s16Product;
    int16_t s16R1;
    uint16_t u16R2;

    s16R1 = (int8_t)*stCPU.Rd;
    u16R2 = *stCPU.Rr;

    s16Product = s16R1 * u16R2;

    stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = ((uint16_t)s16Product) << 1;

    //-- Update Flags --
    Mul_Zero_Flag( (uint16_t)s16Product);
    Mul_Carry_Flag( (uint16_t)s16Product);
}

//---------------------------------------------------------------------------
static void AVR_Opcode_DES( void )
{
}

//---------------------------------------------------------------------------
static inline Unconditional_Jump( uint16_t u16Addr_ )
{
    stCPU.u16PC = u16Addr_;
    stCPU.u16ExtraPC = 0;

    // Feature -- Terminate emulator if jump-to-zero encountered at runtime.
```

```
    if (stCPU.u16PC == 0 && stCPU.bExitOnReset)
    {
        exit(0);
    }
}

//---------------------------------------------------------------------------
static inline Relative_Jump( uint16_t u16Offset_ )
{
    // u16Offset_ Will always be 1 or 2, based on the size of the next opcode
    // in a program

    stCPU.u16PC += u16Offset_;
    stCPU.u16ExtraPC = 0;
    stCPU.u16ExtraCycles += u16Offset_;
}

//---------------------------------------------------------------------------
static void AVR_Opcode_RJMP( void )
{
    int32_t s32NewPC = (int32_t)stCPU.u16PC + (int32_t)stCPU.k_s + 1;

    Unconditional_Jump(  (uint16_t)s32NewPC );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_IJMP( void )
{
    Unconditional_Jump(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_EIJMP( void )
{
}

//---------------------------------------------------------------------------
static void AVR_Opcode_JMP( void )
{
    Unconditional_Jump(  (uint16_t)stCPU.k );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_RCALL( void )
{
    // Push the next instruction address onto the stack
    uint16_t u16SP = (((uint16_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
                     (((uint16_t)stCPU.pstRAM->stRegisters.SPL.r));

    uint16_t u16StoredPC = stCPU.u16PC + 1;

    Data_Write(  u16SP, (uint8_t)(u16StoredPC & 0x00FF));
    Data_Write(  u16SP - 1, (uint8_t)(u16StoredPC >> 8));

    // Stack is post-decremented
    u16SP -= 2;

    // Set the new PC (relative call)
    int32_t s32NewPC = (int32_t)stCPU.u16PC + (int32_t)stCPU.k_s + 1;
    uint16_t u16NewPC = (uint16_t)s32NewPC;

    // Store the new SP.
    stCPU.pstRAM->stRegisters.SPH.r = (u16SP >> 8);
    stCPU.pstRAM->stRegisters.SPL.r = (u16SP & 0x00FF);

    // Set the new PC
    Unconditional_Jump(  u16NewPC );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ICALL( void )
{
    // Push the next instruction address onto the stack
    uint16_t u16SP = (((uint16_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
                     (((uint16_t)stCPU.pstRAM->stRegisters.SPL.r));

    uint16_t u16StoredPC = stCPU.u16PC + 1;

    Data_Write(  u16SP, (uint8_t)(u16StoredPC & 0x00FF));
    Data_Write(  u16SP - 1, (uint8_t)(u16StoredPC >> 8));

    // Stack is post-decremented
    u16SP -= 2;

    // Set the new PC
    uint16_t u16NewPC = stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z;

```

```
    // Store the new SP.
    stCPU.pstRAM->stRegisters.SPH.r = (u16SP >> 8);
    stCPU.pstRAM->stRegisters.SPL.r = (u16SP & 0x00FF);

    // Set the new PC
    Unconditional_Jump(  u16NewPC );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_EICALL( void )
{
}

//---------------------------------------------------------------------------
static void AVR_Opcode_CALL( void )
{
    // See ICALL for documentation
    uint16_t u16SP = (((uint16_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
                     (((uint16_t)stCPU.pstRAM->stRegisters.SPL.r));

    uint16_t u16StoredPC = stCPU.u16PC + 2;

    Data_Write(  u16SP, (uint8_t)(u16StoredPC & 0x00FF));
    Data_Write(  u16SP - 1, (uint8_t)(u16StoredPC >> 8));

    u16SP -= 2;

    uint16_t u16NewPC = stCPU.k;

    stCPU.pstRAM->stRegisters.SPH.r = (u16SP >> 8);
    stCPU.pstRAM->stRegisters.SPL.r = (u16SP & 0x00FF);

    Unconditional_Jump(  u16NewPC );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_RET( void )
{
    // Pop the next instruction off of the stack, pre-incrementing
    uint16_t u16SP = (((uint16_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
                     (((uint16_t)stCPU.pstRAM->stRegisters.SPL.r));
    u16SP += 2;

    uint16_t u16High = Data_Read(  u16SP - 1 );
    uint16_t u16Low = Data_Read(  u16SP );
    uint16_t u16NewPC = (u16High << 8) | u16Low;

    stCPU.pstRAM->stRegisters.SPH.r = (u16SP >> 8);
    stCPU.pstRAM->stRegisters.SPL.r = (u16SP & 0x00FF);

    // Set new PC based on address read from stack
    Unconditional_Jump(  u16NewPC );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_RETI( void )
{
    uint16_t u16SP = (((uint16_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
                     (((uint16_t)stCPU.pstRAM->stRegisters.SPL.r));
    u16SP += 2;

    uint16_t u16High = Data_Read(  u16SP - 1 );
    uint16_t u16Low = Data_Read(  u16SP );
    uint16_t u16NewPC = (u16High << 8) | u16Low;

    stCPU.pstRAM->stRegisters.SPH.r = (u16SP >> 8);
    stCPU.pstRAM->stRegisters.SPL.r = (u16SP & 0x00FF);

//-- Enable interrupts
    stCPU.pstRAM->stRegisters.SREG.I = 1;
    Unconditional_Jump( u16NewPC );

//-- Run callout functions registered when we return from interrupt.
    InterruptCallout_Run( false, 0 );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_CPSE( void )
{
    if (*stCPU.Rr == *stCPU.Rd)
    {
        uint8_t u8NextOpSize = AVR_Opcode_Size( stCPU.pu16ROM[ stCPU.u16PC + 1 ] );
        Relative_Jump(  u8NextOpSize + 1 );
    }
}

//---------------------------------------------------------------------------
```

```
inline void CP_Half_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
{
    stCPU.pstRAM->stRegisters.SREG.H =
            ( ((~Rd_ & Rr_)     & 0x08 )
            | ((Rr_ & (Result_)) & 0x08 )
            | (((Result_) & ~Rd_) & 0x08) ) != false;
}

//---------------------------------------------------------------------------
inline void CP_Full_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
{
    stCPU.pstRAM->stRegisters.SREG.C =
            ( ((~Rd_ & Rr_)     & 0x80 )
            | ((Rr_ & (Result_)) & 0x80 )
            | (((Result_) & ~Rd_) & 0x80) ) != false;
}

//---------------------------------------------------------------------------
inline void CP_Overflow_Flag( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
{
    stCPU.pstRAM->stRegisters.SREG.V =
             ( ((Rd_ & ~Rr_ & ~Result_)  & 0x80 )
             | ((~Rd_ & Rr_ & Result_) & 0x80 ) ) != 0;
}

//---------------------------------------------------------------------------
static void AVR_Opcode_CP( void )
{
    // Compare
    uint8_t u8Result;
    uint8_t u8Rd = *stCPU.Rd;
    uint8_t u8Rr  = *stCPU.Rr;

    u8Result = u8Rd - u8Rr;

    //---
    CP_Half_Carry(  u8Rd, u8Rr, u8Result );
    CP_Overflow_Flag(  u8Rd, u8Rr, u8Result );
    CP_Full_Carry(  u8Rd, u8Rr, u8Result );

    R8_Zero_Flag(  u8Result );
    R8_Negative_Flag(  u8Result );

    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_CPC( void )
{
    // Compare with carry
    uint8_t u8Result;
    uint8_t u8Rd = *stCPU.Rd;
    uint8_t u8Rr  = *stCPU.Rr;
    uint8_t u8C = (stCPU.pstRAM->stRegisters.SREG.C == 1);

    u8Result = u8Rd - u8Rr - u8C;

    //---
    CP_Half_Carry(  u8Rd, u8Rr, u8Result );
    CP_Overflow_Flag(  u8Rd, u8Rr, u8Result );
    CP_Full_Carry(  u8Rd, u8Rr, u8Result );

    R8_CPC_Zero_Flag(  u8Result );
    R8_Negative_Flag(  u8Result );

    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_CPI( void )
{
    // Compare with immediate
    uint8_t u8Result;
    uint8_t u8Rd = *stCPU.Rd;
    uint8_t u8K  = stCPU.K;

    u8Result = u8Rd - u8K;

    //---
    CP_Half_Carry(  u8Rd, u8K, u8Result );
    CP_Overflow_Flag(  u8Rd, u8K, u8Result );
    CP_Full_Carry(  u8Rd, u8K, u8Result );

    R8_Zero_Flag(  u8Result );
    R8_Negative_Flag(  u8Result );

    Signed_Flag();
```

```
}

//---------------------------------------------------------------------------
static void AVR_Opcode_SBRC( void )
{
    // Skip if Bit in IO register clear
    if ((*stCPU.Rd & (1 << stCPU.b)) == 0)
    {
        uint8_t u8NextOpSize = AVR_Opcode_Size( stCPU.pu16ROM[ stCPU.u16PC + 1 ] );
        Relative_Jump(  u8NextOpSize + 1 );
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_SBRS( void )
{
    // Skip if Bit in IO register set
    if ((*stCPU.Rd & (1 << stCPU.b)) != 0)
    {
        uint8_t u8NextOpSize = AVR_Opcode_Size( stCPU.pu16ROM[ stCPU.u16PC + 1 ] );
        Relative_Jump(  u8NextOpSize + 1 );
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_SBIC( void )
{
    // Skip if Bit in IO register clear
    uint8_t u8IOVal = Data_Read(  32 + stCPU.A );
    if ((u8IOVal & (1 << stCPU.b)) == 0)
    {
        uint8_t u8NextOpSize = AVR_Opcode_Size( stCPU.pu16ROM[ stCPU.u16PC + 1 ] );
        Relative_Jump(  u8NextOpSize + 1 );
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_SBIS( void )
{
    // Skip if Bit in IO register set
    uint8_t u8IOVal = Data_Read(  32 + stCPU.A );
    if ((u8IOVal & (1 << stCPU.b)) != 0)
    {
        uint8_t u8NextOpSize = AVR_Opcode_Size( stCPU.pu16ROM[ stCPU.u16PC + 1 ] );
        Relative_Jump(  u8NextOpSize + 1 );
    }
}

//---------------------------------------------------------------------------
static inline Conditional_Branch( void )
{
    stCPU.u16PC = (uint16_t)((int16_t)stCPU.u16PC + stCPU.k_s + 1);
    stCPU.u16ExtraPC = 0;
    stCPU.u16ExtraCycles++;
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRBS( void )
{
    if (0 != (stCPU.pstRAM->stRegisters.SREG.r & (1 << stCPU.b)))
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRBC( void )
{
    if (0 == (stCPU.pstRAM->stRegisters.SREG.r & (1 << stCPU.b)))
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BREQ( void )
{
    if (1 == stCPU.pstRAM->stRegisters.SREG.Z)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRNE( void )
{
    if (0 == stCPU.pstRAM->stRegisters.SREG.Z)
```

```
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRCS( void )
{
    if (1 == stCPU.pstRAM->stRegisters.SREG.C)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRCC( void )
{
    if (0 == stCPU.pstRAM->stRegisters.SREG.C)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRSH( void )
{
    if (0 == stCPU.pstRAM->stRegisters.SREG.C)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRLO( void )
{
    if (1 == stCPU.pstRAM->stRegisters.SREG.C)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRMI( void )
{
    if (1 == stCPU.pstRAM->stRegisters.SREG.N)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRPL( void )
{
    if (0 == stCPU.pstRAM->stRegisters.SREG.N)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRGE( void )
{
    if (0 == stCPU.pstRAM->stRegisters.SREG.S)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRLT( void )
{
    if (1 == stCPU.pstRAM->stRegisters.SREG.S)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRHS( void )
{
    if (1 == stCPU.pstRAM->stRegisters.SREG.H)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
```

```
static void AVR_Opcode_BRHC( void )
{
    if (0 == stCPU.pstRAM->stRegisters.SREG.H)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRTS( void )
{
    if (1 == stCPU.pstRAM->stRegisters.SREG.T)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRTC( void )
{
    if (0 == stCPU.pstRAM->stRegisters.SREG.T)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRVS( void )
{
    if (1 == stCPU.pstRAM->stRegisters.SREG.V)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRVC( void )
{
    if (0 == stCPU.pstRAM->stRegisters.SREG.V)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRIE( void )
{
    if (1 == stCPU.pstRAM->stRegisters.SREG.I)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BRID( void )
{
    if (0 == stCPU.pstRAM->stRegisters.SREG.I)
    {
        Conditional_Branch();
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_MOV( void )
{
    *stCPU.Rd = *stCPU.Rr;
}

//---------------------------------------------------------------------------
static void AVR_Opcode_MOVW( void )
{
    *stCPU.Rd16 = *stCPU.Rr16;
}

//---------------------------------------------------------------------------
static void AVR_Opcode_LDI( void )
{
    *stCPU.Rd = stCPU.K;
}

//---------------------------------------------------------------------------
static void AVR_Opcode_LDS( void )
{
    *stCPU.Rd = Data_Read(  stCPU.K );
}

//---------------------------------------------------------------------------
```

```
static void AVR_Opcode_LD_X_Indirect( void )
{
    *stCPU.Rd =
            Data_Read(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.X );
}
//---------------------------------------------------------------------------
static void AVR_Opcode_LD_X_Indirect_Postinc( void )
{
    *stCPU.Rd =
        Data_Read(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.X++ );
}
//---------------------------------------------------------------------------
static void AVR_Opcode_LD_X_Indirect_Predec( void )
{
    *stCPU.Rd =
        Data_Read(  --stCPU.pstRAM->stRegisters.CORE_REGISTERS.X );
}
//---------------------------------------------------------------------------
static void AVR_Opcode_LD_Y_Indirect( void )
{
    *stCPU.Rd =
        Data_Read(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y );
}
//---------------------------------------------------------------------------
static void AVR_Opcode_LD_Y_Indirect_Postinc( void )
{
    *stCPU.Rd =
        Data_Read(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y++ );
}
//---------------------------------------------------------------------------
static void AVR_Opcode_LD_Y_Indirect_Predec( void )
{
    *stCPU.Rd =
        Data_Read(  --stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y );
}
//---------------------------------------------------------------------------
static void AVR_Opcode_LDD_Y( void )
{
    *stCPU.Rd =
        Data_Read(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y + stCPU.q );
}
//---------------------------------------------------------------------------
static void AVR_Opcode_LD_Z_Indirect( void )
{
    *stCPU.Rd =
        Data_Read(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z );
}
//---------------------------------------------------------------------------
static void AVR_Opcode_LD_Z_Indirect_Postinc( void )
{
    *stCPU.Rd =
        Data_Read(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z++ );

}
//---------------------------------------------------------------------------
static void AVR_Opcode_LD_Z_Indirect_Predec( void )
{
    *stCPU.Rd =
        Data_Read(  --stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z );
}
//---------------------------------------------------------------------------
static void AVR_Opcode_LDD_Z( void )
{
    *stCPU.Rd =
        Data_Read(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z + stCPU.q );
}
//---------------------------------------------------------------------------
static void AVR_Opcode_STS( void )
{
    Data_Write(  stCPU.K, *stCPU.Rd );
}
//---------------------------------------------------------------------------
static void AVR_Opcode_ST_X_Indirect( void )
{
    Data_Write(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.X, *stCPU.Rd );
```

```
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ST_X_Indirect_Postinc( void )
{
    Data_Write(   stCPU.pstRAM->stRegisters.CORE_REGISTERS.X++, *stCPU.Rd );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ST_X_Indirect_Predec( void )
{
    Data_Write(  --stCPU.pstRAM->stRegisters.CORE_REGISTERS.X, *stCPU.Rd );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ST_Y_Indirect( void )
{
    Data_Write(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y, *stCPU.Rd );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ST_Y_Indirect_Postinc( void )
{
    Data_Write(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y++, *stCPU.Rd );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ST_Y_Indirect_Predec( void )
{
    Data_Write(  --stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y, *stCPU.Rd );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_STD_Y( void )
{
    Data_Write(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y + stCPU.q, *stCPU.Rd );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ST_Z_Indirect( void )
{
    Data_Write(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z, *stCPU.Rd );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ST_Z_Indirect_Postinc( void )
{
    Data_Write(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z++ , *stCPU.Rd );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ST_Z_Indirect_Predec( void )
{
    Data_Write(  --stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z , *stCPU.Rd );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_STD_Z( void )
{
    Data_Write(  stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z + stCPU.q, *stCPU.Rd );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_LPM( void )
{
    uint8_t u8Temp;
    if (stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z & 0x0001)
    {
        u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] >> 8);
    }
    else
    {
        u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] & 0x00FF);
    }

    stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0 = u8Temp;
}

//---------------------------------------------------------------------------
static void AVR_Opcode_LPM_Z( void )
{
    uint8_t u8Temp;
    if (stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z & 0x0001)
    {
        u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] >> 8);
    }
    else
```

```
    {
        u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] & 0x00FF);
    }

    *stCPU.Rd = u8Temp;
}

//---------------------------------------------------------------------------
static void AVR_Opcode_LPM_Z_Postinc( void )
{
    uint8_t u8Temp;
    if (stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z & 0x0001)
    {
        u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] >> 8);
    }
    else
    {
        u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] & 0x00FF);
    }

    *stCPU.Rd = u8Temp;
    stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z++;
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ELPM( void )
{
    uint8_t u8Temp;
    if (stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z & 0x0001)
    {
        u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] >> 8);
    }
    else
    {
        u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] & 0x00FF);
    }

    stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0 = u8Temp;
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ELPM_Z( void )
{
    uint8_t u8Temp;
    if (stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z & 0x0001)
    {
        u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] >> 8);
    }
    else
    {
        u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] & 0x00FF);
    }

    *stCPU.Rd = u8Temp;
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ELPM_Z_Postinc( void )
{
    uint8_t u8Temp;
    if (stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z & 0x0001)
    {
        u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] >> 8);
    }
    else
    {
        u8Temp = (uint8_t)(stCPU.pu16ROM[ stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z >> 1 ] & 0x00FF);
    }

    *stCPU.Rd = u8Temp;

    stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z++;
}

//---------------------------------------------------------------------------
static void AVR_Opcode_SPM( void )
{
}

//---------------------------------------------------------------------------
static void AVR_Opcode_SPM_Z_Postinc2( void )
{
}

//---------------------------------------------------------------------------
static void AVR_Opcode_IN( void )
{
```

```
    *stCPU.Rd = Data_Read(  32 + stCPU.A );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_OUT( void )
{
     Data_Write(  32 + stCPU.A , *stCPU.Rd );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_PUSH( void )
{
    uint16_t u16SP = (stCPU.pstRAM->stRegisters.SPL.r) |
                     ((uint16_t)(stCPU.pstRAM->stRegisters.SPH.r) << 8);

    // Store contents from SP to destination register
    Data_Write(  u16SP, *stCPU.Rd );

    // Postdecrement the SP
    u16SP--;

    // Update the SP registers
    stCPU.pstRAM->stRegisters.SPH.r = (uint8_t)(u16SP >> 8);
    stCPU.pstRAM->stRegisters.SPL.r = (uint8_t)(u16SP & 0x00FF);
}

//---------------------------------------------------------------------------
static void AVR_Opcode_POP( void )
{
    // Preincrement the SP
    uint16_t u16SP = (stCPU.pstRAM->stRegisters.SPL.r) |
                     ((uint16_t)(stCPU.pstRAM->stRegisters.SPH.r) << 8);
    u16SP++;

    // Load contents from SP to destination register
    *stCPU.Rd = Data_Read(  u16SP );

    // Update the SP registers
    stCPU.pstRAM->stRegisters.SPH.r = (uint8_t)(u16SP >> 8);
    stCPU.pstRAM->stRegisters.SPL.r = (uint8_t)(u16SP & 0x00FF);
}

//---------------------------------------------------------------------------
static void AVR_Opcode_XCH( void )
{
    uint8_t u8Z;
    uint8_t u8Temp;
    uint16_t u16Addr = stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z;

    u8Z = Data_Read(  u16Addr );
    u8Temp = *stCPU.Rd;

    *stCPU.Rd = u8Z;
    Data_Write(  u16Addr, u8Temp );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_LAS( void )
{
    uint8_t u8Z;
    uint8_t u8Temp;

    uint16_t u16Addr = stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z;

    u8Z = Data_Read(  u16Addr );
    u8Temp = *stCPU.Rd | u8Z;

    *stCPU.Rd = u8Z;
    Data_Write(  u16Addr, u8Temp );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_LAC( void )
{
    uint8_t u8Z;
    uint8_t u8Temp;

    uint16_t u16Addr = stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z;

    u8Z = Data_Read(  u16Addr );
    u8Temp = *stCPU.Rd & ~(u8Z);
    *stCPU.Rd = u8Z;

    Data_Write(  u16Addr, u8Temp );
}

//---------------------------------------------------------------------------
```

```
static void AVR_Opcode_LAT( void )
{
    uint8_t u8Z;
    uint8_t u8Temp;

    uint16_t u16Addr = stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z;

    u8Z = Data_Read(  u16Addr );
    u8Temp = *stCPU.Rd ^ u8Z;
    *stCPU.Rd = u8Z;

    Data_Write(  u16Addr, u8Temp );
}

//---------------------------------------------------------------------------
inline void LSL_HalfCarry_Flag( uint8_t R_ )
{
    stCPU.pstRAM->stRegisters.SREG.H = ((R_ & 0x08) == 0x08);
}

//---------------------------------------------------------------------------
inline void Left_Carry_Flag( uint8_t R_  )
{
    stCPU.pstRAM->stRegisters.SREG.C = ((R_ & 0x80) == 0x80);
}

//---------------------------------------------------------------------------
inline void Rotate_Overflow_Flag()
{
    stCPU.pstRAM->stRegisters.SREG.V = ( stCPU.pstRAM->stRegisters.SREG.N ^ stCPU.pstRAM->stRegisters.SREG.C );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_LSL( void )
{
    // Logical shift left
    uint8_t u8Result = 0;
    uint8_t u8Temp = *stCPU.Rd;

    u8Result = (u8Temp << 1);
    *stCPU.Rd = u8Result;

    // ---- Update flags ----
    LSL_HalfCarry_Flag(  u8Result);
    Left_Carry_Flag( u8Temp);

    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Rotate_Overflow_Flag();
    Signed_Flag();
}

//---------------------------------------------------------------------------
inline void Right_Carry_Flag( uint8_t R_  )
{
    stCPU.pstRAM->stRegisters.SREG.C = ((R_ & 0x01) == 0x01);
}

//---------------------------------------------------------------------------
static void AVR_Opcode_LSR( void )
{
    // Logical shift left
    uint8_t u8Result = 0;
    uint8_t u8Temp = *stCPU.Rd;

    u8Result = (u8Temp >> 1);
    *stCPU.Rd = u8Result;

    // ---- Update flags ----
    Right_Carry_Flag(  u8Temp );
    stCPU.pstRAM->stRegisters.SREG.N = 0;
    R8_Zero_Flag(  u8Result );
    Rotate_Overflow_Flag();
    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ROL( void )
{
    // Rotate left through carry
    uint8_t u8Result = 0;
    uint8_t u8Temp = *stCPU.Rd;

    u8Result = (u8Temp << 1);
    if (stCPU.pstRAM->stRegisters.SREG.C)
    {
```

```
        u8Result |= 0x01;
    }
    *stCPU.Rd = u8Result;

    // ---- Update flags ----
    Left_Carry_Flag(  u8Temp );
    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Rotate_Overflow_Flag();
    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ROR( void )
{
    // Rotate right through carry
    uint8_t u8Result = 0;
    uint8_t u8Temp = *stCPU.Rd;

    u8Result = (u8Temp >> 1);
    if (stCPU.pstRAM->stRegisters.SREG.C)
    {
        u8Result |= 0x80;
    }
    *stCPU.Rd = u8Result;

    // ---- Update flags ----
    Right_Carry_Flag(  u8Temp );
    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Rotate_Overflow_Flag();
    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_ASR( void )
{
    // Shift all bits to the right, keeping sign bit intact
    uint8_t u8Result;
    uint8_t u8Temp = *stCPU.Rd;
    u8Result = (u8Temp & 0x80) | (u8Temp >> 1);
    *stCPU.Rd = u8Result;

    // ---- Update flags ----
    Right_Carry_Flag(  u8Temp );
    R8_Negative_Flag(  u8Result );
    R8_Zero_Flag(  u8Result );
    Rotate_Overflow_Flag();
    Signed_Flag();
}

//---------------------------------------------------------------------------
static void AVR_Opcode_SWAP( void )
{
    uint8_t u8temp;
    u8temp = ((*stCPU.Rd) >> 4) |
             ((*stCPU.Rd) << 4) ;

    *stCPU.Rd = u8temp;
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BSET( void )
{
    stCPU.pstRAM->stRegisters.SREG.r |= (1 << stCPU.b);
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BCLR( void )
{
    stCPU.pstRAM->stRegisters.SREG.r &= ~(1 << stCPU.b);
}

//---------------------------------------------------------------------------
static void AVR_Opcode_SBI( void )
{
    uint8_t u8Temp = Data_Read(  stCPU.A + 32 );
    u8Temp |= (1 << stCPU.b);
    Data_Write(  stCPU.A + 32, u8Temp );
}

//---------------------------------------------------------------------------
static void AVR_Opcode_CBI( void )
{
    uint8_t u8Temp = Data_Read(  stCPU.A + 32 );
    u8Temp &= ~(1 << stCPU.b);
    Data_Write(  stCPU.A + 32, u8Temp );
```

```
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BST( void )
{
    if ((*stCPU.Rd) & (1 << stCPU.b))
    {
        stCPU.pstRAM->stRegisters.SREG.T = 1;
    }
    else
    {
        stCPU.pstRAM->stRegisters.SREG.T = 0;
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BLD( void )
{
    if (stCPU.pstRAM->stRegisters.SREG.T)
    {
        *(stCPU.Rd) |= (1 << stCPU.b);
    }
    else
    {
        *(stCPU.Rd) &= ~(1 << stCPU.b);
    }
}

//---------------------------------------------------------------------------
static void AVR_Opcode_BREAK( void )
{
    // Unimplemented - since this requires debugging HW...
}

//---------------------------------------------------------------------------
static void AVR_Opcode_SLEEP( void )
{
    stCPU.bAsleep = true;
}

//---------------------------------------------------------------------------
static void AVR_Opcode_WDR( void )
{
    stCPU.u32WDTCount = 0;    // Reset watchdog timer counter
}

//---------------------------------------------------------------------------
AVR_Opcode AVR_Opcode_Function( uint16_t OP_ )
{
    switch (OP_)
    {
    case 0x0000: return AVR_Opcode_NOP;

    case 0x9409: return AVR_Opcode_IJMP;
    case 0x9419: return AVR_Opcode_EIJMP;

    case 0x9508: return AVR_Opcode_RET;
    case 0x9509: return AVR_Opcode_ICALL;
    case 0x9518: return AVR_Opcode_RETI;
    case 0x9519: return AVR_Opcode_EICALL;
    case 0x9588: return AVR_Opcode_SLEEP;
    case 0x9598: return AVR_Opcode_BREAK;
    case 0x95A8: return AVR_Opcode_WDR;
    case 0x95C8: return AVR_Opcode_LPM;
    case 0x95D8: return AVR_Opcode_ELPM;
    case 0x95E8: return AVR_Opcode_SPM;
    case 0x95F8: return AVR_Opcode_SPM_Z_Postinc2;
    }

    switch( OP_ & 0xFF8F)
    {
    case 0x9408: return AVR_Opcode_BSET;
    case 0x9488: return AVR_Opcode_BCLR;
    }

    switch (OP_ & 0xFF88)
    {
    case 0x0300: return AVR_Opcode_MULSU;
    case 0x0308: return AVR_Opcode_FMUL;
    case 0x0380: return AVR_Opcode_FMULS;
    case 0x0388: return AVR_Opcode_FMULSU;
    }

    switch (OP_ & 0xFF0F)
    {
    case 0x940B: return AVR_Opcode_DES;
    case 0xEF0F: return AVR_Opcode_SER;
```

```
    }

    switch (OP_ & 0xFF00)
    {
    case 0x0100: return AVR_Opcode_MOVW;
    case 0x9600: return AVR_Opcode_ADIW;
    case 0x9700: return AVR_Opcode_SBIW;

    case 0x9800: return AVR_Opcode_CBI;
    case 0x9900: return AVR_Opcode_SBIC;
    case 0x9A00: return AVR_Opcode_SBI;
    case 0x9B00: return AVR_Opcode_SBIS;
    }

    switch (OP_ & 0xFE0F)
    {
    case 0x8008: return AVR_Opcode_LD_Y_Indirect;
    case 0x8000: return AVR_Opcode_LD_Z_Indirect;
    case 0x8200: return AVR_Opcode_ST_Z_Indirect;
    case 0x8208: return AVR_Opcode_ST_Y_Indirect;

    // -- Single 5-bit register...
    case 0x9000: return AVR_Opcode_LDS;
    case 0x9001: return AVR_Opcode_LD_Z_Indirect_Postinc;
    case 0x9002: return AVR_Opcode_LD_Z_Indirect_Predec;
    case 0x9004: return AVR_Opcode_LPM_Z;
    case 0x9005: return AVR_Opcode_LPM_Z_Postinc;
    case 0x9006: return AVR_Opcode_ELPM_Z;
    case 0x9007: return AVR_Opcode_ELPM_Z_Postinc;
    case 0x9009: return AVR_Opcode_LD_Y_Indirect_Postinc;
    case 0x900A: return AVR_Opcode_LD_Y_Indirect_Predec;
    case 0x900C: return AVR_Opcode_LD_X_Indirect;
    case 0x900D: return AVR_Opcode_LD_X_Indirect_Postinc;
    case 0x900E: return AVR_Opcode_LD_X_Indirect_Predec;
    case 0x900F: return AVR_Opcode_POP;

    case 0x9200: return AVR_Opcode_STS;
    case 0x9201: return AVR_Opcode_ST_Z_Indirect_Postinc;
    case 0x9202: return AVR_Opcode_ST_Z_Indirect_Predec;
    case 0x9204: return AVR_Opcode_XCH;
    case 0x9205: return AVR_Opcode_LAS;
    case 0x9206: return AVR_Opcode_LAC;
    case 0x9207: return AVR_Opcode_LAT;
    case 0x9209: return AVR_Opcode_ST_Y_Indirect_Postinc;
    case 0x920A: return AVR_Opcode_ST_Y_Indirect_Predec;
    case 0x920C: return AVR_Opcode_ST_X_Indirect;
    case 0x920D: return AVR_Opcode_ST_X_Indirect_Postinc;
    case 0x920E: return AVR_Opcode_ST_X_Indirect_Predec;
    case 0x920F: return AVR_Opcode_PUSH;

    // -- One-operand instructions
    case 0x9400: return AVR_Opcode_COM;
    case 0x9401: return AVR_Opcode_NEG;
    case 0x9402: return AVR_Opcode_SWAP;
    case 0x9403: return AVR_Opcode_INC;
    case 0x9405: return AVR_Opcode_ASR;
    case 0x9406: return AVR_Opcode_LSR;
    case 0x9407: return AVR_Opcode_ROR;
    case 0x940A: return AVR_Opcode_DEC;

    }
    switch (OP_ & 0xFE0E)
    {
    case 0x940C: return AVR_Opcode_JMP;
    case 0x940E: return AVR_Opcode_CALL;
    }

    switch (OP_ & 0xFE08)
    {

    // -- BLD/BST Encoding
    case 0xF800: return AVR_Opcode_BLD;
    case 0xFA00: return AVR_Opcode_BST;
    // -- SBRC/SBRS Encoding
    case 0xFC00: return AVR_Opcode_SBRC;
    case 0xFE00: return AVR_Opcode_SBRS;
    }

    switch (OP_ & 0xFC07)
    {
    // -- Conditional branches
    case 0xF000: return AVR_Opcode_BRCS;
    // case 0xF000: return AVR_Opcode_BRLO;            // AKA AVR_Opcode_BRCS;
    case 0xF001: return AVR_Opcode_BREQ;
    case 0xF002: return AVR_Opcode_BRMI;
    case 0xF003: return AVR_Opcode_BRVS;
    case 0xF004: return AVR_Opcode_BRLT;
```

```
    case 0xF006: return AVR_Opcode_BRTS;
    case 0xF007: return AVR_Opcode_BRIE;
    case 0xF400: return AVR_Opcode_BRCC;
    // case 0xF400: return AVR_Opcode_BRSH;            // AKA AVR_Opcode_BRCC;
    case 0xF401: return AVR_Opcode_BRNE;
    case 0xF402: return AVR_Opcode_BRPL;
    case 0xF403: return AVR_Opcode_BRVC;
    case 0xF404: return AVR_Opcode_BRGE;
    case 0xF405: return AVR_Opcode_BRHC;
    case 0xF406: return AVR_Opcode_BRTC;
    case 0xF407: return AVR_Opcode_BRID;
    }

    switch (OP_ & 0xFC00)
    {
    // -- 4-bit register pair
    case 0x0200: return AVR_Opcode_MULS;

    // -- 5-bit register pairs --
    case 0x0400: return AVR_Opcode_CPC;
    case 0x0800: return AVR_Opcode_SBC;
    case 0x0C00: return AVR_Opcode_ADD;
    // case 0x0C00: return AVR_Opcode_LSL; (!! Implemented with: " add rd, rd"
    case 0x1000: return AVR_Opcode_CPSE;
    case 0x1300: return AVR_Opcode_ROL;
    case 0x1400: return AVR_Opcode_CP;
    case 0x1C00: return AVR_Opcode_ADC;
    case 0x1800: return AVR_Opcode_SUB;
    case 0x2000: return AVR_Opcode_AND;
    // case 0x2000: return AVR_Opcode_TST; (!! Implemented with: " and rd, rd"
    case 0x2400: return AVR_Opcode_EOR;
    case 0x2C00: return AVR_Opcode_MOV;
    case 0x2800: return AVR_Opcode_OR;

    // -- 5-bit register pairs -- Destination = R1:R0
    case 0x9C00: return AVR_Opcode_MUL;
    }

    switch (OP_ & 0xF800)
    {
    case  0xB800: return AVR_Opcode_OUT;
    case  0xB000: return AVR_Opcode_IN;
    }

    switch (OP_ & 0xF000)
    {
    // -- Register immediate --
    case 0x3000: return AVR_Opcode_CPI;
    case 0x4000: return AVR_Opcode_SBCI;
    case 0x5000: return AVR_Opcode_SUBI;
    case 0x6000: return AVR_Opcode_ORI;// return AVR_Opcode_SBR;
    case 0x7000: return AVR_Opcode_ANDI;

    //-- 12-bit immediate
    case 0xC000: return AVR_Opcode_RJMP;
    case 0xD000: return AVR_Opcode_RCALL;

    // -- Register immediate
    case 0xE000: return AVR_Opcode_LDI;
    }

    switch (OP_ & 0xD208)
    {
    // -- 7-bit signed offset
    case 0x8000: return AVR_Opcode_LDD_Z;
    case 0x8008: return AVR_Opcode_LDD_Y;
    case 0x8200: return AVR_Opcode_STD_Z;
    case 0x8208: return AVR_Opcode_STD_Y;
    }

    return AVR_Opcode_NOP;
}

//---------------------------------------------------------------------------
void AVR_RunOpcode(  uint16_t OP_ )
{
    AVR_Opcode myOpcode = AVR_Opcode_Function( OP_);
    myOpcode();
}
```

4.41 avr_opcodes.h File Reference

AVR CPU - Opcode interface.

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

Typedefs

- typedef void(∗ **AVR_Opcode**)(void)

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 1856 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 2057 of file avr_opcodes.c.

4.42 avr_opcodes.h

```
/*****************************************************************************
 *     (     (                      (     |
```

```
 *    )\ )  )\ )     (              )\ )    |
 *   (()/( (()/(    )\      (    (  (()/(    | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\    |
 *                                          | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __AVR_OPCODES_H__
#define __AVR_OPCODES_H__

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

//---------------------------------------------------------------------------
// Format opcode function jump table
typedef void (*AVR_Opcode)( void );
//---------------------------------------------------------------------------
AVR_Opcode AVR_Opcode_Function( uint16_t OP_ );

//---------------------------------------------------------------------------
void AVR_RunOpcode(  uint16_t OP_ );

#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

```
/****************************************************************************
 *     (       (                      (      |
 *    )\ )  )\ )     (              )\ )    |
 *   (()/( (()/(    )\      (    (  (()/(    | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
```

```
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                        | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __AVR_PERIPHERAL_H__
#define __AVR_PERIPHERAL_H__

#include <stdint.h>

//---------------------------------------------------------------------------
// Peripheral callout functions - used to implement arbitrary peripherals
// which are able to intercept/react to read/write operations to specific
// I/O addresses.
//---------------------------------------------------------------------------

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
{
    PeriphInit          pfInit;
    PeriphRead          pfRead;
    PeriphWrite         pfWrite;
    PeriphClock         pfClock;

    void                *pvContext;

    uint8_t             u8AddrStart;
    uint8_t             u8AddrEnd;
} AVRPeripheral;

#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

```
/******************************************************************************
 *      (     (                      (     |
 *    )\ )  )\ )     (              )\ )   |
 *   (()/( (()/(     )\     (   (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_)) (((_)()\   )\  /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
 *   | |_   | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __|  | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|    |_____| /_/ \_\  \_/   |_|_\   |
 *                                         | "Yeah, it does Arduino..."
```

```
 * --------------------------------------+---------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __AVR_PERIPHREGS_H__
#define __AVR_PERIPHREGS_H__

#include <stdint.h>

//---------------------------------------------------------------------------
// UART/USART register struct definitions.
//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int MPCM0: 1;
            unsigned int U2X0 : 1;
            unsigned int UPE0 : 1;
            unsigned int DOR0 : 1;
            unsigned int FE0  : 1;
            unsigned int UDRE0 : 1;
            unsigned int TXC0 : 1;
            unsigned int RXC0 : 1;
        };
    };
} AVR_UCSR0A;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int TXB80  : 1;
            unsigned int RXB80  : 1;
            unsigned int UCSZ02 : 1;
            unsigned int TXEN0  : 1;
            unsigned int RXEN0  : 1;
            unsigned int UDRIE0 : 1;
            unsigned int TXCIE0 : 1;
            unsigned int RXCIE0 : 1;
        };
    };
} AVR_UCSR0B;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int UCPOL0  : 1;
            unsigned int UCPHA0  : 1;
            unsigned int UDORD0  : 1;
            unsigned int USBS0   : 1;
            unsigned int UPM00   : 1;
            unsigned int UPM01   : 1;
            unsigned int UMSEL00 : 1;
            unsigned int UMSEL01 : 1;
        };
    };
} AVR_UCSR0C;

//---------------------------------------------------------------------------
// TWI interface register struct definitions
//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int reserved : 1;
            unsigned int TWAM0 : 1;
            unsigned int TWAM1 : 1;
            unsigned int TWAM2 : 1;
            unsigned int TWAM3 : 1;
            unsigned int TWAM4 : 1;
```

```
            unsigned int TWAM5 : 1;
            unsigned int TWAM6 : 1;
        };
    };
} AVR_TWAMR;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int TWIE  : 1;
            unsigned int reserved : 1;
            unsigned int TWEN  : 1;
            unsigned int TWWC  : 1;
            unsigned int TWSTO : 1;
            unsigned int TWSTA : 1;
            unsigned int TWEA  : 1;
            unsigned int TWINT : 1;
        };
    };
} AVR_TWCR;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int TWGCE: 1;
            unsigned int TWA0 : 1;
            unsigned int TWA1 : 1;
            unsigned int TWA2 : 1;
            unsigned int TWA3 : 1;
            unsigned int TWA4 : 1;
            unsigned int TWA5 : 1;
            unsigned int TWA6 : 1;
        };
    };
} AVR_TWAR;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int TWPS0 : 1;
            unsigned int TWPS1 : 1;
            unsigned int reserved : 1;
            unsigned int TWPS3 : 1;
            unsigned int TWPS4 : 1;
            unsigned int TWPS5 : 1;
            unsigned int TWPS6 : 1;
            unsigned int TWPS7 : 1;
        };
    };
} AVR_TWSR;

//---------------------------------------------------------------------------
// Timer 2 register struct  __attribute__ ((__packed__)) definitins.
//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int TCR2BUB : 1;
            unsigned int TCR2AUB : 1;
            unsigned int OCR2BUB : 1;
            unsigned int OCR2AUB : 1;
            unsigned int TCN2UB  : 1;
            unsigned int AS2     : 1;
            unsigned int EXCLK   : 1;
            unsigned int reserved : 1;
        };
    };
} AVR_ASSR;
```

```
//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int CS20  : 1;
            unsigned int CS21  : 1;
            unsigned int CS22  : 1;
            unsigned int WGM22 : 1;
            unsigned int reserved : 2;
            unsigned int FOC2B : 1;
            unsigned int FOC2A : 1;
        };
    };
} AVR_TCCR2B;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int WGM20 : 1;
            unsigned int WGM21 : 1;
            unsigned int reserved : 2;
            unsigned int COM2B0 : 1;
            unsigned int COM2B1 : 1;
            unsigned int COM2A0 : 1;
            unsigned int COM2A1 : 1;
        };
    };
} AVR_TCCR2A;

//---------------------------------------------------------------------------
// Timer 1 Register struct  __attribute__ ((__packed__)) definitions
//---------------------------------------------------------------------------

typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int WGM10 : 1;
            unsigned int WGM11 : 1;
            unsigned int reserved : 2;
            unsigned int COM1B0 : 1;
            unsigned int COM1B1 : 1;
            unsigned int COM1A0 : 1;
            unsigned int COM1A1 : 1;
        };
    };
} AVR_TCCR1A;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int CS10  : 1;
            unsigned int CS11  : 1;
            unsigned int CS12  : 1;
            unsigned int WGM12 : 1;
            unsigned int WGM13 : 1;
            unsigned int reserved : 1;
            unsigned int ICES1 : 1;
            unsigned int ICNC1 : 1;
        };
    };
} AVR_TCCR1B;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
```

```
        struct  __attribute__ ((__packed__))
        {
            unsigned int reserved : 6;
            unsigned int FOC1B : 1;
            unsigned int FOC1A : 1;
        };
    };
} AVR_TCCR1C;

//---------------------------------------------------------------------------
// A2D converter register definitions
//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int AIN0D : 1;
            unsigned int AIN1D : 1;
            unsigned int reserved : 6;
        };
    };
} AVR_DIDR1;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int ADC0D : 1;
            unsigned int ADC1D : 1;
            unsigned int ADC2D : 1;
            unsigned int ADC3D : 1;
            unsigned int ADC4D : 1;
            unsigned int ADC5D : 1;
            unsigned int reserved : 2;
        };
    };
} AVR_DIDR0;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int MUX0     : 1;
            unsigned int MUX1     : 1;
            unsigned int MUX2     : 1;
            unsigned int MUX3     : 1;
            unsigned int reserved : 1;
            unsigned int ADLAR    : 1;
            unsigned int REFS0    : 1;
            unsigned int REFS1    : 1;
        };
    };
} AVR_ADMUX;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int ADPS0 : 1;
            unsigned int ADPS1 : 1;
            unsigned int ADPS2 : 1;
            unsigned int ADIE  : 1;
            unsigned int ADIF  : 1;
            unsigned int ADATE : 1;
            unsigned int ADSC  : 1;
            unsigned int ADEN  : 1;
        };
    };
} AVR_ADCSRA;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
```

```
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int ADTS0     : 1;
            unsigned int ADTS1     : 1;
            unsigned int ADTS2     : 1;
            unsigned int reserved  : 3;
            unsigned int ACMD      : 1;
            unsigned int reserved_ : 1;
        };
    };
} AVR_ADCSRB;

//---------------------------------------------------------------------------
// Timer interrupt mask registers.
//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int TOIE2    : 1;
            unsigned int OCIE2A   : 1;
            unsigned int OCIE2B   : 1;
            unsigned int reserved : 5;
        };
    };
} AVR_TIMSK2;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int TOIE1    : 1;
            unsigned int OCIE1A   : 1;
            unsigned int OCIE1B   : 1;
            unsigned int reserved : 2;
            unsigned int ICIE1    : 1;
            unsigned int reserved_ : 2;
        };
    };
} AVR_TIMSK1;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int TOIE0    : 1;
            unsigned int OCIE0A   : 1;
            unsigned int OCIE0B   : 1;
            unsigned int reserved : 5;
        };
    };
} AVR_TIMSK0;

//---------------------------------------------------------------------------
// Pin change interrupt mask bit definitions
//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int PCINT16 : 1;
            unsigned int PCINT17 : 1;
            unsigned int PCINT18 : 1;
            unsigned int PCINT19 : 1;
            unsigned int PCINT20 : 1;
            unsigned int PCINT21 : 1;
            unsigned int PCINT22 : 1;
            unsigned int PCINT23 : 1;
        };
```

```
    };
} AVR_PCMSK2;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int PCINT8 : 1;
            unsigned int PCINT9 : 1;
            unsigned int PCINT10 : 1;
            unsigned int PCINT11 : 1;
            unsigned int PCINT12 : 1;
            unsigned int PCINT13 : 1;
            unsigned int PCINT14 : 1;
            unsigned int PCINT15 : 1;
        };
    };
} AVR_PCMSK1;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int PCINT0 : 1;
            unsigned int PCINT1 : 1;
            unsigned int PCINT2 : 1;
            unsigned int PCINT3 : 1;
            unsigned int PCINT4 : 1;
            unsigned int PCINT5 : 1;
            unsigned int PCINT6 : 1;
            unsigned int PCINT7 : 1;
        };
    };
} AVR_PCMSK0;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int PCIE0 : 1;
            unsigned int PCIE1 : 1;
            unsigned int PCIE2 : 1;
            unsigned int reserved : 5;
        };
    };
} AVR_PCICR;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int ISC00    : 1;
            unsigned int ISC01    : 1;
            unsigned int ISC10    : 1;
            unsigned int ISC11    : 1;
            unsigned int reserved : 4;
        };
    };
} AVR_EICRA;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int PRADC : 1;
            unsigned int PRUSART0 : 1;
            unsigned int PRSPI: 1;
```

```
            unsigned int PRTIM1: 1;
            unsigned int reserved : 1;
            unsigned int PRTIM0 : 1;
            unsigned int PRTIM2 : 1;
            unsigned int PRTWI : 1;
        };
    };
} AVR_PRR;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int CLKPS0 : 1;
            unsigned int CLKPS1 : 1;
            unsigned int CLKPS2 : 1;
            unsigned int CLKPS3 : 1;
            unsigned int reserved : 3;
            unsigned int CLKPCE : 1;
        };
    };
} AVR_CLKPR;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int WDP0 : 1;
            unsigned int WDP1 : 1;
            unsigned int WDP2 : 1;
            unsigned int WDE  : 1;
            unsigned int WDCE : 1;
            unsigned int WDP3 : 1;
            unsigned int WDIE : 1;
            unsigned int WDIF : 1;
        };
    };
} AVR_WDTCSR;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int C : 1;
            unsigned int Z : 1;
            unsigned int N : 1;
            unsigned int V : 1;
            unsigned int S : 1;
            unsigned int H : 1;
            unsigned int T : 1;
            unsigned int I : 1;
        };
    };
} AVR_SREG;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int SP0 : 1;
            unsigned int SP1 : 1;
            unsigned int SP2 : 1;
            unsigned int SP3 : 1;
            unsigned int SP4 : 1;
            unsigned int SP5 : 1;
            unsigned int SP6 : 1;
            unsigned int SP7 : 1;
        };
    };
} AVR_SPL;

```

```
//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int SP8      : 1;
            unsigned int SP9      : 1;
            unsigned int SP10     : 1;
            unsigned int reserved : 5;
        };
    };
} AVR_SPH;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int SELFPRGEN : 1;
            unsigned int PGERS     : 1;
            unsigned int PGWRT     : 1;
            unsigned int BLBSET    : 1;
            unsigned int RWWSRE    : 1;
            unsigned int RWWSB     : 1;
            unsigned int SPMIE     : 1;
        };
    };
} AVR_SPMCSR;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int IVCE      : 1;
            unsigned int IVSEL     : 1;
            unsigned int reserved  : 2;
            unsigned int PUD       : 1;
            unsigned int BODSE     : 1;
            unsigned int BODS      : 1;
            unsigned int reserved_ : 1;
        };
    };
} AVR_MCUCR;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int PORF     : 1;
            unsigned int EXTRF    : 1;
            unsigned int BORF     : 1;
            unsigned int WDRF     : 1;
            unsigned int reserved : 4;
        };
    };
} AVR_MCUSR;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int SE       : 1;
            unsigned int SM0      : 1;
            unsigned int SM1      : 1;
            unsigned int SM2      : 1;
            unsigned int reserved : 4;
        };
    };
} AVR_SMCR;
```

```
//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int ACIS0 : 1;
            unsigned int ACIS1 : 1;
            unsigned int ACIC  : 1;
            unsigned int ACIE  : 1;
            unsigned int ACI   : 1;
            unsigned int ACO   : 1;
            unsigned int ACBG  : 1;
            unsigned int ACD   : 1;
        };
    };
} AVR_ACSR;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int SPR0 : 1;
            unsigned int SPR1 : 1;
            unsigned int CPHA : 1;
            unsigned int CPOL : 1;
            unsigned int MSTR : 1;
            unsigned int DORD : 1;
            unsigned int SPE  : 1;
            unsigned int SPIE : 1;
        };
    };
} AVR_SPCR;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int SPI2X    : 1;
            unsigned int reserved : 5;
            unsigned int WCOL     : 1;
            unsigned int SPIF     : 1;
        };
    };
} AVR_SPSR;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int PSRSYNC  : 1;
            unsigned int PSRASY   : 1;
            unsigned int reserved : 5;
            unsigned int TSM      : 1;
        };
    };
} AVR_GTCCR;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int WGM00    : 1;
            unsigned int WGM01    : 1;
            unsigned int reserved : 2;
            unsigned int COM0B0   : 1;
            unsigned int COM0B1   : 1;
            unsigned int COM0A0   : 1;
```

```
            unsigned int COM0A1   : 1;
        };
    };
} AVR_TCCR0A;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int CS00     : 1;
            unsigned int CS01     : 1;
            unsigned int CS02     : 1;
            unsigned int WGM02    : 1;
            unsigned int reserved : 2;
            unsigned int FOC0B    : 1;
            unsigned int FOC0A    : 1;
        };
    };
} AVR_TCCR0B;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int EERE     : 1;
            unsigned int EEPE     : 1;
            unsigned int EEMPE    : 1;
            unsigned int EERIE    : 1;
            unsigned int EEPM0    : 1;
            unsigned int EEPM1    : 1;
            unsigned int reserved : 2;
        };
    };
} AVR_EECR;

//---------------------------------------------------------------------------
// External interrupt flag register definitions
//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int INTF0    : 1;
            unsigned int INTF1    : 1;
            unsigned int reserved : 6;
        };
    };
} AVR_EIFR;

//---------------------------------------------------------------------------
// External interrupt mask register definitions
//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int INT0    : 1;
            unsigned int INT1    : 1;
            unsigned int reserved : 6;
        };
    };
} AVR_EIMSK;

//---------------------------------------------------------------------------
// Pin (GPIO) register definitions
//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
```

```
            unsigned int PIN0 : 1;
            unsigned int PIN1 : 1;
            unsigned int PIN2 : 1;
            unsigned int PIN3 : 1;
            unsigned int PIN4 : 1;
            unsigned int PIN5 : 1;
            unsigned int PIN6 : 1;
            unsigned int PIN7 : 1;
        };
    };
} AVR_PIN;

//---------------------------------------------------------------------------
// Data-direction register (GPIO) definitions
//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int DDR0 : 1;
            unsigned int DDR1 : 1;
            unsigned int DDR2 : 1;
            unsigned int DDR3 : 1;
            unsigned int DDR4 : 1;
            unsigned int DDR5 : 1;
            unsigned int DDR6 : 1;
            unsigned int DDR7 : 1;
        };
    };
} AVR_DDR;

//---------------------------------------------------------------------------
// Port (GPIO) register definitions
//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int PORT0 : 1;
            unsigned int PORT1 : 1;
            unsigned int PORT2 : 1;
            unsigned int PORT3 : 1;
            unsigned int PORT4 : 1;
            unsigned int PORT5 : 1;
            unsigned int PORT6 : 1;
            unsigned int PORT7 : 1;
        };
    };
} AVR_PORT;


//---------------------------------------------------------------------------
// Timer interrupt flag register struct  __attribute__ ((__packed__)) definitions
//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int TOV0     : 1;
            unsigned int OCF0A    : 1;
            unsigned int OCF0B    : 1;
            unsigned int reserved : 5;
        };
    };
} AVR_TIFR0;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int TOV1      : 1;
            unsigned int OCF1A     : 1;
            unsigned int OCF1B     : 1;
            unsigned int reserved  : 2;
```

```
                unsigned int ICF1      : 1;
                unsigned int reserved_ : 2;
            };
        };
} AVR_TIFR1;

//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int TOV2     : 1;
            unsigned int OCF2A    : 1;
            unsigned int OCF2B    : 1;
            unsigned int reserved : 5;
        };
    };
} AVR_TIFR2;

//---------------------------------------------------------------------------
// Pin-change interrupt flag bits
//---------------------------------------------------------------------------
typedef struct  __attribute__ ((__packed__))
{
    union  __attribute__ ((__packed__))
    {
        uint8_t r;
        struct  __attribute__ ((__packed__))
        {
            unsigned int PCIF0    : 1;
            unsigned int PCIF1    : 1;
            unsigned int PCIF2    : 1;
            unsigned int reserved : 5;
        };
    };
} AVR_PCIFR;

#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

```
/***************************************************************************
 *       (     (                      (     |
 *      )\ )  )\ )   (              )\ )   |
 *     (()/( (()/(   )\     (   (  (()/(   | -- [ Funkenstein ] -------------
 *      /(_)) /(_))((((_)()\  )\  /(_))   | -- [ Litle ] -------------------
```

```
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                        | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __AVR_REGISTERFILE_H__
#define __AVR_REGISTERFILE_H__

//---------------------------------------------------------------------------
#include "avr_coreregs.h"
#include "avr_periphregs.h"

//---------------------------------------------------------------------------
typedef struct
{
    //-- 0x00
    AVR_CoreRegisters CORE_REGISTERS;

    //-- 0x20
    AVR_PIN     PINA;
    AVR_DDR     DDRA;
    AVR_PORT    PORTA;

    //-- 0x23
    AVR_PIN     PINB;
    AVR_DDR     DDRB;
    AVR_PORT    PORTB;

    //-- 0x26
    AVR_PIN     PINC;
    AVR_DDR     DDRC;
    AVR_PORT    PORTC;

    //-- 0x29
    AVR_PIN     PIND;
    AVR_DDR     DDRD;
    AVR_PORT    PORTD;

    //-- 0x2C
    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;

    //-- 0x35
    AVR_TIFR0   TIFR0;
    AVR_TIFR1   TIFR1;
    AVR_TIFR2   TIFR2;

    //-- 0x38
    uint8_t     RESERVED_0x38;
    uint8_t     RESERVED_0x39;
    uint8_t     RESERVED_0x3A;

    //-- 0x3B
    AVR_PCIFR   PCIFR;
    AVR_EIFR    EIFR;
    AVR_EIMSK   EIMSK;

    //-- 0x3E
    uint8_t     GPIOR0;

    //-- 0x3F
    AVR_EECR    EECR;

    //-- 0x40
    uint8_t     EEDR;
    uint8_t     EEARL;
    uint8_t     EEARH;

    //-- 0x43
    AVR_GTCCR   GTCCR;
    AVR_TCCR0A  TCCR0A;
    AVR_TCCR0B  TCCR0B;
    uint8_t     TCNT0;
    uint8_t     OCR0A;
    uint8_t     OCR0B;

```

```
//-- 0x49
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;

//-- 0x53
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;

//-- 0x60
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;

    //--0xB0
    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;

    //--0xC0
    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;

} AVRRegisterFile;


#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 2014-15, 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_ )
```

```
{
    // Don't add multiple breakpoints at the same address
    if (BreakPoint_EnabledAtAddress(  u16Addr_ ))
    {
        return;
    }

    BreakPoint_t *pstNewBreak = NULL;

    pstNewBreak = (BreakPoint_t*)malloc( sizeof(BreakPoint_t) );

    pstNewBreak->next = stCPU.pstBreakPoints;
    pstNewBreak->prev = NULL;

    pstNewBreak->u16Addr = u16Addr_;

    if (stCPU.pstBreakPoints)
    {
        BreakPoint_t *pstTemp = stCPU.pstBreakPoints;
        pstTemp->prev = pstNewBreak;
    }
    stCPU.pstBreakPoints = pstNewBreak;
}

//---------------------------------------------------------------------------
void BreakPoint_Delete( uint16_t u16Addr_ )
{
    BreakPoint_t *pstTemp = stCPU.pstBreakPoints;

    while (pstTemp)
    {
        if (pstTemp->u16Addr == u16Addr_)
        {
            // Remove node -- reconnect surrounding elements
            BreakPoint_t *pstNext = pstTemp->next;
            if (pstNext)
            {
                pstNext->prev = pstTemp->prev;
            }

            BreakPoint_t *pstPrev = pstTemp->prev;
            if (pstPrev)
            {
                pstPrev->next = pstTemp->next;
            }

            // Adjust list-head if necessary
            if (pstTemp == stCPU.pstBreakPoints)
            {
                stCPU.pstBreakPoints = pstNext;
            }

            // Free the node/iterate to next node.
            pstPrev = pstTemp;
            pstTemp = pstTemp->next;
            free(pstPrev);
        }
        else
        {
            pstTemp = pstTemp->next;
        }
    }
}

//---------------------------------------------------------------------------
bool BreakPoint_EnabledAtAddress( uint16_t u16Addr_ )
{
    BreakPoint_t *pstTemp = stCPU.pstBreakPoints;

    while (pstTemp)
    {
        if (pstTemp->u16Addr == u16Addr_)
        {
            return true;
        }
        pstTemp = pstTemp->next;
    }
    return false;
}
```

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

```
/*****************************************************************************
 *     (     (                       (     |
 *    )\ )  )\ )    (              )\ )   |
 *   (()/( (()/(    )\     (    (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] ------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                         | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __BREAKPOINT_H__
#define __BREAKPOINT_H__

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

#include "avr_cpu.h"

//---------------------------------------------------------------------------
typedef struct _BreakPoint
{
    struct _BreakPoint *next;
    struct _BreakPoint *prev;

    uint16_t    u16Addr;
} BreakPoint_t;

//---------------------------------------------------------------------------
void BreakPoint_Insert( uint16_t u16Addr_ );

//---------------------------------------------------------------------------
void BreakPoint_Delete( uint16_t u16Addr_ );

//---------------------------------------------------------------------------
bool BreakPoint_EnabledAtAddress( uint16_t u16Addr_ );

#endif

```

4.53 code_profile.c File Reference

Code profiling (exeuction and coverage) functionality.

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "debug_sym.h"
#include "code_profile.h"
#include "avr_disasm.h"
#include "tlv_file.h"
```

Data Structures

- struct Profile_t
- struct FunctionProfileTLV_t
- struct FunctionCoverageTLV_t
- struct AddressCoverageTLV_t

Functions

- static void **Profile_TLVInit** (void)
- static void **Profile_FunctionCoverage** (const char ∗szFunc_, uint32_t u32FuncSize_, uint32_t u32HitSize_)
- static void **Profile_Function** (const char ∗szFunc_, uint64_t u64Cycles_, uint64_t u64CPUCycles_)
- static void **Profile_AddressCoverage** (const char ∗szDisasm_, uint32_t u32Addr_, uint64_t u64Hits_)
- void Profile_Hit (uint32_t u32Addr_)

  Profile_Hit.
- void **Profile_ResetEpoch** (void)
- void **Profile_PrintCoverageDissassembly** (void)
- void Profile_Print (void)

  Profile_Print.
- void Profile_Init (uint32_t u32ROMSize_)

  Profile_Init.

Variables

- static Profile_t ∗ **pstProfile** = 0
- static uint32_t **u32ROMSize** = 0
- static TLV_t ∗ **pstFunctionCoverageTLV** = NULL
- static TLV_t ∗ **pstFunctionProfileTLV** = NULL
- static TLV_t ∗ **pstAddressCoverageTLV** = NULL

4.53.1 Detailed Description

Code profiling (exeuction and coverage) functionality.

Definition in file code_profile.c.

4.53.2 Function Documentation

4.53.2.1 void Profile_Hit (uint32_t *u32Addr_*)

Profile_Hit.

Add to profiling counters for the specified address. This should be called on each ROM/FLASH access (not per cycle)

Parameters

*u32Addr_*	- Address in ROM/FLASH being hit.

Definition at line 127 of file code_profile.c.

4.53.2.2 void Profile_Init (uint32_t *u32ROMSize_*)

Profile_Init.

Iniitialze the code profiling module

Parameters

*u32ROMSize_*	- Size of the CPU's ROM/FLASH

Definition at line 280 of file code_profile.c.

4.53.2.3 void Profile_Print (void)

Profile_Print.

Display the cumulative profiling stats

Definition at line 214 of file code_profile.c.

4.54 code_profile.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 2014-15, 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 #include "debug_sym.h"
00026 #include "code_profile.h"
00027 #include "avr_disasm.h"
00028 #include "tlv_file.h"
00029
00030 //---------------------------------------------------------------------------
00031 typedef struct
00032 {
00033     Debug_Symbol_t *pstSym;
00034     uint64_t        u64TotalHit;
00035     uint64_t        u64EpochHit;
00036 } Profile_t;
00037
00038 //---------------------------------------------------------------------------
00039 typedef struct
00040 {
00041     uint64_t u64CyclesTotal;
00042     uint64_t u64CPUCycles;
00043     char szSymName[256];
00044 } FunctionProfileTLV_t;
00045
00046 //---------------------------------------------------------------------------
00047 typedef struct
00048 {
00049     uint32_t u32FunctionSize;
00050     uint32_t u32AddressesHit;
00051     char szSymName[256];
00052 } FunctionCoverageTLV_t;
```

```
//---------------------------------------------------------------------------
typedef struct
{
    uint32_t u32CodeAddress;
    uint64_t u64Hits;
    char     szDisasmLine[256];
} AddressCoverageTLV_t;

//---------------------------------------------------------------------------
static Profile_t *pstProfile = 0;
static uint32_t  u32ROMSize = 0;

//---------------------------------------------------------------------------
static TLV_t *pstFunctionCoverageTLV = NULL;
static TLV_t *pstFunctionProfileTLV = NULL;
static TLV_t *pstAddressCoverageTLV = NULL;

//---------------------------------------------------------------------------
static void Profile_TLVInit(void)
{
    pstFunctionProfileTLV = TLV_Alloc( sizeof(FunctionProfileTLV_t));
    pstFunctionProfileTLV->eTag = TAG_CODE_PROFILE_FUNCTION_GLOBAL;

    pstFunctionCoverageTLV = TLV_Alloc( sizeof(FunctionCoverageTLV_t));
    pstFunctionCoverageTLV->eTag = TAG_CODE_COVERAGE_FUNCTION_GLOBAL;

    pstAddressCoverageTLV = TLV_Alloc( sizeof(AddressCoverageTLV_t));
    pstAddressCoverageTLV->eTag = TAG_CODE_COVERAGE_ADDRESS;
}

//---------------------------------------------------------------------------
static void Profile_FunctionCoverage( const char *szFunc_, uint32_t u32FuncSize_, uint32_t u32HitSize_ )
{
    FunctionCoverageTLV_t *pstData = (FunctionCoverageTLV_t*)(&pstFunctionCoverageTLV->au8Data[0]);

    strcpy(pstData->szSymName, szFunc_);
    pstData->u32FunctionSize = u32FuncSize_;
    pstData->u32AddressesHit = u32HitSize_;
    pstFunctionCoverageTLV->u16Len = strlen(szFunc_) + 8; // Size of the static + variable data

    TLV_Write( pstFunctionCoverageTLV );
}

//---------------------------------------------------------------------------
static void Profile_Function( const char *szFunc_, uint64_t u64Cycles_, uint64_t u64CPUCycles_ )
{
    FunctionProfileTLV_t *pstData = (FunctionProfileTLV_t*)(&pstFunctionProfileTLV->au8Data[0]);

    strcpy(pstData->szSymName, szFunc_);
    pstData->u64CyclesTotal = u64Cycles_;
    pstData->u64CPUCycles = u64CPUCycles_;

    pstFunctionProfileTLV->u16Len = strlen(szFunc_) + 16; // Size of the static + variable data

    TLV_Write( pstFunctionProfileTLV );
}

//---------------------------------------------------------------------------
static void Profile_AddressCoverage( const char *szDisasm_, uint32_t u32Addr_, uint64_t u64Hits_ )
{
    AddressCoverageTLV_t *pstData = (AddressCoverageTLV_t*)(&pstAddressCoverageTLV->au8Data[0]);

    strcpy(pstData->szDisasmLine, szDisasm_);

    pstData->u32CodeAddress = u32Addr_;
    pstData->u64Hits = u64Hits_;

    pstAddressCoverageTLV->u16Len = strlen(szDisasm_) + 12;

    TLV_Write( pstAddressCoverageTLV );
}

//---------------------------------------------------------------------------
void Profile_Hit( uint32_t u32Addr_ )
{
    pstProfile[ u32Addr_ ].u64EpochHit++;
    pstProfile[ u32Addr_ ].u64TotalHit++;

    Debug_Symbol_t *pstSym = pstProfile[ u32Addr_ ].pstSym;
    if (pstSym)
    {
        pstSym->u64EpochRefs++;
        pstSym->u64TotalRefs++;
```

```
    }
}

//---------------------------------------------------------------------------
void Profile_ResetEpoch(void)
{
    // Reset the epoch counters for all addreses
    int i;
    for (i = 0; i < u32ROMSize; i++)
    {
        pstProfile[i].u64EpochHit = 0;
    }

    // Reset the per-symbol epoch counters
    Debug_Symbol_t *pstSym;
    int iSymCount = Symbol_Get_Func_Count();
    for (i = 0; i < iSymCount; i++)
    {
        pstSym = Symbol_Func_At_Index(i);
        pstSym->u64EpochRefs = 0;
    }
}

//---------------------------------------------------------------------------
void Profile_PrintCoverageDissassembly(void)
{
    Debug_Symbol_t *pstSym;
    int iSymCount = Symbol_Get_Func_Count();
    int i;
    int j;

    printf( "=====================================================================================\n");
    printf( "Detailed Code Coverage\n");
    printf( "=====================================================================================\n");
    // Go through all of our symbols and show which instructions have actually
    // been hit.
    for (i = 0; i < iSymCount; i++)
    {
        pstSym = Symbol_Func_At_Index(i);

        if (!pstSym)
        {
            break;
        }

        printf("%s:\n", pstSym->szName);
        j = pstSym->u32StartAddr;
        while (j <= (int)pstSym->u32EndAddr)
        {
            uint16_t OP = stCPU.pu16ROM[j];
            stCPU.u16PC = (uint16_t)j;

            if (pstProfile[j].u64TotalHit)
            {
                printf( "[X]" );
            }
            else
            {
                printf( "[ ]" );
            }
            printf(" 0x%04X: [0x%04X] ", stCPU.u16PC, OP);

            AVR_Decode(OP);

            char szBuf[256];
            AVR_Disasm_Function(OP)(szBuf);
            printf( "%s", szBuf );

            Profile_AddressCoverage( szBuf, stCPU.u16PC, pstProfile[j].u64TotalHit );

            j += AVR_Opcode_Size(OP);
        }
        printf("\n");
    }
}

//---------------------------------------------------------------------------
void Profile_Print(void)
{
    uint64_t u64TotalCycles = 0;

    Debug_Symbol_t *pstSym;
    int iSymCount = Symbol_Get_Func_Count();
    int i;
    for (i = 0; i < iSymCount; i++)
    {
```

```
        pstSym = Symbol_Func_At_Index(i);
        u64TotalCycles += pstSym->u64TotalRefs;
    }
    printf("\n\nTotal cycles spent in known functions: %llu\n\n", u64TotalCycles);

    printf( "======================================================================================\n");
    printf( "%60s: CPU utilization(%%)\n", "Function");
    printf( "======================================================================================\n");
    for (i = 0; i < iSymCount; i++)
    {
        pstSym = Symbol_Func_At_Index(i);
        printf( "%60s: %0.3f\n",
                pstSym->szName,
                100.0 * (double)(pstSym->u64TotalRefs) / (double)(u64TotalCycles) );
        Profile_Function( pstSym->szName, pstSym->u64TotalRefs, u64TotalCycles );
    }

    printf( "======================================================================================\n");
    printf( "Code coverage summary:\n");
    printf( "======================================================================================\n");
    int iGlobalHits = 0;
    int iGlobalMisses = 0;
    for (i = 0; i < iSymCount; i++)
    {
        pstSym = Symbol_Func_At_Index(i);
        int j;
        int iHits = 0;
        int iMisses = 0;

        for (j = pstSym->u32StartAddr; j < pstSym->u32EndAddr; j++)
        {
            if (pstProfile[j].u64TotalHit)
            {
                iHits++;
                iGlobalHits++;
            }
            else
            {
                iMisses++;
                iGlobalMisses++;
            }

            // If this is a 2-opcode instruction, skip the next word, as to not skew the results
            uint16_t OP = stCPU.pu16ROM[j];
            if (2 == AVR_Opcode_Size(OP))
            {
                j++;
            }
        }
        printf("%60s: %0.3f\n", pstSym->szName, 100.0 * (double)iHits/(double)(iHits + iMisses));
        Profile_FunctionCoverage(pstSym->szName, iHits + iMisses, iHits);
    }
    printf( "\n[Global Code Coverage] : %0.3f\n",
            100.0 * (double)iGlobalHits/(double)(iGlobalHits + iGlobalMisses));

}
//---------------------------------------------------------------------------
void Profile_Init( uint32_t u32ROMSize_ )
{
    // Allocate a lookup table, one entry per address in ROM to allow us to
    // gather code-coverage and code-profiling information.
    uint32_t u32BufSize = sizeof(Profile_t) * u32ROMSize_ ;
    u32ROMSize = u32ROMSize_;
    pstProfile = (Profile_t*)malloc( u32BufSize );
    memset( pstProfile, 0, u32BufSize );

    // Go through the list of symbols, and associate each function with its
    // address range in the lookup table.
    int iFuncs = Symbol_Get_Func_Count();
    int i;
    for (i = 0; i < iFuncs; i++)
    {
        Debug_Symbol_t *pstSym = Symbol_Func_At_Index( i );
        int j;
        if (pstSym)
        {
            for (j = pstSym->u32StartAddr; j < pstSym->u32EndAddr; j++)
            {
                pstProfile[j].pstSym = pstSym;
            }
        }
    }

    Profile_TLVInit();

    atexit( Profile_Print );
    atexit( Profile_PrintCoverageDissassembly );
```

```
00310 }
```

4.55 code_profile.h File Reference

Code profiling (exeuction and coverage) functionality.

`#include <stdint.h>`

Functions

- void Profile_Init (uint32_t u32ROMSize_)
  Profile_Init.
- void Profile_Hit (uint32_t u32Addr_)
  Profile_Hit.
- void Profile_Print (void)
  Profile_Print.

4.55.1 Detailed Description

Code profiling (exeuction and coverage) functionality.

Definition in file code_profile.h.

4.55.2 Function Documentation

4.55.2.1 void Profile_Hit (uint32_t *u32Addr_*)

Profile_Hit.

Add to profiling counters for the specified address. This should be called on each ROM/FLASH access (not per cycle)

Parameters

*u32Addr_*	- Address in ROM/FLASH being hit.

Definition at line 127 of file code_profile.c.

4.55.2.2 void Profile_Init (uint32_t *u32ROMSize_*)

Profile_Init.

Iniitialze the code profiling module

Parameters

*u32ROMSize_*	- Size of the CPU's ROM/FLASH

Definition at line 280 of file code_profile.c.

4.55.2.3 void Profile_Print (void)

Profile_Print.

Display the cumulative profiling stats

Definition at line 214 of file code_profile.c.

4.56 code_profile.h

```
/****************************************************************************
 *     (     (                     (    |
 *    )\ ) )\ )    (             )\ )   |
 *   (()/( (()/(   )\     (   ( (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\ /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))  )\ _ )\ ((_)((_)(_))   | -- [ AVR ] ---------------------
 *   | |_  | |   (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__  / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\  \_/   |_|_\   |
 *                                      | "Yeah, it does Arduino..."
 * -------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __CODE_PROFILE_H__
#define __CODE_PROFILE_H__

#include <stdint.h>

//---------------------------------------------------------------------------
void Profile_Init( uint32_t u32ROMSize_ );

//---------------------------------------------------------------------------
void Profile_Hit( uint32_t u32Addr_ );

//---------------------------------------------------------------------------
void Profile_Print(void);


#endif

```

4.57 debug_sym.c File Reference

Symbolic debugging support for data and functions.

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

Functions

- void Symbol_Add_Func (const char ∗szName_, const uint32_t u32Addr_, const uint32_t u32Len_)
  Symbol_Add_Func.
- void Symbol_Add_Obj (const char ∗szName_, const uint32_t u32Addr_, const uint32_t u32Len_)
  Symbol_Add_Obj.
- uint32_t Symbol_Get_Obj_Count (void)
  Symbol_Get_Obj_Count.
- uint32_t Symbol_Get_Func_Count (void)
  Symbol_Get_Func_Count.
- Debug_Symbol_t ∗ Symbol_Func_At_Index (uint32_t u32Index_)
  Symbol_Func_At_Index.
- Debug_Symbol_t ∗ Symbol_Obj_At_Index (uint32_t u32Index_)
  Symbol_Obj_At_Index.
- Debug_Symbol_t ∗ Symbol_Find_Func_By_Name (const char ∗szName_)
  Symbol_Find_Func_By_Name.
- Debug_Symbol_t ∗ Symbol_Find_Obj_By_Name (const char ∗szName_)
  Symbol_Find_Obj_By_Name.

Variables

- static Debug_Symbol_t * **pstFuncSymbols** = 0
- static uint32_t **u32FuncCount** = 0
- static Debug_Symbol_t * **pstObjSymbols** = 0
- static uint32_t **u32ObjCount** = 0

4.57.1 Detailed Description

Symbolic debugging support for data and functions.

Definition in file debug_sym.c.

4.57.2 Function Documentation

4.57.2.1 void Symbol_Add_Func (const char * *szName_*, const uint32_t *u32Addr_*, const uint32_t *u32Len_*)

Symbol_Add_Func.

Add a new function into the emulator's debug symbol table.

Parameters

*szName_*	- Name of the symbol (string)
*u32Addr_*	- Start aadress of the function
*u32Len_*	- Size of the function (in bytes)

Definition at line 36 of file debug_sym.c.

4.57.2.2 void Symbol_Add_Obj (const char * *szName_*, const uint32_t *u32Addr_*, const uint32_t *u32Len_*)

Symbol_Add_Obj.

Add a new object into the emulator's debug symbol table.

Parameters

*szName_*	- Name of the symbol (string)
*u32Addr_*	- Start aadress of the object
*u32Len_*	- Size of the object (in bytes)

Definition at line 51 of file debug_sym.c.

4.57.2.3 Debug_Symbol_t* Symbol_Find_Func_By_Name (const char * *szName_*)

Symbol_Find_Func_By_Name.

Search the local debug symbol table for a function specified by name.

Parameters

*szName_*	- Name of the object to look-up

Returns

Pointer to the symbol retrieved, or NULL if index out-of-range.

Definition at line 98 of file debug_sym.c.

4.57.2.4 Debug_Symbol_t∗ Symbol_Find_Obj_By_Name (const char ∗ *szName_*)

Symbol_Find_Obj_By_Name.

Search the local debug symbol table for an object specified by name.

Parameters

*szName_*	- Name of the object to look up

Returns

Pointer to the symbol retrieved, or NULL if index out-of-range.

Definition at line 112 of file debug_sym.c.

4.57.2.5 Debug_Symbol_t∗ Symbol_Func_At_Index (uint32_t *u32Index_*)

Symbol_Func_At_Index.

Return a point to a debug symbol (function) stored in the table at a specific table index.

Parameters

*u32Index_*	- Table index to look up

Returns

Pointer to the symbol retrieved, or NULL if index out-of-range.

Definition at line 78 of file debug_sym.c.

4.57.2.6 uint32_t Symbol_Get_Func_Count (void)

Symbol_Get_Func_Count.

Get the current count of the functions stored in the symbol table.

Returns

Number of functions in the symbol table

Definition at line 72 of file debug_sym.c.

4.57.2.7 uint32_t Symbol_Get_Obj_Count (void)

Symbol_Get_Obj_Count.

Get the current count of the objects stored in the symbol table

Returns

Number of objects in the symbol table

Definition at line 66 of file debug_sym.c.

4.57.2.8 Debug_Symbol_t∗ Symbol_Obj_At_Index (uint32_t *u32Index_*)

Symbol_Obj_At_Index.

Return a point to a debug symbol (object) stored in the table at a specific table index.

Parameters

*u32Index_*	- Table index to look up

Returns

Pointer to the symbol retrieved, or NULL if index out-of-range.

Definition at line 88 of file debug_sym.c.

4.58 debug_sym.c

```
/*****************************************************************************
 *     (     (                      (    |
 *    )\ )  )\ )    (              )\ )  |
 *   (()/( (()/(    )\     (   (  (()/(  | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))    | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))   | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \  | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /  | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\  |
 *                                       | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#include "debug_sym.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//---------------------------------------------------------------------------
static Debug_Symbol_t *pstFuncSymbols = 0;
static uint32_t        u32FuncCount = 0;

static Debug_Symbol_t *pstObjSymbols = 0;
static uint32_t        u32ObjCount = 0;

//---------------------------------------------------------------------------
void Symbol_Add_Func( const char *szName_, const uint32_t u32Addr_, const uint32_t u32Len_ )
{
    pstFuncSymbols = (Debug_Symbol_t*)realloc( pstFuncSymbols, (u32FuncCount + 1) * sizeof(
Debug_Symbol_t));
    Debug_Symbol_t *pstNew = &pstFuncSymbols[u32FuncCount];

    pstNew->eType           = DBG_FUNC;
    pstNew->szName          = strdup( szName_ );
    pstNew->u32StartAddr    = u32Addr_;
    pstNew->u32EndAddr      = u32Addr_ + u32Len_ - 1;
    pstNew->u64EpochRefs    = 0;
    pstNew->u64TotalRefs    = 0;
    u32FuncCount++;
}

//---------------------------------------------------------------------------
void Symbol_Add_Obj( const char *szName_, const uint32_t u32Addr_, const uint32_t u32Len_ )
{
    pstObjSymbols = (Debug_Symbol_t*)realloc( pstObjSymbols, (u32ObjCount + 1) * sizeof(
Debug_Symbol_t));
    Debug_Symbol_t *pstNew = &pstObjSymbols[u32ObjCount];

    pstNew->eType           = DBG_OBJ;
    pstNew->szName          = strdup( szName_ );
    pstNew->u32StartAddr    = u32Addr_;
    pstNew->u32EndAddr      = u32Addr_ + u32Len_ - 1;

    u32ObjCount++;
}


//---------------------------------------------------------------------------
uint32_t Symbol_Get_Obj_Count( void )
{
    return u32ObjCount;
}

//---------------------------------------------------------------------------
uint32_t Symbol_Get_Func_Count( void )
{
```

```
    return u32FuncCount;
}

//---------------------------------------------------------------------------
Debug_Symbol_t *Symbol_Func_At_Index( uint32_t u32Index_ )
{
    if (u32Index_ >= u32FuncCount)
    {
        return 0;
    }
    return &pstFuncSymbols[u32Index_];
}

//---------------------------------------------------------------------------
Debug_Symbol_t *Symbol_Obj_At_Index( uint32_t u32Index_ )
{
    if (u32Index_ >= u32ObjCount)
    {
        return 0;
    }
    return &pstObjSymbols[u32Index_];
}

//---------------------------------------------------------------------------
Debug_Symbol_t *Symbol_Find_Func_By_Name( const char *szName_ )
{
    uint32_t i = 0;
    for (i = 0; i < u32FuncCount; i++)
    {
        if (0 == strcmp(szName_,pstFuncSymbols[i].szName))
        {
            return &pstFuncSymbols[i];
        }
    }
    return 0;
}

//---------------------------------------------------------------------------
Debug_Symbol_t *Symbol_Find_Obj_By_Name( const char *szName_ )
{
    uint32_t i = 0;
    for (i = 0; i < u32ObjCount; i++)
    {
        if (0 == strcmp(szName_,pstObjSymbols[i].szName))
        {
            return &pstObjSymbols[i];
        }
    }
    return 0;
}


```

4.59 debug_sym.h File Reference

Symbolic debugging support for data and functions.

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

Data Structures

- struct Debug_Symbol_t

Enumerations

- enum **Debug_t** { **DBG_OBJ** = 0, **DBG_FUNC**, **DBG_COUNT** }

Functions

- void Symbol_Add_Func (const char *szName_, const uint32_t u32Addr_, const uint32_t u32Len_)

  Symbol_Add_Func.

- void Symbol_Add_Obj (const char ∗szName_, const uint32_t u32Addr_, const uint32_t u32Len_)

  Symbol_Add_Obj.
- uint32_t Symbol_Get_Obj_Count (void)

  Symbol_Get_Obj_Count.
- uint32_t Symbol_Get_Func_Count (void)

  Symbol_Get_Func_Count.
- Debug_Symbol_t ∗ Symbol_Func_At_Index (uint32_t u32Index_)

  Symbol_Func_At_Index.
- Debug_Symbol_t ∗ Symbol_Obj_At_Index (uint32_t u32Index_)

  Symbol_Obj_At_Index.
- Debug_Symbol_t ∗ Symbol_Find_Func_By_Name (const char ∗szName_)

  Symbol_Find_Func_By_Name.
- Debug_Symbol_t ∗ Symbol_Find_Obj_By_Name (const char ∗szName_)

  Symbol_Find_Obj_By_Name.

4.59.1 Detailed Description

Symbolic debugging support for data and functions.

Definition in file debug_sym.h.

4.59.2 Function Documentation

4.59.2.1 void Symbol_Add_Func (const char ∗ *szName_*, const uint32_t *u32Addr_*, const uint32_t *u32Len_*)

Symbol_Add_Func.

Add a new function into the emulator's debug symbol table.

Parameters

*szName_*	- Name of the symbol (string)
*u32Addr_*	- Start aadress of the function
*u32Len_*	- Size of the function (in bytes)

Definition at line 36 of file debug_sym.c.

4.59.2.2 void Symbol_Add_Obj (const char ∗ *szName_*, const uint32_t *u32Addr_*, const uint32_t *u32Len_*)

Symbol_Add_Obj.

Add a new object into the emulator's debug symbol table.

Parameters

*szName_*	- Name of the symbol (string)
*u32Addr_*	- Start aadress of the object
*u32Len_*	- Size of the object (in bytes)

Definition at line 51 of file debug_sym.c.

4.59.2.3 Debug_Symbol_t∗ Symbol_Find_Func_By_Name (const char ∗ *szName_*)

Symbol_Find_Func_By_Name.

Search the local debug symbol table for a function specified by name.

Parameters

*szName_*	- Name of the object to look-up

Returns

Pointer to the symbol retrieved, or NULL if index out-of-range.

Definition at line 98 of file debug_sym.c.

4.59.2.4 Debug_Symbol_t* Symbol_Find_Obj_By_Name (const char * *szName_*)

Symbol_Find_Obj_By_Name.

Search the local debug symbol table for an object specified by name.

Parameters

*szName_*	- Name of the object to look up

Returns

Pointer to the symbol retrieved, or NULL if index out-of-range.

Definition at line 112 of file debug_sym.c.

4.59.2.5 Debug_Symbol_t* Symbol_Func_At_Index (uint32_t *u32Index_*)

Symbol_Func_At_Index.

Return a point to a debug symbol (function) stored in the table at a specific table index.

Parameters

*u32Index_*	- Table index to look up

Returns

Pointer to the symbol retrieved, or NULL if index out-of-range.

Definition at line 78 of file debug_sym.c.

4.59.2.6 uint32_t Symbol_Get_Func_Count (void)

Symbol_Get_Func_Count.

Get the current count of the functions stored in the symbol table.

Returns

Number of functions in the symbol table

Definition at line 72 of file debug_sym.c.

4.59.2.7 uint32_t Symbol_Get_Obj_Count (void)

Symbol_Get_Obj_Count.

Get the current count of the objects stored in the symbol table

Returns

Number of objects in the symbol table

Definition at line 66 of file debug_sym.c.

4.59.2.8 Debug_Symbol_t* Symbol_Obj_At_Index (uint32_t *u32Index_*)

Symbol_Obj_At_Index.

Return a point to a debug symbol (object) stored in the table at a specific table index.

Parameters

*u32Index_*	- Table index to look up

Returns

Pointer to the symbol retrieved, or NULL if index out-of-range.

Definition at line 88 of file debug_sym.c.

4.60 debug_sym.h

```
/*****************************************************************************
 *     (      (                       (      |
 *    )\ )   )\ )     (              )\ )   |
 *   (()/(  (()/(     )\      (    (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_))  /(_))((((_)()\   )\   /(_))     | -- [ Litle ] -------------------
 *   (_))_| (_))   )\ _ )\ ((_) ((_) (_))    | -- [ AVR ] ---------------------
 *   | |_   | |    (_)_\(_)\ \ / /  | _ \   | -- [ Virtual ] -----------------
 *   | __|  | |__   / _ \   \ V /   |   /   | -- [ Runtime ] -----------------
 *   |_|    |____| /_/ \_\   \_/    |_|_\   |
 *                                          | "Yeah, it does Arduino..."
 * ---------------------------------------------+---------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 *****************************************************************************/
#ifndef __DEBUG_SYM_H__
#define __DEBUG_SYM_H__

#include <stdint.h>

//---------------------------------------------------------------------------
typedef enum
{
    DBG_OBJ = 0,
    DBG_FUNC,
//--
    DBG_COUNT
} Debug_t;

//---------------------------------------------------------------------------
typedef struct
{
    Debug_t      eType;
    uint32_t     u32StartAddr;
    uint32_t     u32EndAddr;
    const char *szName;

    uint64_t     u64TotalRefs;
    uint64_t     u64EpochRefs;
} Debug_Symbol_t;

//---------------------------------------------------------------------------
void Symbol_Add_Func( const char *szName_, const uint32_t u32Addr_, const uint32_t u32Len_ )
;

//---------------------------------------------------------------------------
void Symbol_Add_Obj( const char *szName_, const uint32_t u32Addr_, const uint32_t u32Len_ );

//---------------------------------------------------------------------------
uint32_t Symbol_Get_Obj_Count( void );

//---------------------------------------------------------------------------
```

```
uint32_t Symbol_Get_Func_Count( void );

//---------------------------------------------------------------------------
Debug_Symbol_t *Symbol_Func_At_Index( uint32_t u32Index_ );

//---------------------------------------------------------------------------
Debug_Symbol_t *Symbol_Obj_At_Index( uint32_t u32Index_ );

//---------------------------------------------------------------------------
Debug_Symbol_t *Symbol_Find_Func_By_Name( const char *szName_ );

//---------------------------------------------------------------------------
Debug_Symbol_t *Symbol_Find_Obj_By_Name( const char *szName_ );

#endif
```

4.61 elf_print.c File Reference

```
#include "elf_print.h"
#include "elf_types.h"
#include "elf_process.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
```

Functions

- void ELF_PrintHeader (const uint8_t *pau8Buffer_)

  ELF_PrintHeader.
- void ELF_PrintSections (const uint8_t *pau8Buffer_)

  ELF_PrintSections.
- void ELF_PrintSymbols (const uint8_t *pau8Buffer_)

  ELF_PrintSymbols.
- void ELF_PrintProgramHeaders (const uint8_t *pau8Buffer_)

  ELF_PrintProgramHeaders.

4.61.1 Function Documentation

4.61.1.1 void ELF_PrintHeader (const uint8_t * *pau8Buffer_*)

ELF_PrintHeader.

Print the contents of a loaded ELF file's header data to standard output.

Parameters

*pau8Buffer_*	Buffer containing the loaded ELF contents

Definition at line 33 of file elf_print.c.

4.61.1.2 void ELF_PrintProgramHeaders (const uint8_t * *pau8Buffer_*)

ELF_PrintProgramHeaders.

Print the list of program headers stored in the loaded ELF file .

Parameters

*pau8Buffer_*	Buffer containing the loaded ELF contents

Definition at line 246 of file elf_print.c.

4.61.1.3 void ELF_PrintSections (const uint8_t * *pau8Buffer_*)

ELF_PrintSections.

Print a list of named sections contained in the loaded ELF file.

Parameters

*pau8Buffer_*	Buffer containing the loaded ELF contents

Definition at line 147 of file elf_print.c.

4.61.1.4 void ELF_PrintSymbols (const uint8_t * *pau8Buffer_*)

ELF_PrintSymbols.

Print a list of ELF Symbols contained in the loaded ELF file.

Parameters

*pau8Buffer_*	Buffer containing the loaded ELF contents

Definition at line 192 of file elf_print.c.

4.62 elf_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 2014-15, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  ****************************************************************************/
00021 #include "elf_print.h"
00022 #include "elf_types.h"
00023 #include "elf_process.h"
00024
00025 #include <stdint.h>
00026 #include <stdio.h>
00027 #include <stdlib.h>
00028 #include <sys/types.h>
00029 #include <sys/stat.h>
00030 #include <unistd.h>
00031
00032 //---------------------------------------------------------------------------
00033 void ELF_PrintHeader( const uint8_t *pau8Buffer_ )
00034 {
00035     ElfHeader_t *pstHeader = (ElfHeader_t*)pau8Buffer_;
00036
00037     if (!pstHeader)
00038     {
00039         printf("NULL Header object\n");
00040         return;
00041     }
00042
00043     printf( "--[Magic Number:  ");
00044     if (pstHeader->u32IdentMagicNumber == ELF_MAGIC_NUMBER)
00045     {
00046         printf( "Valid]\n");
00047     }
```

```
    else
    {
        printf( "Invalid (%08X)]\n", pstHeader->u32IdentMagicNumber);
        return;
    }

    printf( "--[Format:  ");
    switch (pstHeader->u8IdentFormat)
    {
    case ELF_CLASS_32BIT:   printf( "32Bit]\n" ); break;
    case ELF_CLASS_64BIT:   printf( "64Bit]\n" ); break;
    default:
        printf( "Unknown (0x%02X)]\n", pstHeader->u8IdentFormat );
        break;
    }

    printf( "--[Endianness:  ");
    switch (pstHeader->u8IdentEndianness)
    {
    case ELF_ENDIAN_BIG:    printf( "Big]\n" ); break;
    case ELF_ENDIAN_LITTLE: printf( "Little]\n" ); break;
    default:
        printf( "Unknown (0x%02X)]\n", pstHeader->u8IdentEndianness );
        break;
    }

    printf( "--[Version:  ");
    if (pstHeader->u8IdentVersion == ELF_IDENT_VERSION_ORIGINAL)
    {
        printf( "Original ELF]\n");
    }
    else
    {
        printf( "Unknown (0x%02X)]\n", pstHeader->u8IdentVersion );
    }

    printf( "--[ABI Format: ");
    switch (pstHeader->u8IdentABI)
    {
    case ELF_OSABI_SYSV:        printf( "System V]\n" ); break;
    case ELF_OSABI_HPUX:        printf( "HP-UX]\n" ); break;
    case ELF_OSABI_NETBSD:      printf( "NetBSD]\n" ); break;
    case ELF_OSABI_LINUX:       printf( "Linux]\n" ); break;
    case ELF_OSABI_SOLARIS:     printf( "Solarix]\n" ); break;
    case ELF_OSABI_AIX:         printf( "AIX]\n" ); break;
    case ELF_OSABI_IRIX:        printf( "IRIX]\n" ); break;
    case ELF_OSABI_FREEBSD:     printf( "FreeBSD]\n" ); break;
    case ELF_OSABI_OPENBSD:     printf( "OpenBSD]\n" ); break;
    default:
        printf( "unknown (0x%02X)]\n", pstHeader->u8IdentABI );
        break;
    }

    printf( "--[ABI Version: 0x%02X]\n", pstHeader->u8IdentABIVersion );

    printf( "--[Binary Type: ");
    switch (pstHeader->u16Type)
    {
    case ELF_TYPE_RELOCATABLE:  printf( "Relocatable]\n"); break;
    case ELF_TYPE_EXECUTABLE:   printf( "Executable]\n"); break;
    case ELF_TYPE_SHARED:       printf( "Shared]\n"); break;
    case ELF_TYPE_CORE:         printf( "Core]\n"); break;
    default:
        printf( "unknown (0x%04X)]\n", pstHeader->u16Type );
        break;
    }

    printf( "--[Machine Type: ");
    switch (pstHeader->u16Machine)
    {
    case ELF_MACHINE_SPARC:     printf( "SPARC]\n" ); break;
    case ELF_MACHINE_X86:       printf( "x86]\n" ); break;
    case ELF_MACHINE_MIPS:      printf( "MIPS]\n" ); break;
    case ELF_MACHINE_POWERPC:   printf( "PowerPC]\n" ); break;
    case ELF_MACHINE_ARM:       printf( "ARM]\n" ); break;
    case ELF_MACHINE_SUPERH:    printf( "SuperH]\n" ); break;
    case ELF_MACHINE_IA64:      printf( "IA64]\n" ); break;
    case ELF_MACHINE_X86_64:    printf( "x86-64]\n" ); break;
    case ELF_MACHINE_AARCH64:   printf( "AArch64]\n" ); break;
    case ELF_MACHINE_AVR:       printf( "Atmel AVR]\n" ); break;
    default:
        printf( "unknown (0x%04X)]\n", pstHeader->u16Machine );
        break;
    }

    printf( "--[Version: 0x%08X]\n",            pstHeader->u32Version );
    printf( "--[Entry Point: 0x%08X]\n",        pstHeader->u32EntryPoint );
```

```
    printf( "--[Program Header Offset: 0x%08X]\n", pstHeader->u32PHOffset );
    printf( "--[Section Header Offset: 0x%08X]\n", pstHeader->u32SHOffset );
    printf( "--[Flags: 0x%08X]\n",                  pstHeader->u32Flags );
    printf( "--[Elf Header Size: %d]\n",            pstHeader->u16EHSize );
    printf( "--[Program Header Size: %d]\n",        pstHeader->u16PHSize );
    printf( "--[Program Header Count: %d]\n",       pstHeader->u16PHNum );
    printf( "--[Section Header Size: %d]\n",        pstHeader->u16SHSize );
    printf( "--[Section Header Count: %d]\n",       pstHeader->u16SHNum );
    printf( "--[Sextion Header Index: %d]\n",       pstHeader->u16SHIndex );
}

//---------------------------------------------------------------------------
void ELF_PrintSections( const uint8_t *pau8Buffer_ )
{
    ElfHeader_t *pstHeader = (ElfHeader_t*)pau8Buffer_;
    uint32_t u32StringOffset = ELF_GetHeaderStringTableOffset( pau8Buffer_ );

    uint32_t u32Offset;
    uint16_t u16SHCount;

    u32Offset = pstHeader->u32SHOffset;
    u16SHCount = pstHeader->u16SHNum;

    while (u16SHCount)
    {
        ElfSectionHeader_t *pstSHeader = (ElfSectionHeader_t*)(&
pau8Buffer_[u32Offset]);
        printf( "\n--[Section header @ 0x%08X]\n", u32Offset );
        printf( "--[Name       %s]\n", &pau8Buffer_[u32StringOffset + pstSHeader->u32Name] );

        printf( "--[Type        " );
        switch (pstSHeader->u32Type)
        {
        case ELF_SECTION_TYPE_NULL:     printf( "NULL]\n" ); break;
        case ELF_SECTION_TYPE_NOBITS:   printf( "NOBITS]\n" ); break;
        case ELF_SECTION_TYPE_PROGBITS: printf( "PROGBITS]\n" ); break;
        case ELF_SECTION_TYPE_STRTAB:   printf( "STRTAB]\n" ); break;
        case ELF_SECTION_TYPE_SYMTAB:   printf( "SYMTAB]\n" ); break;
        default:
            printf( "(unknown) 0x%08X]\n", pstSHeader->u32Type );
            break;
        }

        printf( "--[Flags      @ 0x%08X]\n", pstSHeader->u32Flags );
        printf( "--[Address    @ 0x%08X]\n", pstSHeader->u32Address );
        printf( "--[Offset     @ 0x%08X]\n", pstSHeader->u32Offset );
        printf( "--[Size       @ 0x%08X]\n", pstSHeader->u32Size );
        printf( "--[Link       @ 0x%08X]\n", pstSHeader->u32Link );
        printf( "--[Info       @ 0x%08X]\n", pstSHeader->u32Info );
        printf( "--[Alignment  @ 0x%08X]\n", pstSHeader->u32Alignment );
        printf( "--[Entry Size @ 0x%08X]\n", pstSHeader->u32EntrySize );
        //--
        u16SHCount--;
        u32Offset += (pstHeader->u16SHSize);
    }
}

//---------------------------------------------------------------------------
void ELF_PrintSymbols( const uint8_t *pau8Buffer_ )
{
    // Get a pointer to the section header for the symbol table
    uint32_t u32Offset = ELF_GetSymbolTableOffset( pau8Buffer_ );
    ElfSectionHeader_t *pstSymHeader = (ElfSectionHeader_t*)(&
pau8Buffer_[u32Offset]);

    // Get a pointer to the section header for the symbol table's strings
    u32Offset = ELF_GetSymbolStringTableOffset( pau8Buffer_ );
    ElfSectionHeader_t *pstStrHeader = (ElfSectionHeader_t*)(&
pau8Buffer_[u32Offset]);

    // Iterate through the symbol table section, printing out the details of each.
    uint32_t u32SymOffset = pstSymHeader->u32Offset;
    ElfSymbol_t *pstSymbol = (ElfSymbol_t*)(&pau8Buffer_[u32SymOffset]);

    printf( "VALUE SIZE TYPE SCOPE ID NAME\n");
    while (u32SymOffset < (pstSymHeader->u32Offset + pstSymHeader->u32Size))
    {
        printf( "%08X, ", pstSymbol->u32Value );
        printf( "%5d, ", pstSymbol->u32Size );
        uint8_t u8Type = pstSymbol->u8Info & 0x0F;
        switch (u8Type)
        {
            case 0:     printf( "NOTYPE, " ); break;
            case 1:     printf( "OBJECT, " ); break;
            case 2:     printf( "FUNC,   " ); break;
            case 3:     printf( "SECTION," ); break;
            case 4:     printf( "FILE,   " ); break;
```

```
            default:    printf( "Unknown (%02X), ", u8Type); break;
        }
        u8Type = (pstSymbol->u8Info >> 4) & 0x0F;
        switch (u8Type)
        {
            case 0:     printf( "LOCAL, " ); break;
            case 1:     printf( "GLOBAL " ); break;
            case 2:     printf( "WEAK,  " ); break;
            default:    printf( "Unknown (%02X), ", u8Type); break;
        }

        if (65521 == pstSymbol->u16SHIndex) // 65521 == special value "ABS"
        {
            printf("  ABS, ");
        }
        else
        {
            printf( "%5d, ", pstSymbol->u16SHIndex );
        }
        printf( "%s\n", &pau8Buffer_[pstSymbol->u32Name + pstStrHeader->u32Offset] );

        u32SymOffset += pstSymHeader->u32EntrySize;
        pstSymbol = (ElfSymbol_t*)(&pau8Buffer_[u32SymOffset]);
    }
}

//---------------------------------------------------------------------------
void ELF_PrintProgramHeaders( const uint8_t *pau8Buffer_ )
{
    ElfHeader_t *pstHeader = (ElfHeader_t*)(pau8Buffer_);
    uint32_t u32Offset = pstHeader->u32PHOffset;
    uint32_t u16Count = pstHeader->u16PHNum;

    while (u16Count)
    {
        ElfProgramHeader_t *pstProgHeader = (
ElfProgramHeader_t*)(&pau8Buffer_[u32Offset]);
        printf( "Program Header:\n" );
        printf( "--[Type:      %08X]\n",  pstProgHeader->u32Type );
        printf( "--[Offset:    %08X]\n",  pstProgHeader->u32Offset );
        printf( "--[VAddr:     %08X]\n",  pstProgHeader->u32VirtualAddress );
        printf( "--[PAddr:     %08X]\n",  pstProgHeader->u32PhysicalAddress );
        printf( "--[FileSize:  %08X]\n",  pstProgHeader->u32FileSize );
        printf( "--[MemSize:   %08X]\n",  pstProgHeader->u32MemSize );
        printf( "--[Flags:     %08X]\n",  pstProgHeader->u32Flags );
        printf( "--[Alignment: %08X]\n",  pstProgHeader->u32Alignment );

        //---
        u32Offset += pstHeader->u16PHSize;
        u16Count--;
    }
}
```

4.63 elf_print.h File Reference

Functions to print information from ELF files.

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

Functions

- void ELF_PrintHeader (const uint8_t *pau8Buffer_)

  ELF_PrintHeader.
- void ELF_PrintSections (const uint8_t *pau8Buffer_)

  ELF_PrintSections.
- void ELF_PrintSymbols (const uint8_t *pau8Buffer_)

  ELF_PrintSymbols.
- void ELF_PrintProgramHeaders (const uint8_t *pau8Buffer_)

  ELF_PrintProgramHeaders.

4.63.1 Detailed Description

Functions to print information from ELF files.

Definition in file elf_print.h.

4.63.2 Function Documentation

4.63.2.1 void ELF_PrintHeader (const uint8_t * *pau8Buffer_*)

ELF_PrintHeader.

Print the contents of a loaded ELF file's header data to standard output.

Parameters

*pau8Buffer_*	Buffer containing the loaded ELF contents

Definition at line 33 of file elf_print.c.

4.63.2.2 void ELF_PrintProgramHeaders (const uint8_t * *pau8Buffer_*)

ELF_PrintProgramHeaders.

Print the list of program headers stored in the loaded ELF file .

Parameters

*pau8Buffer_*	Buffer containing the loaded ELF contents

Definition at line 246 of file elf_print.c.

4.63.2.3 void ELF_PrintSections (const uint8_t * *pau8Buffer_*)

ELF_PrintSections.

Print a list of named sections contained in the loaded ELF file.

Parameters

*pau8Buffer_*	Buffer containing the loaded ELF contents

Definition at line 147 of file elf_print.c.

4.63.2.4 void ELF_PrintSymbols (const uint8_t * *pau8Buffer_*)

ELF_PrintSymbols.

Print a list of ELF Symbols contained in the loaded ELF file.

Parameters

*pau8Buffer_*	Buffer containing the loaded ELF contents

Definition at line 192 of file elf_print.c.

4.64 elf_print.h

```
/****************************************************************************
 *     (     (                      (     |
 *    )\ )  )\ )      (            )\ )    |
 *   (()/( (()/(     )\      (   (  (()/(    | -- [ Funkenstein ] -------------
```

```
 *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                         | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __ELF_PRINT_H__
#define __ELF_PRINT_H__

#include "elf_types.h"
#include <stdint.h>

//---------------------------------------------------------------------------
void ELF_PrintHeader( const uint8_t *pau8Buffer_ );

//---------------------------------------------------------------------------
void ELF_PrintSections( const uint8_t *pau8Buffer_ );

//---------------------------------------------------------------------------
void ELF_PrintSymbols( const uint8_t *pau8Buffer_ );

//---------------------------------------------------------------------------
void ELF_PrintProgramHeaders( const uint8_t *pau8Buffer_ );

#endif //__ELF_PRINT_H__
```

4.65 elf_process.c File Reference

Functions used to process ELF Binaries.

```
#include "elf_process.h"
#include "elf_types.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
```

Macros

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

Functions

- uint32_t ELF_GetHeaderStringTableOffset (const uint8_t *pau8Buffer_)

  ELF_GetHeaderStringTableOffset.
- uint32_t ELF_GetSymbolStringTableOffset (const uint8_t *pau8Buffer_)

  ELF_GetSymbolStringTableOffset.
- uint32_t ELF_GetSymbolTableOffset (const uint8_t *pau8Buffer_)

  ELF_GetSymbolTableOffset.
- int ELF_LoadFromFile (uint8_t **ppau8Buffer_, const char *szPath_)

  ELF_LoadFromFile.

4.65.1 Detailed Description

Functions used to process ELF Binaries.

Definition in file elf_process.c.

4.65.2 Function Documentation

4.65.2.1 uint32_t ELF_GetHeaderStringTableOffset (const uint8_t * *pau8Buffer_*)

ELF_GetHeaderStringTableOffset.

Returns an offset (in bytes) from the beginning of a buffer containing an elf file, corresponding to the location of the header string table.

Parameters

*pau8Buffer_*	- Pointer to a buffer containing a loaded elf file

Returns

Offset, or 0 if no table found

Definition at line 34 of file elf_process.c.

4.65.2.2 uint32_t ELF_GetSymbolStringTableOffset (const uint8_t * *pau8Buffer_*)

ELF_GetSymbolStringTableOffset.

Returns an offset (in bytes) from the beginning of a buffer containing an elf file, corresponding to the location of the symbol-string table.

Parameters

*pau8Buffer_*	- Pointer to a buffer containing a loaded elf file

Returns

Offset, or 0 if no table found

Definition at line 45 of file elf_process.c.

4.65.2.3 uint32_t ELF_GetSymbolTableOffset (const uint8_t * *pau8Buffer_*)

ELF_GetSymbolTableOffset.

Returns an offset (in bytes) from the beginning of a buffer containing an elf file, corresponding to the location of the symbol table.

Parameters

*pau8Buffer_*	- Pointer to a buffer containing a loaded elf file

Returns

Offset, or 0 if no symbol table

Definition at line 77 of file elf_process.c.

4.65.2.4 int ELF_LoadFromFile (uint8_t ** *ppau8Buffer_*, const char * *szPath_*)

ELF_LoadFromFile.

Read the contents of a specific ELF file from disk into a buffer, allocated to a process-local RAM buffer.

Parameters

*ppau8Buffer_*	- Byte-array pointer, which will point to a newly-allocated buffer on successful read (or NULL) on error.
*szPath_*	- File path to load

Returns

0 on success, -1 on error.

Definition at line 104 of file elf_process.c.

4.66 elf_process.c

```
/*****************************************************************************
 *     (      (                         (     |
 *    )\ )   )\ )     (              )\ )    |
 *   (()/(  (()/(    )\     (   (   (()/(    | -- [ Funkenstein ] -------------
 *    /(_))  /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
 *   | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *   | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\    |
 *                                          | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#include "elf_process.h"
#include "elf_types.h"

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

#define DEBUG_PRINT(...)

//---------------------------------------------------------------------------
uint32_t ELF_GetHeaderStringTableOffset( const uint8_t *pau8Buffer_ )
{
    ElfHeader_t *pstHeader = (ElfHeader_t*)pau8Buffer_;

    ElfSectionHeader_t *pstStringTable =
            (ElfSectionHeader_t*)(&pau8Buffer_[pstHeader->u32SHOffset + (pstHeader->
u16SHSize * pstHeader->u16SHIndex)]);

    return pstStringTable->u32Offset;
}

//---------------------------------------------------------------------------
uint32_t ELF_GetSymbolStringTableOffset( const uint8_t *pau8Buffer_ )
{
    uint32_t u32Offset;
    uint16_t u16SHCount;

    ElfHeader_t *pstHeader = (ElfHeader_t*)pau8Buffer_;
    uint32_t u32StringOffset = ELF_GetHeaderStringTableOffset( pau8Buffer_ );

    u32Offset = pstHeader->u32SHOffset;
    u16SHCount = pstHeader->u16SHNum;

    while (u16SHCount)
    {
        ElfSectionHeader_t *pstSHeader = (ElfSectionHeader_t*)(&
pau8Buffer_[u32Offset]);
        if (
            (ELF_SECTION_TYPE_STRTAB == pstSHeader->u32Type) &&
            (0 == strcmp( ".strtab", &pau8Buffer_[u32StringOffset + pstSHeader->u32Name]))
           )

        {
            return u32Offset;
        }

        //--
        u16SHCount--;
```

```
        u32Offset += pstHeader->u16SHSize;
    }

    return 0;
}

//---------------------------------------------------------------------------
uint32_t ELF_GetSymbolTableOffset( const uint8_t *pau8Buffer_ )
{
    uint32_t u32Offset;
    uint16_t u16SHCount;

    ElfHeader_t *pstHeader = (ElfHeader_t*)pau8Buffer_;

    u32Offset = pstHeader->u32SHOffset;
    u16SHCount = pstHeader->u16SHNum;

    while (u16SHCount)
    {
        ElfSectionHeader_t *pstSHeader = (ElfSectionHeader_t*)(&
pau8Buffer_[u32Offset]);
        if (ELF_SECTION_TYPE_SYMTAB == pstSHeader->u32Type)
        {
            return u32Offset;
        }

        //--
        u16SHCount--;
        u32Offset += pstHeader->u16SHSize;
    }

    return 0;
}

//---------------------------------------------------------------------------
int ELF_LoadFromFile( uint8_t **ppau8Buffer_, const char *szPath_ )
{
    size_t      file_size;
    FILE        *my_file;

    my_file = fopen( szPath_, "rb" );
    if (NULL == my_file)
    {
        DEBUG_PRINT( "Unable to read file @ %s\n", szPath_ );
        return -1;
    }
    fseek(my_file, 0, SEEK_END);
    file_size = ftell(my_file);
    fseek(my_file, 0, SEEK_SET);

    uint8_t *bufptr = (uint8_t*)malloc(file_size);
    *ppau8Buffer_ = bufptr;

    if (!bufptr)
    {
        DEBUG_PRINT( "Unable to malloc elf file buffer\n" );
        fclose( my_file );
        return -1;
    }

    size_t bytes_read = 0;
    while (bytes_read < file_size)
    {
        size_t iter_read = fread( bufptr, 1, 4096, my_file );
        if( iter_read == 0 )
        {
            DEBUG_PRINT( "%d read total\n", bytes_read );
            break;
        }
        bytes_read += iter_read;
        bufptr += iter_read;
    }

    DEBUG_PRINT( "Success reading %d bytes\n", file_size );
    fclose( my_file );
    return 0;
}
```

4.67 elf_process.h File Reference

Functions used to process ELF Binaries.

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

Functions

- uint32_t ELF_GetHeaderStringTableOffset (const uint8_t *pau8Buffer_)

    ELF_GetHeaderStringTableOffset.
- uint32_t ELF_GetSymbolStringTableOffset (const uint8_t *pau8Buffer_)

    ELF_GetSymbolStringTableOffset.
- uint32_t ELF_GetSymbolTableOffset (const uint8_t *pau8Buffer_)

    ELF_GetSymbolTableOffset.
- int ELF_LoadFromFile (uint8_t **ppau8Buffer_, const char *szPath_)

    ELF_LoadFromFile.

4.67.1 Detailed Description

Functions used to process ELF Binaries.

Definition in file elf_process.h.

4.67.2 Function Documentation

4.67.2.1 uint32_t ELF_GetHeaderStringTableOffset (const uint8_t * *pau8Buffer_*)

ELF_GetHeaderStringTableOffset.

Returns an offset (in bytes) from the beginning of a buffer containing an elf file, corresponding to the location of the header string table.

Parameters

*pau8Buffer_*	- Pointer to a buffer containing a loaded elf file

Returns

Offset, or 0 if no table found

Definition at line 34 of file elf_process.c.

4.67.2.2 uint32_t ELF_GetSymbolStringTableOffset (const uint8_t * *pau8Buffer_*)

ELF_GetSymbolStringTableOffset.

Returns an offset (in bytes) from the beginning of a buffer containing an elf file, corresponding to the location of the symbol-string table.

Parameters

*pau8Buffer_*	- Pointer to a buffer containing a loaded elf file

Returns

Offset, or 0 if no table found

Definition at line 45 of file elf_process.c.

4.67.2.3 uint32_t ELF_GetSymbolTableOffset (const uint8_t * *pau8Buffer_*)

ELF_GetSymbolTableOffset.

Returns an offset (in bytes) from the beginning of a buffer containing an elf file, corresponding to the location of the symbol table.

Parameters

*pau8Buffer_*	- Pointer to a buffer containing a loaded elf file

Returns

Offset, or 0 if no symbol table

Definition at line 77 of file elf_process.c.

4.67.2.4 int ELF_LoadFromFile (uint8_t ** *ppau8Buffer_*, const char * *szPath_*)

ELF_LoadFromFile.

Read the contents of a specific ELF file from disk into a buffer, allocated to a process-local RAM buffer.

Parameters

*ppau8Buffer_*	- Byte-array pointer, which will point to a newly-allocated buffer on successful read (or NULL) on error.
*szPath_*	- File path to load

Returns

0 on success, -1 on error.

Definition at line 104 of file elf_process.c.

4.68 elf_process.h

```
/*****************************************************************************
 *     (      (                          (      |
 *    )\ )   )\ )     (                )\ )    |
 *   (()/(  (()/(    )\      (    (   (()/(    | -- [ Funkenstein ] -------------
 *    /(_))  /(_))((((_)()\   )\   /(_))      | -- [ Litle ] -------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ---------------------
 *   | |_  | |     (_)_\(_)\ \ / / | _ \     | -- [ Virtual ] -----------------
 *   | __| | |__    / _ \   \ V /  |   /     | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\     |
 *                                          | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __ELF_PROCESS_H__
#define __ELF_PROCESS_H__

#include "elf_types.h"
#include <stdint.h>

//---------------------------------------------------------------------------
uint32_t ELF_GetHeaderStringTableOffset( const uint8_t *pau8Buffer_ );

//---------------------------------------------------------------------------
uint32_t ELF_GetSymbolStringTableOffset( const uint8_t *pau8Buffer_ );

//---------------------------------------------------------------------------
uint32_t ELF_GetSymbolTableOffset( const uint8_t *pau8Buffer_ );

//---------------------------------------------------------------------------
int ELF_LoadFromFile( uint8_t **ppau8Buffer_, const char *szPath_ );

#endif //__ELF_PROCESS_H__
```

4.69 elf_types.h File Reference

Defines and types used by ELF loader and supporting functionality.

`#include <stdint.h>`

Data Structures

- struct ElfHeader_t
- struct ElfProgramHeader_t
- struct ElfSectionHeader_t
- struct ElfSymbol_t

Macros

- #define **ELF_MAGIC_NUMBER** ((uint32_t)0x464C457F)
- #define **ELF_CLASS_32BIT** ((uint8_t)1)
- #define **ELF_CLASS_64BIT** ((uint8_t)2)
- #define **ELF_ENDIAN_LITTLE** ((uint8_t)1)
- #define **ELF_ENDIAN_BIG** ((uint8_t)2)
- #define **ELF_IDENT_VERSION_ORIGINAL** ((uint8_t)1)
- #define **ELF_OSABI_SYSV** ((uint8_t)0x00)
- #define **ELF_OSABI_HPUX** ((uint8_t)0x01)
- #define **ELF_OSABI_NETBSD** ((uint8_t)0x02)
- #define **ELF_OSABI_LINUX** ((uint8_t)0x03)
- #define **ELF_OSABI_SOLARIS** ((uint8_t)0x06)
- #define **ELF_OSABI_AIX** ((uint8_t)0x07)
- #define **ELF_OSABI_IRIX** ((uint8_t)0x08)
- #define **ELF_OSABI_FREEBSD** ((uint8_t)0x09)
- #define **ELF_OSABI_OPENBSD** ((uint8_t)0x0C)
- #define **ELF_TYPE_RELOCATABLE** ((uint8_t)0x01)
- #define **ELF_TYPE_EXECUTABLE** ((uint8_t)0x02)
- #define **ELF_TYPE_SHARED** ((uint8_t)0x03)
- #define **ELF_TYPE_CORE** ((uint8_t)0x04)
- #define **ELF_MACHINE_SPARC** ((uint16_t)0x02)
- #define **ELF_MACHINE_X86** ((uint16_t)0x03)
- #define **ELF_MACHINE_MIPS** ((uint16_t)0x08)
- #define **ELF_MACHINE_POWERPC** ((uint16_t)0x14)
- #define **ELF_MACHINE_ARM** ((uint16_t)0x28)
- #define **ELF_MACHINE_SUPERH** ((uint16_t)0x2A)
- #define **ELF_MACHINE_IA64** ((uint16_t)0x32)
- #define **ELF_MACHINE_X86_64** ((uint16_t)0x3E)
- #define **ELF_MACHINE_AVR** ((uint16_t)0x53)
- #define **ELF_MACHINE_AARCH64** ((uint16_t)0xB7)
- #define **ELF_VERSION_ORIGINAL** ((uint32_t)1)
- #define **ELF_SECTION_TYPE_NULL** ((uint32_t)0)
- #define **ELF_SECTION_TYPE_PROGBITS** ((uint32_t)1)
- #define **ELF_SECTION_TYPE_SYMTAB** ((uint32_t)2)
- #define **ELF_SECTION_TYPE_STRTAB** ((uint32_t)3)
- #define **ELF_SECTION_TYPE_NOBITS** ((uint32_t)8)

4.69.1 Detailed Description

Defines and types used by ELF loader and supporting functionality.

Definition in file elf_types.h.

4.70 elf_types.h

```
/***************************************************************************
 *     (     (                      (     |
 *    )\ )  )\ )     (             )\ )   |
 *   (()/( (()/(     )\      (    (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\   )\  /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\    |
 *                                         | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ***************************************************************************/
#ifndef __ELF_TYPES_H__
#define __ELF_TYPES_H__

#include <stdint.h>

//---------------------------------------------------------------------------
#define ELF_MAGIC_NUMBER            ((uint32_t)0x464C457F) // "~ELF"

#define ELF_CLASS_32BIT             ((uint8_t)1)
#define ELF_CLASS_64BIT             ((uint8_t)2)

#define ELF_ENDIAN_LITTLE           ((uint8_t)1)
#define ELF_ENDIAN_BIG              ((uint8_t)2)

#define ELF_IDENT_VERSION_ORIGINAL  ((uint8_t)1)

#define ELF_OSABI_SYSV              ((uint8_t)0x00)
#define ELF_OSABI_HPUX              ((uint8_t)0x01)
#define ELF_OSABI_NETBSD            ((uint8_t)0x02)
#define ELF_OSABI_LINUX             ((uint8_t)0x03)
#define ELF_OSABI_SOLARIS           ((uint8_t)0x06)
#define ELF_OSABI_AIX               ((uint8_t)0x07)
#define ELF_OSABI_IRIX              ((uint8_t)0x08)
#define ELF_OSABI_FREEBSD           ((uint8_t)0x09)
#define ELF_OSABI_OPENBSD           ((uint8_t)0x0C)

#define ELF_TYPE_RELOCATABLE        ((uint8_t)0x01)
#define ELF_TYPE_EXECUTABLE         ((uint8_t)0x02)
#define ELF_TYPE_SHARED             ((uint8_t)0x03)
#define ELF_TYPE_CORE               ((uint8_t)0x04)

#define ELF_MACHINE_SPARC           ((uint16_t)0x02)
#define ELF_MACHINE_X86             ((uint16_t)0x03)
#define ELF_MACHINE_MIPS            ((uint16_t)0x08)
#define ELF_MACHINE_POWERPC         ((uint16_t)0x14)
#define ELF_MACHINE_ARM             ((uint16_t)0x28)
#define ELF_MACHINE_SUPERH          ((uint16_t)0x2A)
#define ELF_MACHINE_IA64            ((uint16_t)0x32)
#define ELF_MACHINE_X86_64          ((uint16_t)0x3E)
#define ELF_MACHINE_AVR             ((uint16_t)0x53)
#define ELF_MACHINE_AARCH64         ((uint16_t)0xB7)

#define ELF_VERSION_ORIGINAL        ((uint32_t)1)

#define ELF_SECTION_TYPE_NULL       ((uint32_t)0)
#define ELF_SECTION_TYPE_PROGBITS   ((uint32_t)1)
#define ELF_SECTION_TYPE_SYMTAB     ((uint32_t)2)
#define ELF_SECTION_TYPE_STRTAB     ((uint32_t)3)
#define ELF_SECTION_TYPE_NOBITS     ((uint32_t)8)

//---------------------------------------------------------------------------
typedef struct
{
    // (Explicit line breaks to show 32-bit alignment)
    //---- 0x00
    uint32_t    u32IdentMagicNumber;

    //---- 0x04
    uint8_t     u8IdentFormat;
    uint8_t     u8IdentEndianness;
```

```
    uint8_t     u8IdentVersion;
    uint8_t     u8IdentABI;

    //---- 0x08
    uint8_t     u8IdentABIVersion;
    uint8_t     u8Pad1[7];

    //---- 0x10
    uint16_t    u16Type;
    uint16_t    u16Machine;

    //---- 0x14
    uint32_t    u32Version;

    //---- 0x18
    uint32_t    u32EntryPoint;

    //---- 0x1C
    uint32_t    u32PHOffset;

    //---- 0x20
    uint32_t    u32SHOffset;

    //---- 0x24
    uint32_t    u32Flags;

    //---- 0x28
    uint16_t    u16EHSize;
    uint16_t    u16PHSize;

    //---- 0x2C
    uint16_t    u16PHNum;
    uint16_t    u16SHSize;

    //---- 0x30
    uint16_t    u16SHNum;
    uint16_t    u16SHIndex;

} ElfHeader_t;

//---------------------------------------------------------------------------
typedef struct
{
    uint32_t    u32Type;
    uint32_t    u32Offset;
    uint32_t    u32VirtualAddress;
    uint32_t    u32PhysicalAddress;
    uint32_t    u32FileSize;
    uint32_t    u32MemSize;
    uint32_t    u32Flags;
    uint32_t    u32Alignment;
} ElfProgramHeader_t;

//---------------------------------------------------------------------------
typedef struct
{
    uint32_t    u32Name;
    uint32_t    u32Type;
    uint32_t    u32Flags;
    uint32_t    u32Address;
    uint32_t    u32Offset;
    uint32_t    u32Size;
    uint32_t    u32Link;
    uint32_t    u32Info;
    uint32_t    u32Alignment;
    uint32_t    u32EntrySize;
} ElfSectionHeader_t;

//---------------------------------------------------------------------------
typedef struct
{
    uint32_t    u32Name;
    uint32_t    u32Value;
    uint32_t    u32Size;
    uint8_t     u8Info;
    uint8_t     u8Other;
    uint16_t    u16SHIndex;
} ElfSymbol_t;


#endif //__ELF_TYPES_H__
```

4.71 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.

4.71.1 Detailed Description

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

Definition in file emu_config.h.

4.71.2 Macro Definition Documentation

4.71.2.1 #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.

4.71.2.2 #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.

4.72 emu_config.h

```
00001 /****************************************************************************
00002  *     (     (                       (     |
00003  *    )\ )  )\ )     (              )\ )   |
00004  *   (()/( (()/(    )\      (   (  (()/(   | -- [ Funkenstein ] -------------
00005  *    /(_)) /(_))(((_)()\  )\  /(_))       | -- [ Litle ] -------------------
00006  *   (_))_|(_))   )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
00007  *   | |_  | |    (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
```

```
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |_____| /_/ \_\   \_/   |_|_\   |
 *                                         | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __EMU_CONFIG_H__
#define __EMU_CONFIG_H__

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

#define CONFIG_IO_ADDRESS_BYTES        (256)                     // First bytes of address space are I/O
 range

#define FEATURE_USE_JUMPTABLES          (1)

#define CONFIG_TRACEBUFFER_SIZE        (1000)

#endif

```

4.73 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 "mega_uart.h"
#include "mega_eint.h"
#include "mega_timer16.h"
#include "mega_timer8.h"
#include "mega_eeprom.h"
#include "avr_disasm.h"
#include "trace_buffer.h"
#include "options.h"
#include "interactive.h"
#include "breakpoint.h"
#include "watchpoint.h"
#include "kernel_aware.h"
#include "code_profile.h"
#include "tlv_file.h"
#include "gdb_rsp.h"
```

Enumerations

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

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**

4.73.1 Detailed Description

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

Definition in file flavr.c.

4.74 flavr.c

```
/****************************************************************************
 *     (     (                    (     |
 *    )\ )  )\ )      (           )\ )   |
 *   (()/( (()/(     )\     (   (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
 *   | |_  | |     (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__    / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                        | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#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 "mega_uart.h"
#include "mega_eint.h"
#include "mega_timer16.h"
#include "mega_timer8.h"
#include "mega_eeprom.h"

//---------------------------------------------------------------------------
#include "avr_disasm.h"
#include "trace_buffer.h"
#include "options.h"
#include "interactive.h"
#include "breakpoint.h"
#include "watchpoint.h"
#include "kernel_aware.h"
#include "code_profile.h"
#include "tlv_file.h"
#include "gdb_rsp.h"
```

```
//---------------------------------------------------------------------------
typedef enum
{
    EEPROM_TOO_BIG,
    RAM_TOO_BIG,
    RAM_TOO_SMALL,
    ROM_TOO_BIG,
    INVALID_HEX_FILE,
    INVALID_VARIANT,
    INVALID_DEBUG_OPTIONS
} ErrorReason_t;

//---------------------------------------------------------------------------
static TraceBuffer_t stTraceBuffer;

//---------------------------------------------------------------------------
void splash(void)
{
printf(
       "* ---------------------------------------+-----------------------------------\n"
       "*     (     (                       (     |\n"
       "*    )\\ )  )\\ )    (                )\\ )    |\n"
       "*   (()/( (()/(     )\\      (    (   (()/(   | -- [ Funkenstein ] -------------\n"
       "*    /(_)) /(_))((((_)()\\  )\\   /(_))      | -- [ Litle ] --------------------\n"
       "*   (_))_|(_))   )\\ _ )\\ ((_)((_)(_))      | -- [ AVR ] ----------------------\n"
       "*   | |_  | |    (_)_\\(_)\\ \\ / / | _ \\    | -- [ Virtual ] -----------------\n"
       "*   | __| | |__   / _ \\   \\ V /  |   /    | -- [ Runtime ] -----------------\n"
       "*   |_|   |____| /_/ \\_\\   \\_/   |_|_\\    |\n"
       "*                                        | \"From the makers of Mark3!\"\n"
       "* ---------------------------------------+-----------------------------------\n"
       "* (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved\n"
       "*     See license.txt for details\n"
      );
}

//---------------------------------------------------------------------------
void error_out( ErrorReason_t eReason_ )
{
    switch (eReason_)
    {
        case EEPROM_TOO_BIG:
            printf( "EERPOM Size specified is too large\n" );
            break;
        case RAM_TOO_BIG:
            printf( "RAM Size specified is too large\n" );
            break;
        case RAM_TOO_SMALL:
            printf( "RAM Size specified is too small\n" );
            break;
        case ROM_TOO_BIG:
            printf( "ROM Size specified is too large\n" );
            break;
        case INVALID_HEX_FILE:
            printf( "HEX Programming file cannot be loaded\n");
            break;
        case INVALID_VARIANT:
            printf( "Unknown variant not supported\n");
            break;
        case INVALID_DEBUG_OPTIONS:
            printf( "GDB and built-in interactive debugger are mutually exclusive\n");
        default:
            printf( "Some other reason\n" );
    }

    Options_PrintUsage();

    exit (-1);
}

//---------------------------------------------------------------------------
void emulator_loop(void)
{
    bool bUseTrace = false;
    bool bProfile = false;
    bool bUseGDB = false;

    if ( Options_GetByName("--trace") && Options_GetByName("--debug") )
    {
        bUseTrace = true;
    }

    if ( Options_GetByName("--profile"))
    {
        bProfile = true;
    }

```

```
    if ( Options_GetByName("--gdb"))
    {
        bUseGDB = true;
    }

    while (1)
    {
        // Check to see if we've hit a breakpoint
        if (BreakPoint_EnabledAtAddress(stCPU.u16PC))
        {
            if (bUseGDB)
            {
                GDB_Set();
            }
            else
            {
                Interactive_Set();
            }
        }

        // Check to see if we're in interactive debug mode, and thus need to wait for input
        if (bUseGDB)
        {
            GDB_CheckAndExecute();
        }
        else
        {
            Interactive_CheckAndExecute();
        }

        // Store the current CPU state into the tracebuffer
        if (bUseTrace)
        {
            TraceBuffer_StoreFromCPU(&stTraceBuffer);
        }

        // Run code profiling logic
        if (bProfile)
        {
            Profile_Hit(stCPU.u16PC);
        }

        // Execute a machine cycle
        CPU_RunCycle();
    }
    // doesn't return, except by quitting from debugger, or by signal.
}

//---------------------------------------------------------------------------
void add_plugins(void)
{
    CPU_AddPeriph(&stUART);
    CPU_AddPeriph(&stEINT_a);
    CPU_AddPeriph(&stEINT_b);
    CPU_AddPeriph(&stTimer16);
    CPU_AddPeriph(&stTimer16a);
    CPU_AddPeriph(&stTimer16b);
    CPU_AddPeriph(&stTimer8);
    CPU_AddPeriph(&stTimer8a);
    CPU_AddPeriph(&stTimer8b);
    CPU_AddPeriph(&stEEPROM);
}

//---------------------------------------------------------------------------
void flavr_disasm(void)
{
    uint32_t u32Size;

    u32Size = stCPU.u32ROMSize / sizeof(uint16_t);
    stCPU.u16PC = 0;

    while (stCPU.u16PC < u32Size)
    {
        uint16_t OP = stCPU.pu16ROM[stCPU.u16PC];
        char szBuf[256];

        printf("0x%04X: [0x%04X] ", stCPU.u16PC, OP);
        AVR_Decode(OP);
        AVR_Disasm_Function(OP)(szBuf);
        printf( "%s", szBuf );
        stCPU.u16PC += AVR_Opcode_Size(OP);
    }
    exit(0);
}

//---------------------------------------------------------------------------
void emulator_init(void)
```

```
{
    AVR_CPU_Config_t stConfig;

    // -- Initialize the emulator based on command-line args
    const AVR_Variant_t *pstVariant;

    pstVariant = Variant_GetByName( Options_GetByName("--variant") );
    if (!pstVariant)
    {
        error_out( INVALID_VARIANT );
    }

    if (Options_GetByName("--exitreset"))
    {
        stConfig.bExitOnReset = true;
    }
    else
    {
        stConfig.bExitOnReset = false;
    }

    stConfig.u32EESize  = pstVariant->u32EESize;
    stConfig.u32RAMSize = pstVariant->u32RAMSize;
    stConfig.u32ROMSize = pstVariant->u32ROMSize;

    if (stConfig.u32EESize >= 32768)
    {
        error_out( EEPROM_TOO_BIG );
    }

    if (stConfig.u32RAMSize >= 65535)
    {
        error_out( RAM_TOO_BIG );
    }
    else if (stConfig.u32RAMSize < 256)
    {
        error_out( RAM_TOO_SMALL );
    }

    if (stConfig.u32ROMSize >= (256*1024))
    {
        error_out( ROM_TOO_BIG );
    }

    CPU_Init(&stConfig);

    TraceBuffer_Init( &stTraceBuffer );

    if (Options_GetByName("--hexfile"))
    {
        if (!AVR_Load_HEX( Options_GetByName("--hexfile") ))
        {
            error_out( INVALID_HEX_FILE );
        }
    }
    else if (Options_GetByName("--elffile"))
    {
        if (!AVR_Load_ELF( Options_GetByName("--elffile") ))
        {
            error_out( INVALID_HEX_FILE );
        }
    }
    else
    {
        error_out( INVALID_HEX_FILE );
    }

    if (Options_GetByName("--disasm"))
    {
        // terminates after disassembly is complete
        flavr_disasm();
    }

    if (Options_GetByName("--debug"))
    {
        Interactive_Init( &stTraceBuffer );
    }
    if (Options_GetByName("--gdb"))
    {
        GDB_Init();
    }

    // Only insert a breakpoint/enter interactive debugging mode if specified.
    // Otherwise, start with the emulator running.
    if (Options_GetByName("--debug") && Options_GetByName("--gdb"))
    {
        error_out( INVALID_DEBUG_OPTIONS );
```

```
    }
    if (Options_GetByName("--debug"))
    {
        BreakPoint_Insert( 0 );
    }

    add_plugins();

    if (Options_GetByName("--mark3") || Options_GetByName("--profile"))
    {
        // Initialize tag-length-value code if we're running with code
        // profiling or kernel-aware debugging, since they generate a
        // lot of data that's better stored in a binary format for
        // efficiency.
        TLV_WriteInit( "flavr.tlv" );
    }

    if (Options_GetByName("--mark3"))
    {
        // Mark3 kernel-aware mode should only be enabled on-demand
        KernelAware_Init();
    }

    if (Options_GetByName("--profile"))
    {
        Profile_Init( stConfig.u32ROMSize );
        atexit( Profile_Print );
    }
}

//---------------------------------------------------------------------------
int main( int argc, char **argv )
{

    // Initialize all emulator data
    Options_Init(argc, argv);

    if (!Options_GetByName("--silent"))
    {
        splash();
    }

    emulator_init();

    // Run the emulator/debugger loop.
    emulator_loop();

    return 0;

}
```

4.75 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.

4.75.1 Detailed Description

Module for decoding Intel hex formatted programming files.

Definition in file intel_hex.c.

4.75.2 Function Documentation

4.75.2.1 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.

4.75.2.2 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.

4.76 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 2014-15, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  ****************************************************************************/
00021 #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"

//---------------------------------------------------------------------------
void HEX_Print_Record( HEX_Record_t *stRecord_ )
{
    printf(  "Line: %d\n"
             "ByteCount: %d\n"
             "RecordType: %d\n"
             "Address: %X\n"
             "Data:",
             stRecord_->u32Line,
             stRecord_->u8ByteCount,
             stRecord_->u8RecordType,
             stRecord_->u16Address );
    int i;
    for (i = 0; i < stRecord_->u8ByteCount; i++)
    {
        printf( " %02X", stRecord_->u8Data[i]);
    }
    printf( "\n" );
}

//---------------------------------------------------------------------------
static bool HEX_Read_Header( int fd_ )
{
    ssize_t bytes_read;
    char acBuf[2] = {0};

    bytes_read = read(fd_, acBuf, 1);
    if (1 != bytes_read)
    {
        return false;
    }
    if (':' == acBuf[0])
    {
        return true;
    }
    return false;
}

//---------------------------------------------------------------------------
static bool HEX_Next_Line( int fd_, HEX_Record_t *stRecord_ )
{
    ssize_t bytes_read;
    char acBuf[2] = {0};

    stRecord_->u32Line++;
    do
    {
        bytes_read = read(fd_, acBuf, 1);
        if (1 != bytes_read)
        {
            return false;
        }
    } while(acBuf[0] != '\n');

    return true;
}

//---------------------------------------------------------------------------
static bool HEX_Read_Record_Type( int fd_, HEX_Record_t *stRecord_ )
{
    ssize_t bytes_read;
    uint32_t u32Hex;
    char acBuf[3] = {0};

    bytes_read = read(fd_, acBuf, 2);
    if (2 != bytes_read)
    {
        return false;
    }
    sscanf(acBuf, "%02X", &u32Hex);
    stRecord_->u8RecordType = (uint8_t)u32Hex;

    if (stRecord_->u8RecordType >= RECORD_TYPE_MAX)
    {
        return false;
    }

```

```
    return true;
}

//---------------------------------------------------------------------------
static bool HEX_Read_Byte_Count( int fd_, HEX_Record_t *stRecord_ )
{
    ssize_t bytes_read;
    uint32_t u32Hex;
    char acBuf[3] = {0};

    bytes_read = read(fd_, acBuf, 2);
    if (2 != bytes_read)
    {
        return false;
    }
    sscanf(acBuf, "%02X", &u32Hex);
    stRecord_->u8ByteCount = (uint8_t)u32Hex;

    return true;
}

//---------------------------------------------------------------------------
static bool HEX_Read_Address( int fd_, HEX_Record_t *stRecord_ )
{
    ssize_t bytes_read;
    uint32_t u32Hex;
    char acBuf[5] = {0};

    bytes_read = read(fd_, acBuf, 4);
    if (4 != bytes_read)
    {
        return false;
    }
    sscanf(acBuf, "%04X", &u32Hex);
    stRecord_->u16Address = (uint16_t)u32Hex;

    return true;
}

//---------------------------------------------------------------------------
static bool HEX_Read_Data( int fd_, HEX_Record_t *stRecord_ )
{
    ssize_t bytes_read;
    uint32_t u32Hex;
    char acBuf[MAX_HEX_DATA_BYTES * 2] = {0};

    int i;
    for (i = 0; i < stRecord_->u8ByteCount; i++)
    {
        // printf("i:%d\n", i);
        bytes_read = read(fd_, acBuf, 2);
        if (2 != bytes_read)
        {
            return false;
        }
        sscanf(acBuf, "%02X", &u32Hex);
        stRecord_->u8Data[i] = (uint8_t)u32Hex;
    }

    return true;
}

//---------------------------------------------------------------------------
static bool HEX_Read_Checksum( int fd_, HEX_Record_t *stRecord_ )
{
    ssize_t bytes_read;
    uint32_t u32Hex;
    char acBuf[3] = {0,0,0};

    bytes_read = read(fd_, acBuf, 2);
    if (2 != bytes_read)
    {
        return false;
    }
    sscanf(acBuf, "%02X", &u32Hex);
    stRecord_->u8Checksum = (uint8_t)u32Hex;

    return true;
}

//---------------------------------------------------------------------------
static bool HEX_Line_Validate( HEX_Record_t *stRecord_ )
{
    // Calculate the CRC for the fields in the struct and compare
    // against the value read from file...
    uint8_t u8CRC = 0;
    u8CRC += (uint8_t)(stRecord_->u16Address >> 8);
```

```
    u8CRC += (uint8_t)(stRecord_->u16Address & 0x00FF);
    u8CRC += stRecord_->u8ByteCount;
    u8CRC += stRecord_->u8RecordType;

    uint8_t i;
    for (i = 0; i < stRecord_->u8ByteCount; i++)
    {
        u8CRC += stRecord_->u8Data[i];
    }

    u8CRC = (~u8CRC) + 1;   // Spec says to take the 2's complement
    if (u8CRC != stRecord_->u8Checksum)
    {
        return false;
    }

    return true;
}

//---------------------------------------------------------------------------
bool HEX_Read_Record( int fd_, HEX_Record_t *stRecord_ )
{
    bool rc = true;
    if (rc)
    {
        rc = HEX_Read_Header(fd_);
    }
    if (rc)
    {
        rc = HEX_Read_Byte_Count(fd_, stRecord_);
    }
    if (rc)
    {
        rc = HEX_Read_Address(fd_, stRecord_);
    }
    if (rc)
    {
        rc = HEX_Read_Record_Type(fd_, stRecord_);
    }
    if (rc)
    {
        rc = HEX_Read_Data(fd_, stRecord_);
    }
    if (rc)
    {
        rc = HEX_Read_Checksum(fd_, stRecord_);
    }
    if (rc)
    {
        rc = HEX_Line_Validate(stRecord_);
    }

    HEX_Next_Line(fd_, stRecord_);
    return rc;
}
```

4.77 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.

4.77.1 Detailed Description

Module for decoding Intel hex formatted programming files.

Definition in file intel_hex.h.

4.77.2 Function Documentation

4.77.2.1 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.

4.77.2.2 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.

4.78 intel_hex.h

```
/*****************************************************************************
 *     (     (                       (     |
 *    )\ )  )\ )    (              )\ )   |
 *   (()/(  (()/(    )\     (   (  (()/(   | -- [ Funkenstein ] -------------
```

```
 *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\    |
 *                                          | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __INTEL_HEX_H__
#define __INTEL_HEX_H__

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

//---------------------------------------------------------------------------
// Load a hex file into the ROM section of a virtual AVR.
#define MAX_HEX_DATA_BYTES      (255)    // max data bytes per line in a record

//---------------------------------------------------------------------------
// Record types in the HEX specification
#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)

//---------------------------------------------------------------------------
// For reference, this is the line format for an intel hex record.
// :WWXXYYYYzz.....zzCC
//  Where : = the ":" start code
//  WW = the byte count in the data field
//  XX = the record type
//  YYYY = record address
//  zz = data bytes
//  CC = 2's complement checksum of all fields, excluding start code and checksum

//---------------------------------------------------------------------------
typedef struct
{
    uint8_t  u8ByteCount;
    uint8_t  u8RecordType;
    uint16_t u16Address;
    uint8_t  u8Data[MAX_HEX_DATA_BYTES];
    uint8_t  u8Checksum;
    uint32_t u32Line;
} HEX_Record_t;

//---------------------------------------------------------------------------
void HEX_Print_Record( HEX_Record_t *stRecord_ );

//---------------------------------------------------------------------------
bool HEX_Read_Record( int fd_, HEX_Record_t *stRecord_ );

#endif
```

4.79 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 "debug_sym.h"
#include "write_callout.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_BreakFunc (char ∗szCommand_)

Interactive_BreakFunc.

- static bool Interactive_WatchObj (char *szCommand_)

    Interactive_WatchObj.
- static bool Interactive_ListObj (char *szCommand_)

    Interactive_ListObj.
- static bool Interactive_ListFunc (char *szCommand_)

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

    Interactive_CheckAndExecute.
- void Interactive_Set (void)

    Interactive_Set.
- bool **Interactive_WatchpointCallback** (uint16_t u16Addr_, uint8_t u8Val_)
- 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** []

4.79.1 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.

4.79.2 Typedef Documentation

4.79.2.1 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 46 of file interactive.c.

4.79.3 Function Documentation

4.79.3.1 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 478 of file interactive.c.

4.79.3.2 static bool Interactive_BreakFunc (char * *szCommand_*) [static]

Interactive_BreakFunc.

Toggle a breakpoint at the beginning of a function referenced by name. Requires that the symbol name match a valid debug symbol loaded from an elf binary (i.e., not from a hex file).

Parameters

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

Returns

false - continue interactive debugging

Definition at line 667 of file interactive.c.

4.79.3.3 void Interactive_CheckAndExecute (void)

Interactive_CheckAndExecute.

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

Definition at line 341 of file interactive.c.

4.79.3.4 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 470 of file interactive.c.

4.79.3.5 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 646 of file interactive.c.

4.79.3.6 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 586 of file interactive.c.

4.79.3.7 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 633 of file interactive.c.

4.79.3.8 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 382 of file interactive.c.

4.79.3.9 static bool Interactive_ListFunc (char * *szCommand_*) [static]

Interactive_ListFunc.

Display a list of functions in the symbol table, if the program was read from an ELF file, and contains debug symbols.

Parameters

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

Returns

false - continue interactive debugging

Definition at line 783 of file interactive.c.

4.79.3.10 static bool Interactive_ListObj (char * *szCommand_*) [static]

Interactive_ListObj.

Display a list of objects in the symbol table, if the program was read from an ELF file, and contains debug symbols.

Parameters

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

Returns

false - continue interactive debugging

Definition at line 763 of file interactive.c.

4.79.3.11 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 620 of file interactive.c.

4.79.3.12 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 559 of file interactive.c.

4.79.3.13 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 613 of file interactive.c.

4.79.3.14 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 532 of file interactive.c.

4.79.3.15 void Interactive_Set (void)

Interactive_Set.

Enable interactive-debug mode on the next instruction cycle.

Definition at line 361 of file interactive.c.

4.79.3.16 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 626 of file interactive.c.

4.79.3.17 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 660 of file interactive.c.

4.79.3.18 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 506 of file interactive.c.

4.79.3.19 static bool Interactive_WatchObj (char * *szCommand_*) [static]

Interactive_WatchObj.

Toggle a watchpoint at the beginning of an object referenced by name. Requires that the symbol name match a valid debug symbol loaded from an elf binary (i.e., not from a hex file).

Parameters

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

Returns

false - continue interactive debugging

Definition at line 711 of file interactive.c.

4.79.4 Variable Documentation

4.79.4.1 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 },
    { "lfunc",    "List Functions", Interactive_ListFunc },
    { "help",     "List commands", Interactive_Help },
    { "step",     "Step to next instruction", Interactive_Step },
    { "quit",     "Quit emulator", Interactive_Quit },
    { "lobj",     "List Objects", Interactive_ListObj },
    { "bsym",     "Toggle breakpoint at function referenced by symbol",
```

```
      Interactive_BreakFunc },
    { "wobj",     "Toggle watchpoint on object referenced by symbol",
      Interactive_WatchObj },
    { "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 252 of file interactive.c.

4.80 interactive.c

```
/**************************************************************************
 *     (      (                      (      |
 *    )\ )   )\ )      (            )\ )    |
 *   (()/(  (()/(     )\      (    (  (()/(    | -- [ Funkenstein ] -------------
 *    /(_))  /(_))((((_)()\   )\   /(_))      | -- [ Litle ] -------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ---------------------
 *   | |_  | |     (_)_\(_)\ \ / / | _ \     | -- [ Virtual ] -----------------
 *   | __| | |__    / _ \   \ V /  |   /     | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\     |
 *                                            | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 **************************************************************************/
#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 "debug_sym.h"
#include "write_callout.h"

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

//---------------------------------------------------------------------------
typedef struct
{
    const char *szCommand;
    const char *szDescription;
    Interactive_Handler pfHandler;
} Interactive_Command_t;

//---------------------------------------------------------------------------
static bool bIsInteractive;
static bool bRetrigger;

static TraceBuffer_t *pstTrace = 0;

//---------------------------------------------------------------------------
static bool Interactive_Continue( char *szCommand_ );

//---------------------------------------------------------------------------
static bool Interactive_Step( char *szCommand_ );

//---------------------------------------------------------------------------
static bool Interactive_Break( char *szCommand_ );

//---------------------------------------------------------------------------
static bool Interactive_Watch( char *szCommand_ );

//---------------------------------------------------------------------------
static bool Interactive_ROM( char *szCommand_ );
```

```
//---------------------------------------------------------------------------
static bool Interactive_RAM( char *szCommand_ );

//---------------------------------------------------------------------------
static bool Interactive_EE( char *szCommand_ );

//---------------------------------------------------------------------------
static bool Interactive_Registers( char *szCommand_ );

//---------------------------------------------------------------------------
static bool Interactive_Quit( char *szCommand_ );

//---------------------------------------------------------------------------
static bool Interactive_Help( char *szCommand_ );

//---------------------------------------------------------------------------
static bool Interactive_Disasm( char *szCommand_ );

//---------------------------------------------------------------------------
static bool Interactive_Trace( char *szCommand_ );

//---------------------------------------------------------------------------
static bool Interactive_BreakFunc( char *szCommand_ );

//---------------------------------------------------------------------------
static bool Interactive_WatchObj( char *szCommand_ );

//---------------------------------------------------------------------------
static bool Interactive_ListObj( char *szCommand_ );

//---------------------------------------------------------------------------
static bool Interactive_ListFunc( char *szCommand_ );

//---------------------------------------------------------------------------
// Command-handler table
static Interactive_Command_t astCommands[] =
{
    { "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 },
    { "lfunc",    "List Functions", Interactive_ListFunc },
    { "help",     "List commands", Interactive_Help },
    { "step",     "Step to next instruction", Interactive_Step },
    { "quit",     "Quit emulator", Interactive_Quit },
    { "lobj",     "List Objects", Interactive_ListObj },
    { "bsym",     "Toggle breakpoint at function referenced by symbol", Interactive_BreakFunc },
    { "wobj",     "Toggle watchpoint on object referenced by symbol", Interactive_WatchObj },
    { "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 }
};

//---------------------------------------------------------------------------
static bool Interactive_Execute_i( void )
{
    // Interactive mode - grab a line from standard input.
    char szCmdBuf[256];
    int iCmd = 0;

    printf( "> " );

    // Bail if stdin reaches EOF...
    if (0 == fgets(szCmdBuf, 255, stdin))
    {
        printf("[EOF]\n");
        exit(0);
    }

    iCmd = strlen(szCmdBuf);
    if ( iCmd <= 1 )
    {
```

```
        printf("\n");
        iCmd = 0;
    }
    else
    {
        szCmdBuf[ iCmd - 1 ] = 0;
    }

    // Compare command w/elements in the command table
    Interactive_Command_t *pstCommand = astCommands;
    bool bFound = false;
    bool bContinue = false;

    while (pstCommand->szCommand)
    {
        if ( (0 == strncmp(pstCommand->szCommand, szCmdBuf, strlen(pstCommand->szCommand)))
                && ( szCmdBuf[ strlen(pstCommand->szCommand) ] == ' ' ||
                     szCmdBuf[ strlen(pstCommand->szCommand) ] == '\0' ||
                     szCmdBuf[ strlen(pstCommand->szCommand) ] == '\n' ||
                     szCmdBuf[ strlen(pstCommand->szCommand) ] == '\r' ) )
        {

            // printf( "Found match: %s\n", pstCommand->szCommand );
            bFound = true;
            bContinue = pstCommand->pfHandler( szCmdBuf );
            break;
        }
        // Next command
        pstCommand++;
    }

    if (!bFound)
    {
        printf( "Invalid Command\n");
    }

    return bContinue;
}

//---------------------------------------------------------------------------
void Interactive_CheckAndExecute( void )
{
    // If we're in non-interactive mode (i.e. native execution), then return
    // out instantly.
    if (false == bIsInteractive)
    {
        if (false == bRetrigger)
        {
            return;
        }
        bIsInteractive = true;
        bRetrigger = false;
    }
    printf( "Debugging @ Address [0x%X]\n", stCPU.u16PC );

    // Keep attempting to parse commands until a valid one was encountered
    while (!Interactive_Execute_i()) { /* Do Nothing */ }
}

//---------------------------------------------------------------------------
void Interactive_Set( void )
{
    bIsInteractive = true;
    bRetrigger = false;
}

//---------------------------------------------------------------------------
bool Interactive_WatchpointCallback( uint16_t u16Addr_, uint8_t u8Val_ )
{
    if (WatchPoint_EnabledAtAddress(u16Addr_))
    {
        Interactive_Set();
        printf( "Watchpoint @ 0x%04X hit.  Old Value => %d, New Value => %d\n",
                    u16Addr_,
                    stCPU.pstRAM->au8RAM[ u16Addr_ ],
                    u8Val_ );
    }
    return true;
}

//---------------------------------------------------------------------------
void Interactive_Init( TraceBuffer_t *pstTrace_ )
{
    pstTrace = pstTrace_;
    bIsInteractive = false;
    bRetrigger = false;
```

```
    // Add the watchpoint handler as a wildcard callout (i.e. every write
    // triggers is, it's up to the callout to handle filtering on its own).
    WriteCallout_Add( Interactive_WatchpointCallback, 0 );

}

//---------------------------------------------------------------------------
static bool  Token_ScanNext( char *szCommand_, int iStart_, int *piTokenStart_, int *piTokenLen_)
{
    int i = iStart_;

    // Parse leading whitespace
    while ( (szCommand_[i] == ' ') ||
            (szCommand_[i] == '\t') ||
            (szCommand_[i] == '\r') ||
            (szCommand_[i] == '\n')
            ) { i++; }

    // Check null termination
    if (szCommand_[i] == '\0' )
    {
        return false;
    }

    // Parse token
    *piTokenStart_ = i;
    while ( (szCommand_[i] != ' ') &&
            (szCommand_[i] != '\t') &&
            (szCommand_[i] != '\r') &&
            (szCommand_[i] != '\n') &&
            (szCommand_[i] != '\0')
            ) { i++; }
    *piTokenLen_ = (i - *piTokenStart_);

    // printf( "Start, Len: %d, %d\n", i, *piTokenLen_ );
    return true;
}

//---------------------------------------------------------------------------
static bool Token_DiscardNext( char *szCommand_, int iStart_, int *piNextTokenStart_ )
{
    int iTempStart;
    int iTempLen;
    if (!Token_ScanNext(szCommand_, iStart_, &iTempStart, &iTempLen ))
    {
        return false;
    }
    *piNextTokenStart_ = iTempStart + iTempLen + 1;
    return true;
}

//---------------------------------------------------------------------------
static bool Token_ReadNextHex( char *szCommand_, int iStart_, int *piNextTokenStart_, unsigned int *puiVal_
 )
{
    int iTempStart = iStart_;
    int iTempLen;

    if (!Token_ScanNext(szCommand_, iStart_, &iTempStart, &iTempLen ))
    {
        return false;
    }

    szCommand_[iTempStart + iTempLen] = 0;

    if (0 == sscanf( &szCommand_[iTempStart], "%x", puiVal_ ))
    {
        if (0 == sscanf( &szCommand_[iTempStart], "x%x", puiVal_ ))
        {
            if (0 == sscanf( &szCommand_[iTempStart], "0x%x", puiVal_ ))
            {
                printf( "Missing Argument\n" );
                return false;
            }
        }
    }

    *piNextTokenStart_ = iTempStart + iTempLen + 1;
    return true;
}


//---------------------------------------------------------------------------
static bool Interactive_Continue( char *szCommand_ )
{
    bIsInteractive = false;
```

```
    bRetrigger = false;
    return true;
}

//---------------------------------------------------------------------------
static bool Interactive_Break( char *szCommand_ )
{
    unsigned int uiAddr;
    int iTokenStart;

    if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
    {
        return false;
    }

    if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiAddr))
    {
        return false;
    }

    if (BreakPoint_EnabledAtAddress( (uint16_t)uiAddr))
    {
        BreakPoint_Delete( (uint16_t)uiAddr);
    }
    else
    {
        BreakPoint_Insert( (uint16_t)uiAddr);
    }

    return false;
}

//---------------------------------------------------------------------------
static bool Interactive_Watch( char *szCommand_ )
{
    unsigned int uiAddr;
    int iTokenStart;

    if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
    {
        return false;
    }

    if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiAddr))
    {
        return false;
    }

    if (WatchPoint_EnabledAtAddress((uint16_t)uiAddr))
    {
        WatchPoint_Delete( (uint16_t)uiAddr);
    }
    else
    {
        WatchPoint_Insert( (uint16_t)uiAddr);
    }
    return false;
}
//---------------------------------------------------------------------------
static bool Interactive_ROM( char *szCommand_ )
{
    unsigned int uiAddr;
    unsigned int uiLen;
    int iTokenStart;

    if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
    {
        return false;
    }

    if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiAddr))
    {
        return false;
    }

    if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiLen))
    {
        return false;
    }

    print_rom( (uint16_t)uiAddr, (uint16_t)uiLen );

    return false;
}

//---------------------------------------------------------------------------
static bool Interactive_RAM( char *szCommand_ )
```

```
{
    unsigned int uiAddr;
    unsigned int uiLen;
    int iTokenStart;

    if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
    {
        return false;
    }

    if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiAddr))
    {
        return false;
    }

    if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiLen))
    {
        return false;
    }

    print_ram( (uint16_t)uiAddr, (uint16_t)uiLen );

    return false;
}

//---------------------------------------------------------------------------
static bool Interactive_EE( char *szCommand_ )
{
    unsigned int uiAddr;
    unsigned int uiLen;
    int iTokenStart;

    if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
    {
        return false;
    }

    if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiAddr))
    {
        return false;
    }

    if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiLen))
    {
        return false;
    }

    printf( "Dump EEPROM [%x:%x]\n", uiAddr, uiLen );

    return false;
}

//---------------------------------------------------------------------------
static bool Interactive_Registers( char *szCommand_ )
{
    print_core_regs();
    return false;
}

//---------------------------------------------------------------------------
static bool Interactive_Quit( char *szCommand_ )
{
    exit(0);
}

//---------------------------------------------------------------------------
static bool Interactive_Step( char *szCommand_ )
{
    bRetrigger = true; // retrigger debugging on next loop
    return true;
}

//---------------------------------------------------------------------------
static bool Interactive_Help( char *szCommand_ )
{
    Interactive_Command_t *pstCommand_ = astCommands;
    printf( "FLAVR interactive debugger commands:\n");
    while (pstCommand_->szCommand)
    {
        printf( "    %s: %s\n", pstCommand_->szCommand, pstCommand_->szDescription );
        pstCommand_++;
    }
    return false;
}

//---------------------------------------------------------------------------
```

```
static bool Interactive_Disasm( char *szCommand_ )
{
    char szBuf[256];
    uint16_t OP = stCPU.pu16ROM[stCPU.u16PC];

    printf("0x%04X: [0x%04X] ", stCPU.u16PC, OP);
    AVR_Decode(OP);
    AVR_Disasm_Function(OP)(szBuf);
    printf( "%s", szBuf );

    return false;
}

//---------------------------------------------------------------------------
static bool Interactive_Trace( char *szCommand_ )
{
    TraceBuffer_Print( pstTrace, TRACE_PRINT_COMPACT | TRACE_PRINT_DISASSEMBLY );
    return false;
}

//---------------------------------------------------------------------------
static bool Interactive_BreakFunc( char *szCommand_ )
{
    unsigned int uiAddr;
    unsigned int uiLen;
    int iTokenStart;
    int iEnd;

    if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
    {
        return false;
    }

    if (!Token_ScanNext( szCommand_, iTokenStart, &iEnd, &uiLen ) )
    {
        return false;
    }

    szCommand_[iTokenStart+uiLen] = 0;

    char *szName = &szCommand_[iTokenStart];
    Debug_Symbol_t *pstSym = Symbol_Find_Func_By_Name( szName );

    if (!pstSym)
    {
        printf( "Unknown function: %s", szName );
        return false;
    }
    printf( "Name: %s, Start Addr: %x, End Addr: %x\n", pstSym->szName, pstSym->u32StartAddr, pstSym->u32EndAddr );

    if (BreakPoint_EnabledAtAddress(pstSym->u32StartAddr))
    {
        printf( "Removing breakpoint @ 0x%04X\n", pstSym->u32StartAddr );
        BreakPoint_Delete( pstSym->u32StartAddr );
    }
    else
    {
        printf( "Inserting breakpoint @ 0x%04X\n", pstSym->u32StartAddr );
        BreakPoint_Insert( pstSym->u32StartAddr );
    }

    return false;
}

//---------------------------------------------------------------------------
static bool Interactive_WatchObj( char *szCommand_ )
{
    unsigned int uiAddr;
    unsigned int uiLen;
    int iTokenStart;
    int iEnd;

    if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
    {
        return false;
    }

    if (!Token_ScanNext( szCommand_, iTokenStart, &iEnd, &uiLen ) )
    {
        return false;
    }

    szCommand_[iTokenStart+uiLen] = 0;

    char *szName = &szCommand_[iTokenStart];
```

```
    Debug_Symbol_t *pstSym = Symbol_Find_Obj_By_Name( szName );

    if (!pstSym)
    {
        printf( "Unknown object: %s", szName );
        return false;
    }
    printf( "Name: %s, Start Addr: %x, End Addr: %x\n", pstSym->szName, pstSym->u32StartAddr, pstSym->u32EndAddr );

    if (WatchPoint_EnabledAtAddress(pstSym->u32StartAddr))
    {
        printf( "Removing watchpoint @ 0x%04X\n", pstSym->u32StartAddr );
        uint32_t i;
        for (i = pstSym->u32StartAddr; i <= pstSym->u32EndAddr; i++)
        {
            WatchPoint_Delete( i );
        }
    }
    else
    {
        printf( "Inserting watchpoint @ 0x%04X\n", pstSym->u32StartAddr );
        uint32_t i;
        for (i = pstSym->u32StartAddr; i <= pstSym->u32EndAddr; i++)
        {
            WatchPoint_Insert( i );
        }
    }

    return false;
}

//---------------------------------------------------------------------------
static bool Interactive_ListObj( char *szCommand_ )
{
    uint32_t u32Count = Symbol_Get_Obj_Count();
    uint32_t i;
    printf( "Listing objects:\n" );
    for (i = 0; i < u32Count; i++)
    {
        Debug_Symbol_t *pstSymbol = Symbol_Obj_At_Index(i);
        if (!pstSymbol)
        {
            break;
        }

        printf( "%d: %s\n", i, pstSymbol->szName );
    }
    printf( "  done\n" );
    return false;
}

//---------------------------------------------------------------------------
static bool Interactive_ListFunc( char *szCommand_ )
{
    uint32_t u32Count = Symbol_Get_Func_Count();
    uint32_t i;
    printf( "Listing functions:\n" );
    for (i = 0; i < u32Count; i++)
    {
        Debug_Symbol_t *pstSymbol = Symbol_Func_At_Index(i);
        if (!pstSymbol)
        {
            break;
        }

        printf( "%d: %s\n", i, pstSymbol->szName );
    }
    printf( "  done\n" );
    return false;
}
```

4.81 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.

4.81.1 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.h.

4.81.2 Function Documentation

4.81.2.1 void Interactive_CheckAndExecute (void)

Interactive_CheckAndExecute.

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

Definition at line 341 of file interactive.c.

4.81.2.2 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 382 of file interactive.c.

4.81.2.3 void Interactive_Set (void)

Interactive_Set.

Enable interactive-debug mode on the next instruction cycle.

Definition at line 361 of file interactive.c.

4.82 interactive.h

```
/*****************************************************************************
 *     (      (                    (     |
 *    )\ )   )\ )     (              )\ )   |
 *   (()/(  (()/(     )\      (    (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_))  /(_))((((_)()\   )\   /(_))      | -- [ Litle ] -------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))       | -- [ AVR ] ---------------------
 *   | |_   | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *   | __|  | |__    / _ \   \ V /  |   /     | -- [ Runtime ] -----------------
 *   |_|    |____|  /_/ \_\   \_/   |_|_\    |
 *                                          | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
```

```
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __INTERACTIVE_H__
#define __INTERACTIVE_H__

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

//---------------------------------------------------------------------------
void Interactive_CheckAndExecute( void );

//---------------------------------------------------------------------------
void Interactive_Set( void );

//---------------------------------------------------------------------------
void Interactive_Init( TraceBuffer_t *pstTrace_);

#endif
```

4.83 interrupt_callout.c File Reference

Module providing functionality allowing emulator extensions to be triggered on interrupts.

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

Data Structures

- struct Interrupt_Callout_

Typedefs

- typedef struct Interrupt_Callout_ **Interrupt_Callout_t**

Functions

- void InterruptCallout_Add (InterruptCalloutFunc pfCallout_)

  InterruptCallout_Add.
- void InterruptCallout_Run (bool bEntry_, uint8_t u8Vector_)

  InterruptCallout_Run.

Variables

- static Interrupt_Callout_t ∗ **pstCallouts** = 0

4.83.1 Detailed Description

Module providing functionality allowing emulator extensions to be triggered on interrupts.

Definition in file interrupt_callout.c.

4.83.2 Function Documentation

4.83.2.1 void InterruptCallout_Add (InterruptCalloutFunc *pfCallout_*)

InterruptCallout_Add.

Add a particular callout function to be executed whenever an interrupt is called (or returned-from).

Parameters

*pfCallout_*	Pointer to an interrupt callout function.

Definition at line 39 of file interrupt_callout.c.

4.83.2.2 void InterruptCallout_Run (bool *bEntry_*, uint8_t *u8Vector_*)

InterruptCallout_Run.

Run all interrupt callouts currently installed.

Parameters

*bEntry_*	true - interrupt entry, false - interrupt exit
*u8Vector_*	Interrupt vector # (undefined for interrupt-exit)

Definition at line 50 of file interrupt_callout.c.

4.84 interrupt_callout.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 2014-15, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  ****************************************************************************/
00022 #include "interrupt_callout.h"
00023 
00024 #include <stdint.h>
00025 #include <stdio.h>
00026 #include <stdlib.h>
00027 #include <string.h>
00028 //---------------------------------------------------------------------------
00029 typedef struct Interrupt_Callout_
00030 {
00031     struct Interrupt_Callout_ *pstNext;
00032     InterruptCalloutFunc  pfCallout;
00033 } Interrupt_Callout_t;
00034 
00035 //---------------------------------------------------------------------------
00036 static Interrupt_Callout_t *pstCallouts = 0;
00037 
00038 //---------------------------------------------------------------------------
00039 void InterruptCallout_Add( InterruptCalloutFunc pfCallout_ )
00040 {
00041     Interrupt_Callout_t *pstNewCallout = (Interrupt_Callout_t*)(
      malloc(sizeof(*pstNewCallout)));
00042 
00043     pstNewCallout->pstNext = pstCallouts;
00044     pstNewCallout->pfCallout = pfCallout_;
00045 
00046     pstCallouts = pstNewCallout;
00047 }
00048 
00049 //---------------------------------------------------------------------------
00050 void InterruptCallout_Run( bool bEntry_, uint8_t u8Vector_ )
00051 {
```

```
    Interrupt_Callout_t *pstCallout = pstCallouts;
    while (pstCallout)
    {
        pstCallout->pfCallout( bEntry_, u8Vector_ );
        pstCallout = pstCallout->pstNext;
    }
}
```

4.85 interrupt_callout.h File Reference

Module providing functionality allowing emulator extensions to be triggered on interrupts.

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

Typedefs

- typedef void(∗ InterruptCalloutFunc)(bool bEntry_, uint8_t u8Vector_)

  Function type used for interrupt callouts.

Functions

- void InterruptCallout_Add (InterruptCalloutFunc pfCallout_)

  InterruptCallout_Add.
- void InterruptCallout_Run (bool bEntry_, uint8_t u8Vector_)

  InterruptCallout_Run.

4.85.1 Detailed Description

Module providing functionality allowing emulator extensions to be triggered on interrupts.

Definition in file interrupt_callout.h.

4.85.2 Function Documentation

4.85.2.1 void InterruptCallout_Add (InterruptCalloutFunc *pfCallout_*)

InterruptCallout_Add.

Add a particular callout function to be executed whenever an interrupt is called (or returned-from).

Parameters

*pfCallout_*	Pointer to an interrupt callout function.

Definition at line 39 of file interrupt_callout.c.

4.85.2.2 void InterruptCallout_Run (bool *bEntry_*, uint8_t *u8Vector_*)

InterruptCallout_Run.

Run all interrupt callouts currently installed.

Parameters

*bEntry_*	true - interrupt entry, false - interrupt exit
*u8Vector_*	Interrupt vector # (undefined for interrupt-exit)

Definition at line 50 of file interrupt_callout.c.

4.86 interrupt_callout.h

```
/*****************************************************************************
 *     (     (                      (    |
 *    )\ )  )\ )    (              )\ )  |
 *   (()/( (()/(    )\     (   (  (()/(  | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))    | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))   | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \  | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /  | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\  |
 *                                       | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 *****************************************************************************/
#ifndef __INTERRUPT_CALLOUT_H__
#define __INTERRUPT_CALLOUT_H__

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

//---------------------------------------------------------------------------
typedef void (*InterruptCalloutFunc)( bool bEntry_, uint8_t u8Vector_ );

//---------------------------------------------------------------------------
void InterruptCallout_Add( InterruptCalloutFunc pfCallout_ );

//---------------------------------------------------------------------------
void InterruptCallout_Run( bool bEntry_, uint8_t u8Vector_ );


#endif

```

4.87 ka_graphics.c File Reference

Mark3 RTOS Kernel-Aware graphics library.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdint.h>
#include <SDL/SDL.h>
#include "kernel_aware.h"
#include "debug_sym.h"
#include "write_callout.h"
#include "interrupt_callout.h"
```

Data Structures

- struct DrawPoint_t

Macros

- #define **GFX_RES_X** (128)
- #define **GFX_RES_Y** (160)

- #define **GFX_SCALE** (3)

Functions

- void **KA_Graphics_Close** (void)
- void **KA_Graphics_ClearScreen** (void)
- void **KA_Graphics_DrawPoint** (DrawPoint_t ∗pstPoint_)
- void **KA_Graphics_Flip** (void)
- bool **KA_Graphics_Command** (uint16_t u16Addr_, uint8_t u8Data_)
- void **KA_Graphics_Init** (void)

Variables

- static SDL_Surface ∗ **pstScreen** = 0

4.87.1 Detailed Description

Mark3 RTOS Kernel-Aware graphics library.

Definition in file ka_graphics.c.

4.88 ka_graphics.c

```
/*****************************************************************************
 *     (     (                      (    |
 *    )\ )  )\ )    (             )\ )  |
 *   (()/( (()/(    )\     (   ( (()/(  | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))   | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))  | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \ | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   / | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\ |
 *                                      | "Yeah, it does Arduino..."
 * -------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <stdint.h>
#include <SDL/SDL.h>

#include "kernel_aware.h"
#include "debug_sym.h"
#include "write_callout.h"
#include "interrupt_callout.h"

//---------------------------------------------------------------------------
#define GFX_RES_X       (128)
#define GFX_RES_Y       (160)
#define GFX_SCALE       (3)

//---------------------------------------------------------------------------
typedef struct
{
    uint16_t usX;
    uint16_t usY;
    uint32_t uColor;
} DrawPoint_t;

//---------------------------------------------------------------------------
static SDL_Surface *pstScreen = 0;

//---------------------------------------------------------------------------
void KA_Graphics_Close(void)
{
    if (pstScreen)
    {
```

```
        SDL_FreeSurface(pstScreen);
    }
    SDL_Quit();
}

//---------------------------------------------------------------------------
void KA_Graphics_ClearScreen(void)
{
    memset( pstScreen->pixels, 0, sizeof(uint16_t) * (GFX_RES_X*GFX_SCALE) * (GFX_RES_Y*GFX_SCALE) );
}

//---------------------------------------------------------------------------
void KA_Graphics_DrawPoint(DrawPoint_t *pstPoint_)
{
    uint32_t *pixels = (uint32_t*)pstScreen->pixels;

   // printf( "X:%d Y:%d C=%08X\n", pstPoint_->usX, pstPoint_->usY, pstPoint_->uColor );
    if ((pstPoint_->usX < GFX_RES_X ) && (pstPoint_->usY < GFX_RES_Y))
    {
        int i,j;
        for (i = 0; i < GFX_SCALE; i++)
        {
            for (j = 0; j < GFX_SCALE; j++)
            {
                pixels[ ((uint32_t)((pstPoint_->usY*GFX_SCALE)+i) * (GFX_RES_X*GFX_SCALE) ) +
                         (uint32_t)((pstPoint_->usX*GFX_SCALE)+j) ] = (uint32_t)pstPoint_->uColor;
            }
        }
    }
}

//---------------------------------------------------------------------------
void KA_Graphics_Flip(void)
{
    if (pstScreen)
    {
        SDL_Flip(pstScreen);
    }
}

//---------------------------------------------------------------------------
bool KA_Graphics_Command( uint16_t u16Addr_, uint8_t u8Data_ )
{
    Debug_Symbol_t *pstSymbol = Symbol_Find_Obj_By_Name( "g_pclPoint" );

    switch( u8Data_ )
    {
        case 1:
            if (pstSymbol)
            {
                uint16_t u16PointAddr = *(uint16_t*)(&stCPU.pstRAM->au8RAM[ pstSymbol->u32StartAddr ]);
                DrawPoint_t *pstPoint = (DrawPoint_t*)(&stCPU.pstRAM->au8RAM[ u16PointAddr ]);
                KA_Graphics_DrawPoint( pstPoint );
            }
            break;
        case 2:
            KA_Graphics_Flip();
            break;
        case 0:
        default:
            break;
    }

    return true;
}

//---------------------------------------------------------------------------
void KA_Graphics_Init(void)
{
    Debug_Symbol_t *pstSymbol = 0;
    pstSymbol = Symbol_Find_Obj_By_Name( "g_u8GfxCommand" );

    // Use pstSymbol's address to get a pointer to the current thread.
    if (!pstSymbol)
    {
        fprintf(stderr, "Kernel-aware graphics driver not found\n" );
        return;
    }

    // Ensure that we actually have the information we need at a valid address
    uint16_t u16CurrPtr = (uint16_t)(pstSymbol->u32StartAddr & 0x0000FFFF);
    if (!u16CurrPtr)
    {
```

```
        fprintf(stderr, "Invalid address for graphics driver global\n" );
        return;
    }

    // Add a callback so that when g_pstCurrent changes, we can update our
    // locally-tracked statistics.
    WriteCallout_Add( KA_Graphics_Command, u16CurrPtr );

    SDL_Init( SDL_INIT_EVERYTHING );
    pstScreen = SDL_SetVideoMode( GFX_RES_X * GFX_SCALE, GFX_RES_Y * GFX_SCALE, 32, SDL_SWSURFACE);
    fprintf(stderr, "Kernel-Aware Graphics Installed\n");

    atexit( KA_Graphics_Close );

}
```

4.89 ka_graphics.h File Reference

Mark3 RTOS Kernel-Aware graphics library.

`#include "kernel_aware.h"`

4.89.1 Detailed Description

Mark3 RTOS Kernel-Aware graphics library.

Definition in file ka_graphics.h.

4.90 ka_graphics.h

```
/*****************************************************************************
 *     (       (                          (      |
 *    )\ )   )\ )    (                  )\ )    |
 *   (()/(  (()/(    )\      (    (   (()/(    | -- [ Funkenstein ] -------------
 *    /(_))  /(_))((((_)()\  )\   /(_))      | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\    |
 *                                          | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __KA_TRACE__
#define __KA_TRACE__

#include "kernel_aware.h"
//---------------------------------------------------------------------------
//void KA_Graphics_Init( void );

#endif

```

4.91 ka_interrupt.c File Reference

Mark3 RTOS Kernel-Aware Interrupt Logging.

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "avr_cpu.h"
#include "kernel_aware.h"
#include "ka_interrupt.h"
#include "write_callout.h"
#include "interrupt_callout.h"
#include "tlv_file.h"
```

Data Structures

- struct Mark3Interrupt_TLV_t

Functions

- static void **KA_Interrupt** (bool bEntry_, uint8_t u8Vector_)
- void KA_Interrupt_Init (void)

  KA_Interrupt_Init.

Variables

- static TLV_t ∗ **pstTLV** = NULL

4.91.1 Detailed Description

Mark3 RTOS Kernel-Aware Interrupt Logging.

Definition in file ka_interrupt.c.

4.91.2 Function Documentation

4.91.2.1 void KA_Interrupt_Init (void)

KA_Interrupt_Init.

Initialize the kernel-aware interrupt logging functionality in the emulator

Definition at line 59 of file ka_interrupt.c.

4.92 ka_interrupt.c

```
/****************************************************************************
 *     (     (                        (     |
 *    )\ )  )\ )     (               )\ )   |
 *   (()/( (()/(    )\      (    (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                         | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
```

```
 ===========================================================================*/
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "avr_cpu.h"
#include "kernel_aware.h"
#include "ka_interrupt.h"
#include "write_callout.h"
#include "interrupt_callout.h"
#include "tlv_file.h"

//---------------------------------------------------------------------------
static TLV_t *pstTLV = NULL;

//---------------------------------------------------------------------------
typedef struct
{
    uint64_t    u64TimeStamp;
    uint8_t     u8Vector;
    bool        bEntry;

} Mark3Interrupt_TLV_t;

//---------------------------------------------------------------------------
static void KA_Interrupt( bool bEntry_, uint8_t u8Vector_ )
{
    Mark3Interrupt_TLV_t stData;
    stData.u64TimeStamp = stCPU.u64CycleCount;
    stData.u8Vector = u8Vector_;
    stData.bEntry = bEntry_;

    memcpy( &(pstTLV->au8Data[0]), &stData, sizeof(stData) );
    TLV_Write(pstTLV);
}

//---------------------------------------------------------------------------
void KA_Interrupt_Init(void)
{
    pstTLV = TLV_Alloc( sizeof(Mark3Interrupt_TLV_t) );
    if (!pstTLV)
    {
        return;
    }

    pstTLV->eTag = TAG_KERNEL_AWARE_INTERRUPT;
    pstTLV->u16Len = sizeof(Mark3Interrupt_TLV_t);

    InterruptCallout_Add( KA_Interrupt );
}
```

4.93 ka_interrupt.h File Reference

Mark3 RTOS Kernel-Aware Interrupt Logging.

Functions

- void KA_Interrupt_Init (void)

  KA_Interrupt_Init.

4.93.1 Detailed Description

Mark3 RTOS Kernel-Aware Interrupt Logging.

Definition in file ka_interrupt.h.

4.93.2 Function Documentation

4.93.2.1 void KA_Interrupt_Init (void)

KA_Interrupt_Init.

Initialize the kernel-aware interrupt logging functionality in the emulator

Definition at line 59 of file ka_interrupt.c.

4.94 ka_interrupt.h

```
/***************************************************************************
 *      (      (                       (      |
 *     )\ )   )\ )     (              )\ )    |
 *    (()/(  (()/(     )\      (    (  (()/(   | -- [ Funkenstein ] -------------
 *     /(_))  /(_))((((_)()\   )\   /(_))      | -- [ Litle ] -------------------
 *    (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ---------------------
 *    | |_  | |     (_)_\(_)\ \ / / | _ \     | -- [ Virtual ] -----------------
 *    | __| | |__    / _ \   \ V /  |   /     | -- [ Runtime ] -----------------
 *    |_|   |____|  /_/ \_\   \_/   |_|_\     |
 *                                             | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __KA_INTERRUPT_H__
#define __KA_INTERRUPT_H__

//---------------------------------------------------------------------------
void KA_Interrupt_Init(void);

#endif
```

4.95 ka_joystick.c File Reference

Mark3 RTOS Kernel-Aware graphics library.

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <SDL/SDL.h>
#include "ka_joystick.h"
#include "write_callout.h"
#include "debug_sym.h"
#include "avr_cpu.h"
```

Macros

- #define **FLAVR_JOY_UP** 0x01
- #define **FLAVR_JOY_DOWN** 0x02
- #define **FLAVR_JOY_LEFT** 0x04
- #define **FLAVR_JOY_RIGHT** 0x08
- #define **FLAVR_JOY_FIRE** 0x10

Functions

- static bool **KA_Scan_Joystick** (uint16_t u16Addr_, uint8_t u8Data_)
- void **KA_Joystick_Init** (void)

Variables

- static uint8_t **u8Val** = 0

4.95.1 Detailed Description

Mark3 RTOS Kernel-Aware graphics library.

Definition in file ka_joystick.c.

4.96 ka_joystick.c

```
/****************************************************************************
 *     (     (                      (     |
 *    )\ )  )\ )    (              )\ )   |
 *   (()/( (()/(    )\     (   (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))    | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                        | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#include <SDL/SDL.h>

#include "ka_joystick.h"
#include "write_callout.h"
#include "debug_sym.h"
#include "avr_cpu.h"

//---------------------------------------------------------------------------
#define FLAVR_JOY_UP        0x01
#define FLAVR_JOY_DOWN      0x02
#define FLAVR_JOY_LEFT      0x04
#define FLAVR_JOY_RIGHT     0x08
#define FLAVR_JOY_FIRE      0x10

//---------------------------------------------------------------------------
static uint8_t u8Val = 0;

//---------------------------------------------------------------------------
static bool KA_Scan_Joystick( uint16_t u16Addr_, uint8_t u8Data_ )
{
    Debug_Symbol_t *pstSymbol = 0;
    pstSymbol = Symbol_Find_Obj_By_Name( "g_u8FlavrJoy" );

    if (!pstSymbol)
    {
        fprintf(stderr, "Invalid joystick scan register\n");
        return true;
    }

    uint16_t u16Addr = (uint16_t)(pstSymbol->u32StartAddr & 0x0000FFFF);

    SDL_Event stEvent;

    while (SDL_PollEvent(&stEvent))
    {
        switch (stEvent.type)
        {
            case SDL_KEYDOWN:
            {
                switch( stEvent.key.keysym.sym )
                {
                    case SDLK_UP:
                        u8Val |= FLAVR_JOY_UP;
                        break;
                    case SDLK_DOWN:
                        u8Val |= FLAVR_JOY_DOWN;
                        break;
```

```
                    case SDLK_LEFT:
                        u8Val |= FLAVR_JOY_LEFT;
                        break;
                    case SDLK_RIGHT:
                        u8Val |= FLAVR_JOY_RIGHT;
                        break;
                    case SDLK_a:
                        u8Val |= FLAVR_JOY_FIRE;
                        break;
                    case SDLK_ESCAPE:
                        exit(0);
                        break;
                    default:
                        break;
                }
            }
                break;
            case SDL_KEYUP:
            {
                switch( stEvent.key.keysym.sym )
                {
                    case SDLK_UP:
                        u8Val &= ~FLAVR_JOY_UP;
                        break;
                    case SDLK_DOWN:
                        u8Val &= ~FLAVR_JOY_DOWN;
                        break;
                    case SDLK_LEFT:
                        u8Val &= ~FLAVR_JOY_LEFT;
                        break;
                    case SDLK_RIGHT:
                        u8Val &= ~FLAVR_JOY_RIGHT;
                        break;
                    case SDLK_a:
                        u8Val &= ~FLAVR_JOY_FIRE;
                        break;
                    default:
                        break;
                }
            }
                break;
            default:
                break;
        }
    }

    stCPU.pstRAM->au8RAM[ u16Addr ] = u8Val;

    return true;
}

//---------------------------------------------------------------------------
void KA_Joystick_Init( void )
{
    Debug_Symbol_t *pstSymbol = 0;
    pstSymbol = Symbol_Find_Obj_By_Name( "g_u8FlavrJoyUp" );

    if (!pstSymbol)
    {
        fprintf(stderr, "Kernel-aware joystick driver not found\n" );
        return;
    }

    // Ensure that we actually have the information we need at a valid address
    uint16_t u16CurrPtr = (uint16_t)(pstSymbol->u32StartAddr & 0x0000FFFF);
    if (!u16CurrPtr)
    {
        fprintf(stderr, "Invalid address for joystick driver global\n" );
        return;
    }

    // Add a callback so that when a joystick scan is requested, we parse keyboard input
    WriteCallout_Add( KA_Scan_Joystick, u16CurrPtr );

}
```

4.97 ka_joystick.h File Reference

Mark3 RTOS Kernel-Aware graphics library.

```
#include "kernel_aware.h"
```

4.97.1 Detailed Description

Mark3 RTOS Kernel-Aware graphics library.

Definition in file ka_joystick.h.

4.98 ka_joystick.h

```
/***************************************************************************
 *     (     (                      (    |
 *    )\ )  )\ )    (              )\ )  |
 *   (()/( (()/(    )\     (   (  (()/(  | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))    | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))   | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \  | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /  | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\  |
 *                                       | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ***************************************************************************/
#ifndef __KA_JOYSTICK_H__
#define __KA_JOYSTICK_H__

#include "kernel_aware.h"
//void KA_Joystick_Init( void );

#endif // __KA_JOYSTICK_H__
```

4.99 ka_profile.c File Reference

Mark3 RTOS Kernel-Aware Profilng.

```
#include "kernel_aware.h"
#include "debug_sym.h"
#include "write_callout.h"
#include "interrupt_callout.h"
#include "ka_profile.h"
#include "tlv_file.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
```

Data Structures

- struct Mark3Profile_TLV_t

Functions

- static void **KA_PrintProfileResults** (void)
- void KA_Command_Profile_Begin (void)

  KA_Command_Profle_Begin.
- void KA_Command_Profile_Start (void)

  KA_Command_Profile_Start.
- void KA_Command_Profile_Stop (void)

  KA_Command_Profile_Stop.

- void KA_Command_Profile_Report (void)

  KA_Command_Profile_Report.
- void KA_Profile_Init (void)

  KA_Profile_Init.

Variables

- static uint64_t u64ProfileEpochStart = 0
- static uint64_t **u64ProfileTotal** = 0
- static uint64_t **u64ProfileCount** = 0
- static char **szNameBuffer** [32] = {}
- static TLV_t ∗ **pstTLV** = NULL

4.99.1 Detailed Description

Mark3 RTOS Kernel-Aware Profilng.

Definition in file ka_profile.c.

4.99.2 Function Documentation

4.99.2.1 void KA_Profile_Init (void)

KA_Profile_Init.

Initialize the kernel-aware profiling code.

Definition at line 120 of file ka_profile.c.

4.99.3 Variable Documentation

4.99.3.1 uint64_t u64ProfileEpochStart = 0 [static]

! This is all singleton data... could be better hosted in a struct... ! Especially if Mark3 ever supports multiple concurrent Profilers

Definition at line 37 of file ka_profile.c.

4.100 ka_profile.c

```
/***************************************************************************
 *     (     (                      (    |
 *    )\ )  )\ )    (              )\ )  |
 *   (()/( (()/(    )\     (   (  (()/(  | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))   | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))   | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \  | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /  | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\  |
 *                                       | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#include "kernel_aware.h"
#include "debug_sym.h"
#include "write_callout.h"
#include "interrupt_callout.h"
#include "ka_profile.h"
#include "tlv_file.h"

```

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

//---------------------------------------------------------------------------
static uint64_t u64ProfileEpochStart = 0;
static uint64_t u64ProfileTotal = 0;
static uint64_t u64ProfileCount = 0;
static char szNameBuffer[32] = {};
static TLV_t *pstTLV = NULL;

//---------------------------------------------------------------------------
typedef struct
{
    uint64_t u64Timestamp;
    uint64_t u64ProfileCount;
    uint64_t u64ProfileTotalCycles;
    char     szName[32];
} Mark3Profile_TLV_t;

//---------------------------------------------------------------------------
static void KA_PrintProfileResults(void)
{
    Mark3Profile_TLV_t stTLV;

    stTLV.u64ProfileCount       = u64ProfileCount;
    stTLV.u64ProfileTotalCycles = u64ProfileTotal;
    stTLV.u64Timestamp          = stCPU.u64CycleCount;

    strcpy( stTLV.szName, szNameBuffer );
    memcpy( pstTLV->au8Data, &stTLV, sizeof(Mark3Profile_TLV_t) );

    printf( "%s: %llu, %llu, %llu\n", stTLV.szName, stTLV.u64Timestamp, stTLV.u64ProfileCount, stTLV.u64ProfileTotalCycles );

    TLV_Write( pstTLV );
}

//---------------------------------------------------------------------------
void KA_Command_Profile_Begin(void)
{
    u64ProfileCount = 0;
    u64ProfileTotal = 0;
    u64ProfileEpochStart = 0;

    Debug_Symbol_t *pstSymbol = Symbol_Find_Obj_By_Name( "g_stKAData" );
    if (!pstSymbol)
    {
        return;
    }

    uint16_t u16NamePtr = *((uint16_t*)&stCPU.pstRAM->au8RAM[ pstSymbol->u32StartAddr ]);
    const char *szName = (const char*)&stCPU.pstRAM->au8RAM[ u16NamePtr ];
    if (szName)
    {
        strcpy( szNameBuffer, szName );
    }
    else
    {
        strcpy( szNameBuffer, "(NONE)" );
    }

}

//---------------------------------------------------------------------------
void KA_Command_Profile_Start(void)
{
    // Profile stop or reset
    u64ProfileEpochStart = stCPU.u64CycleCount;
}

//---------------------------------------------------------------------------
void KA_Command_Profile_Stop(void)
{
    u64ProfileTotal += (stCPU.u64CycleCount - u64ProfileEpochStart);
    u64ProfileEpochStart = 0;
    u64ProfileCount++;

}

//---------------------------------------------------------------------------
void KA_Command_Profile_Report(void)
{
```

```
00114     KA_PrintProfileResults();
00115     u64ProfileTotal = 0;
00116     u64ProfileEpochStart = 0;
00117 }
00118
00119 //---------------------------------------------------------------------------
00120 void KA_Profile_Init(void)
00121 {
00122     pstTLV = TLV_Alloc(sizeof(Mark3Profile_TLV_t));
00123     pstTLV->eTag = TAG_KERNEL_AWARE_PROFILE;
00124     pstTLV->u16Len = sizeof(Mark3Profile_TLV_t);
00125 }
```

4.101 ka_profile.h File Reference

Mark3 RTOS Kernel-Aware Profilng.

Functions

- void KA_Profile_Init (void)

    KA_Profile_Init.
- void KA_Command_Profile_Begin (void)

    KA_Command_Profle_Begin.
- void KA_Command_Profile_Start (void)

    KA_Command_Profile_Start.
- void KA_Command_Profile_Stop (void)

    KA_Command_Profile_Stop.
- void KA_Command_Profile_Report (void)

    KA_Command_Profile_Report.

4.101.1 Detailed Description

Mark3 RTOS Kernel-Aware Profilng.

Definition in file ka_profile.h.

4.101.2 Function Documentation

4.101.2.1 void KA_Profile_Init (void)

KA_Profile_Init.

Initialize the kernel-aware profiling code.

Definition at line 120 of file ka_profile.c.

4.102 ka_profile.h

```
/*****************************************************************************
 *     (      (                        (      |
 *    )\ )   )\ )     (               )\ )    |
 *   (()/(  (()/(    )\      (    (  (()/(    | -- [ Funkenstein ] -------------
 *    /(_))  /(_))((((_)()\  )\  /(_))        | -- [ Litle ] -------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))       | -- [ AVR ] ---------------------
 *   | |_  | |     (_)_\(_)\ \ / / | _ \      | -- [ Virtual ] -----------------
 *   | __| | |__    / _ \   \ V /  |   /      | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\      |
 *                                            | "Yeah, it does Arduino..."
 * -------------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
```

```
 *     See license.txt for details
 ****************************************************************************/
#ifndef __KA_PROFILE_H__
#define __KA_PROFILE_H__

//---------------------------------------------------------------------------
void KA_Profile_Init(void);

//---------------------------------------------------------------------------
void KA_Command_Profile_Begin(void);

//---------------------------------------------------------------------------
void KA_Command_Profile_Start(void);

//---------------------------------------------------------------------------
void KA_Command_Profile_Stop(void);

//---------------------------------------------------------------------------
void KA_Command_Profile_Report(void);

#endif
```

4.103 ka_thread.c File Reference

Mark3 RTOS Kernel-Aware Thread Profiling.

```
#include "kernel_aware.h"
#include "debug_sym.h"
#include "write_callout.h"
#include "interrupt_callout.h"
#include "tlv_file.h"
#include "ka_thread.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
```

Data Structures

- struct Mark3_Thread_t
- struct Mark3_Thread_Info_t
- struct Mark3ContextSwitch_TLV_t

Macros

- #define **THREAD_STATE_EXIT** 0
- #define **THREAD_STATE_READY** 1
- #define **THREAD_STATE_BLOCKED** 2
- #define **THREAD_STATE_STOP** 3

Functions

- static void **Mark3KA_AddKnownThread** (Mark3_Thread_t ∗pstThread_)
- Mark3_Thread_t ∗ **Mark3KA_GetCurrentThread** (void)
- static uint8_t **Mark3KA_GetCurrentPriority** (void)
- static uint16_t **Mark3KA_GetStackMargin** (Mark3_Thread_t ∗pstThread_)
- static uint16_t **Mark3KA_GetCurrentStackMargin** (void)
- static bool **KA_StackWarning** (uint16_t u16Addr_, uint8_t u8Data_)
- static bool **KA_ThreadChange** (uint16_t u16Addr_, uint8_t u8Data_)

- void **KA_PrintThreadInfo** (void)
- void **KA_Thread_Init** (void)
- char ∗ **KA_Get_Thread_Info_XML** (uint8_t ∗∗thread_ids, uint16_t ∗thread_count)
- Mark3_Context_t ∗ **KA_Get_Thread_Context** (uint8_t id_)
- int **KA_Get_Thread_ID** (void)
- int **KA_Get_Thread_Priority** (int id_)
- const char ∗ **KA_Get_Thread_State** (int id_)

Variables

- static uint64_t **u64IdleTime** = 0
- static FILE ∗ **fKernelState** = NULL
- static FILE ∗ **fInterrupts** = NULL
- static Mark3_Thread_Info_t ∗ **pstThreadInfo** = NULL
- static uint16_t **u16NumThreads** = 0
- static Mark3_Thread_t ∗ **pstLastThread** = NULL
- static uint64_t **u64LastTime** = 0
- static uint8_t **u8LastPri** = 255
- static TLV_t ∗ **pstTLV** = NULL

4.103.1 Detailed Description

Mark3 RTOS Kernel-Aware Thread Profiling.

Definition in file ka_thread.c.

4.104 ka_thread.c

```
/***************************************************************************
 *     (     (                      (     |
 *    )\ )  )\ )    (              )\ )   |
 *   (()/( (()/(    )\     (   (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                        | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#include "kernel_aware.h"
#include "debug_sym.h"
#include "write_callout.h"
#include "interrupt_callout.h"
#include "tlv_file.h"
#include "ka_thread.h"

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define THREAD_STATE_EXIT       0
#define THREAD_STATE_READY      1
#define THREAD_STATE_BLOCKED    2
#define THREAD_STATE_STOP       3


//---------------------------------------------------------------------------
typedef struct
{
    uint16_t u16NextPtr;
    uint16_t u16PrevPtr;

    uint16_t u16StackTopPtr;
```

```
    uint16_t u16StackPtr;

    uint8_t u8ThreadID;

    uint8_t u8Priority;

    uint8_t u8CurPriority;

    uint8_t u8ThreadState;

    uint16_t u16StackSize;

    uint16_t u16CurrentThreadList;
    uint16_t u16OwnerThreadList;

    uint16_t u16EntryPoint;

    void *m_pvArg;

    uint16_t u16Quantum;

} Mark3_Thread_t;

//---------------------------------------------------------------------------
typedef struct
{
    Mark3_Thread_t *pstThread;
    uint8_t        u8ThreadID;
    uint64_t       u64TotalCycles;
    uint64_t       u64EpockCycles;
    bool           bActive;
} Mark3_Thread_Info_t;

//---------------------------------------------------------------------------
typedef struct
{
    uint64_t        u64Timestamp;
    uint16_t        u16StackMargin;
    uint8_t         u8ThreadID;
    uint8_t         u8ThreadPri;
} Mark3ContextSwitch_TLV_t;

//---------------------------------------------------------------------------
static uint64_t u64IdleTime = 0;
static FILE *fKernelState = NULL;
static FILE *fInterrupts = NULL;
static Mark3_Thread_Info_t *pstThreadInfo = NULL;
static uint16_t u16NumThreads = 0;

static Mark3_Thread_t *pstLastThread = NULL;
static uint64_t u64LastTime = 0;
static uint8_t u8LastPri = 255;
//---------------------------------------------------------------------------
static TLV_t *pstTLV = NULL;

//---------------------------------------------------------------------------
static void Mark3KA_AddKnownThread( Mark3_Thread_t *pstThread_ )
{
    // Bail if the thread pointer is NULL
    if (!pstThread_ || ((uint32_t)pstThread_ == (uint32_t)stCPU.pstRAM->au8RAM))
    {
        return;
    }

    // Check to see if a thread has already been tagged at this address
    bool bExists = false;
    if (pstThreadInfo)
    {
        int i;
        for (i = 0; i < u16NumThreads; i++)
        {
            Mark3_Thread_t *pstThread = pstThreadInfo[i].pstThread;
            // If there are other threads that exist at this address,
            if (pstThread == pstThread_)
            {
                // If the stored thread's ID is different than the ID being presented here,
                // then it's a dynamic thread involved.  Create a new threadinfo object to track it.
                if (pstThreadInfo[i].u8ThreadID != pstThread_->u8ThreadID)
                {
                    pstThreadInfo[i].bActive = false;
                }
                // Thread IDs are the same, thread has already been tracked, don't do anything.
                else
                {
                    bExists = true;
                }
```

```
            }
        }
    }

    // If not already known, add the thread to the list of known threads.
    if (!bExists)
    {
        u16NumThreads++;
        pstThreadInfo = (Mark3_Thread_Info_t*)realloc(pstThreadInfo, sizeof(Mark3_Thread_Info_t) * u16NumThreads);

        pstThreadInfo[u16NumThreads - 1].pstThread = pstThread_;
        pstThreadInfo[u16NumThreads - 1].u64EpockCycles = 0;
        pstThreadInfo[u16NumThreads - 1].u64TotalCycles = 0;
        pstThreadInfo[u16NumThreads - 1].u8ThreadID = pstThread_->u8ThreadID;
        pstThreadInfo[u16NumThreads - 1].bActive = true;
    }
}

//---------------------------------------------------------------------------
Mark3_Thread_t *Mark3KA_GetCurrentThread(void)
{
    Debug_Symbol_t *pstSymbol = 0;

    pstSymbol = Symbol_Find_Obj_By_Name( "g_pclCurrent" );

    // Use pstSymbol's address to get a pointer to the current thread.
    if (!pstSymbol)
    {
        return 0;
    }

    uint16_t u16CurrPtr = (uint16_t)(pstSymbol->u32StartAddr & 0x0000FFFF);
    if (!u16CurrPtr)
    {
        return 0;
    }

    // Now that we have the address of g_pstCurrent, dereference the pointer
    // to get the address of the current thread.

    uint16_t u16CurrAddr = ((uint16_t)(stCPU.pstRAM->au8RAM[ u16CurrPtr + 1 ]) << 8) +
                            stCPU.pstRAM->au8RAM[ u16CurrPtr ];

    // Return a pointer to the thread as it is in memory.
    return (Mark3_Thread_t*)(&stCPU.pstRAM->au8RAM[ u16CurrAddr ]);
}

//---------------------------------------------------------------------------
static uint8_t Mark3KA_GetCurrentPriority(void)
{
    Mark3_Thread_t *pstThread = Mark3KA_GetCurrentThread();
    if (!pstThread)
    {
        return 0;
    }
    uint8_t *pucData = (uint8_t*)pstThread;

    // If the curpriority member is set, it means we're in the middle of
    // priority inheritence.  If it's zero, return the normal priority
    if (0 == pstThread->u8CurPriority)
    {
        return pstThread->u8Priority;
    }
    return pstThread->u8CurPriority;
}

//---------------------------------------------------------------------------
static uint16_t Mark3KA_GetStackMargin( Mark3_Thread_t *pstThread_ )
{
    uint16_t u16StackBase = pstThread_->u16StackPtr;
    uint16_t u16StackSize = pstThread_->u16StackSize;

    int i;

    for (i = 0; i < u16StackSize; i++)
    {
        if (255 != stCPU.pstRAM->au8RAM[ u16StackBase + i ])
        {
            return (uint16_t)i;
        }
    }

    return u16StackSize;
}

//---------------------------------------------------------------------------
```

```
static uint16_t Mark3KA_GetCurrentStackMargin(void)
{
   return Mark3KA_GetStackMargin( Mark3KA_GetCurrentThread() );
}

//---------------------------------------------------------------------------
static bool KA_StackWarning( uint16_t u16Addr_, uint8_t u8Data_ )
{
    if (u8Data_ != 0xFF && stCPU.pstRAM->au8RAM[ u16Addr_ ] == 0xFF )
    {
        fprintf( stderr, "[WARNING] Near stack-overflow detected - Thread %d, Stack Margin %d\n",
                Mark3KA_GetCurrentThread()->u8ThreadID,
                Mark3KA_GetCurrentStackMargin() );
    }
    return true;
}

//---------------------------------------------------------------------------
static bool KA_ThreadChange( uint16_t u16Addr_, uint8_t u8Data_ )
{
    uint8_t  u8Pri = Mark3KA_GetCurrentPriority();
    uint8_t  u8Thread = Mark3KA_GetCurrentThread()->u8ThreadID;
    uint16_t u16Margin = Mark3KA_GetCurrentStackMargin();

    // -- Add context switch instrumentation to TLV
    Mark3ContextSwitch_TLV_t stData;

    stData.u8ThreadID = u8Thread;
    stData.u8ThreadPri = u8Pri;
    stData.u16StackMargin = u16Margin;
    stData.u64Timestamp = stCPU.u64CycleCount;

    memcpy( &(pstTLV->au8Data[0]), &stData, sizeof(stData) );
    TLV_Write( pstTLV );

    if (u8LastPri == 0)
    {
        u64IdleTime += (stCPU.u64CycleCount - u64LastTime);
    }

    // Track this as a known-thread internally for future reporting.
    Mark3KA_AddKnownThread( Mark3KA_GetCurrentThread() );

    if (pstLastThread && u64LastTime)
    {
        Mark3_Thread_t *pstThread;
        int i;
        for ( i = 0; i < u16NumThreads; i++ )
        {
            if  ( (pstLastThread == pstThreadInfo[i].pstThread) &&
                  (pstLastThread->u8ThreadID == pstThreadInfo[i].u8ThreadID) )
            {
                pstThreadInfo[i].u64TotalCycles += stCPU.u64CycleCount - u64LastTime;
            }
        }
    }

    u64LastTime = stCPU.u64CycleCount;
    u8LastPri = u8Pri;

    // Add watchpoints on active thread stack at 32-bytes from the end
    // of the stack.  That way, we can immediately detect stack smashing threats
    // without having to hunt.

    uint16_t u16StackWarning = Mark3KA_GetCurrentThread()->u16StackPtr + 32;
    WriteCallout_Add( KA_StackWarning, u16StackWarning );

    // Cache the current thread for use as the "last run" thread in
    // subsequent iterations
    pstLastThread = Mark3KA_GetCurrentThread();

    return true;
}

//---------------------------------------------------------------------------
void KA_PrintThreadInfo(void)
{
    int i;
    uint64_t u64TrackedThreadTime = 0;

    uint16_t u16LastThread = (uint16_t)((void*)Mark3KA_GetCurrentThread() - (void*)&stCPU.pstRAM->au8RAM[0]
);

    KA_ThreadChange( u16LastThread, 0 );

    for ( i = 0; i < u16NumThreads; i++ )
    {
```

```
        u64TrackedThreadTime += pstThreadInfo[i].u64TotalCycles;
    }

    printf( "ThreadID, ThreadAddr, TotalCycles, PercentCPU, IsActive, Prio, StackMargin\n");
    for ( i = 0; i < u16NumThreads; i++ )
    {
        printf( "%d, %04X, %llu, %0.3f, %d, %d, %d\n",
                        pstThreadInfo[i].u8ThreadID,
                        (uint16_t)((void*)(pstThreadInfo[i].pstThread) - (void*)(&stCPU.pstRAM->au8RAM[0])),
                        pstThreadInfo[i].u64TotalCycles,
                        (double)pstThreadInfo[i].u64TotalCycles / u64TrackedThreadTime * 100.0f,
                        pstThreadInfo[i].bActive,
                        (pstThreadInfo[i].bActive ? pstThreadInfo[i].pstThread->u8Priority : 0),
                        (pstThreadInfo[i].bActive ? Mark3KA_GetStackMargin(pstThreadInfo[i].pstThread) : 0)
                    ) ;
    }
}

//---------------------------------------------------------------------------
void KA_Thread_Init( void )
{
    Debug_Symbol_t *pstSymbol = 0;
    pstSymbol = Symbol_Find_Obj_By_Name( "g_pclCurrent" );

    // Use pstSymbol's address to get a pointer to the current thread.
    if (!pstSymbol)
    {
        return;
    }

    // Ensure that we actually have the information we need at a valid address
    uint16_t u16CurrPtr = (uint16_t)(pstSymbol->u32StartAddr & 0x0000FFFF);
    if (!u16CurrPtr)
    {
        return;
    }

    // Add a callback so that when g_pstCurrent changes, we can update our
    // locally-tracked statistics.
    WriteCallout_Add( KA_ThreadChange , u16CurrPtr + 1 );

    pstTLV = TLV_Alloc( sizeof(Mark3ContextSwitch_TLV_t) );
    pstTLV->eTag = TAG_KERNEL_AWARE_CONTEXT_SWITCH;
    pstTLV->u16Len = sizeof(Mark3ContextSwitch_TLV_t);

    atexit( KA_PrintThreadInfo );
}

//---------------------------------------------------------------------------
char *KA_Get_Thread_Info_XML(uint8_t **thread_ids, uint16_t *thread_count)
{
    char *ret = (char*)malloc(4096);
    char *writer = ret;
    uint8_t *new_ids;

    if (u16NumThreads && thread_ids)
    {
        new_ids = (uint8_t*)malloc(u16NumThreads);
        *thread_ids = new_ids;
    }

    writer += sprintf( writer,
            "<threads>" );

    if (!u16NumThreads) {
        writer += sprintf( writer,
        "  <thread id=\"0\" core=\"0\">"
        "  System Thread - Priority N/A [Running] "
        "  </thread>");
    }

    int i;
    int count = 0;
    for (i = 0; i < u16NumThreads; i++)
    {
        if (pstThreadInfo[i].bActive)
        {
            if (pstThreadInfo[i].u8ThreadID == 255)
            {
                writer += sprintf(writer,
                "  <thread id=\"255\" core=\"0\">"
                "  Mark3 Thread - Priority 0 [IDLE]");
            }
            else if (pstThreadInfo[i].u8ThreadID == Mark3KA_GetCurrentThread()->u8ThreadID)
            {
                writer += sprintf(writer,
```

```
                "  <thread id=\"%d\" core=\"0\">"
                "  Mark3 Thread - Priority %d [Running] " ,
                pstThreadInfo[i].u8ThreadID,
                pstThreadInfo[i].pstThread->u8CurPriority );
            }
            else
            {
                writer += sprintf(writer,
                "  <thread id=\"%d\" core=\"0\">"
                "  Mark3 Thread - Priority %d" ,
                pstThreadInfo[i].u8ThreadID,
                pstThreadInfo[i].pstThread->u8CurPriority );
            }
            if (thread_ids)
            {
                new_ids[count++] = pstThreadInfo[i].u8ThreadID;
            }
        }
        writer += sprintf( writer, "  </thread>");
    }

    sprintf( writer, "</threads>" );
    if (thread_count)
    {
        *thread_count = count;
    }
    return ret;
}

//---------------------------------------------------------------------------
Mark3_Context_t *KA_Get_Thread_Context(uint8_t id_)
{
    int i;
    for (i = 0; i < u16NumThreads; i++)
    {
        if (pstThreadInfo[i].bActive)
        {
            if (pstThreadInfo[i].u8ThreadID == id_)
            {
                Mark3_Context_t *new_ctx = (Mark3_Context_t*)malloc(sizeof(Mark3_Context_t));
                uint16_t context_addr = pstThreadInfo[i].pstThread->u16StackTopPtr;

                new_ctx->SPH = stCPU.pstRAM->au8RAM[context_addr - 1];
                new_ctx->SPL = stCPU.pstRAM->au8RAM[context_addr];

                int j = 0;
                for (i = 31; i >= 0; i--)
                {
                    new_ctx->r[i] = stCPU.pstRAM->au8RAM[context_addr + 1 + j];
                    j++;
                }
                new_ctx->SREG = stCPU.pstRAM->au8RAM[context_addr + 33];
                uint16_t PC = *(uint16_t*)(&stCPU.pstRAM->au8RAM[context_addr + 34]);
                PC = ((PC & 0xFF00)>>8) | ((PC & 0x00FF) << 8);
                new_ctx->PC = PC;

                return new_ctx;
            }
        }
    }
    return NULL;
}

//---------------------------------------------------------------------------
int KA_Get_Thread_ID(void)
{
    return Mark3KA_GetCurrentThread()->u8ThreadID;
}

//---------------------------------------------------------------------------
int KA_Get_Thread_Priority( int id_ )
{
    int i;
    for (i = 0; i < u16NumThreads; i++)
    {
        if (pstThreadInfo[i].bActive)
        {
            if (pstThreadInfo[i].u8ThreadID == id_)
            {
                return pstThreadInfo[i].pstThread->u8CurPriority;
            }
        }
    }
    return -1;
}
```

```
//----------------------------------------------------------------------------
const char *KA_Get_Thread_State( int id_ )
{
    int i;
    for (i = 0; i < u16NumThreads; i++)
    {
        if (pstThreadInfo[i].bActive)
        {
            if (pstThreadInfo[i].u8ThreadID == id_)
            {
                switch (pstThreadInfo[i].pstThread->u8ThreadState)
                {
                case THREAD_STATE_BLOCKED:
                    return "Blocked";
                case THREAD_STATE_EXIT:
                    return "Exit";
                case THREAD_STATE_READY:
                    if (id_ == Mark3KA_GetCurrentThread()->u8ThreadID)
                    {
                        return "Running";
                    }
                    return "Ready";
                case THREAD_STATE_STOP:
                    return "Stopped";
                default:
                    return "unknown";
                }
            }
        }
    }
    return -1;
}
```

4.105 ka_thread.h File Reference

Mark3 RTOS Kernel-Aware Thread Profiling.

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

Data Structures

- struct Mark3_Context_t

Functions

- void **KA_Thread_Init** (void)
- int **KA_Get_Thread_Priority** (int id_)
- const char ∗ **KA_Get_Thread_State** (int id_)

4.105.1 Detailed Description

Mark3 RTOS Kernel-Aware Thread Profiling.

Definition in file ka_thread.h.

4.106 ka_thread.h

```
/*****************************************************************************
 *     (     (                     (     |
 *    )\ )  )\ )    (              )\ )  |
 *   (()/( (()/(    )\      (   (  (()/(  | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))   | -- [ AVR ] ---------------------
 *   | |_  | |     (_)_\(_)\ \ / / | _ \  | -- [ Virtual ] -----------------
```

```
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |_____| /_/ \_\   \_/   |_|_\   |
 *                                          | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __KA_THREAD_H__
#define __KA_THREAD_H__

#include <stdint.h>

typedef struct
{
    uint8_t SPH;
    uint8_t SPL;
    uint8_t r[32];
    uint8_t SREG;
    uint16_t PC;
} Mark3_Context_t;

//---------------------------------------------------------------------------
void KA_Thread_Init( void );

int KA_Get_Thread_Priority(int id_);

const char *KA_Get_Thread_State( int id_ );

#endif
```

4.107 ka_trace.c File Reference

Mark3 RTOS Kernel-Aware Trace functionality.

```
#include "kernel_aware.h"
#include "debug_sym.h"
#include "ka_trace.h"
#include "tlv_file.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
```

Data Structures

- struct KernelAwareTrace_t

Functions

- void KA_EmitTrace (KernelAwareCommand_t eCmd_)

    KA_EmitTrace.
- void KA_Print (void)

    KA_Print.
- void KA_Trace_Init (void)

    KA_Trace_Init.

Variables

- static TLV_t * **pstTLV** = NULL

4.107.1 Detailed Description

Mark3 RTOS Kernel-Aware Trace functionality.

Definition in file ka_trace.c.

4.107.2 Function Documentation

4.107.2.1 void KA_EmitTrace (KernelAwareCommand_t *eCmd_*)

KA_EmitTrace.

Process a kernel trace event and emit the appropriate record into our TLV stream output

Parameters

*eCmd_*	Type of trace command being emitted.

Definition at line 47 of file ka_trace.c.

4.107.2.2 void KA_Print (void)

KA_Print.

Print a kernel string event to the console and TLV stream.

Definition at line 81 of file ka_trace.c.

4.107.2.3 void KA_Trace_Init (void)

KA_Trace_Init.

Initialize the local TLV buffers, etc. Must be called prior to use

Definition at line 97 of file ka_trace.c.

4.108 ka_trace.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 2014-15, Funkenstein Software Consulting, All rights reserved
00013  *     See license.txt for details
00014  ****************************************************************************/
00021 #include "kernel_aware.h"
00022 #include "debug_sym.h"
00023
00024 #include "ka_trace.h"
00025 #include "tlv_file.h"
00026
00027 #include <stdint.h>
00028 #include <stdio.h>
00029 #include <stdlib.h>
00030 #include <string.h>
00031 #include <time.h>
00032
00033 //---------------------------------------------------------------------------
00034 typedef struct
00035 {
00036     uint16_t u16File;
00037     uint16_t u16Line;
```

```
    uint16_t u16Code;
    uint16_t u16Arg1;
    uint16_t u16Arg2;
} KernelAwareTrace_t;

//---------------------------------------------------------------------------
static TLV_t *pstTLV = NULL;

//---------------------------------------------------------------------------
void KA_EmitTrace( KernelAwareCommand_t eCmd_ )
{
    Debug_Symbol_t *pstSymbol = Symbol_Find_Obj_By_Name( "g_stKAData"
);
    if (!pstSymbol)
    {
        return;
    }

    KernelAwareTrace_t *pstTrace = (KernelAwareTrace_t*)&stCPU.
pstRAM->au8RAM[ pstSymbol->u32StartAddr ];
    switch (eCmd_)
    {
    case KA_COMMAND_TRACE_0:
        pstTLV->eTag = KA_COMMAND_TRACE_0;
        pstTLV->u16Len = 6;
        break;
    case KA_COMMAND_TRACE_1:
        pstTLV->eTag = KA_COMMAND_TRACE_1;
        pstTLV->u16Len = 8;
        break;
    case KA_COMMAND_TRACE_2:
        pstTLV->eTag = KA_COMMAND_TRACE_2;
        pstTLV->u16Len = 10;
        break;
    default:
        return;
    }
    fprintf(stderr, "Trace: %04X, %04X, %04X, %04X, %04x\n", pstTrace->u16File, pstTrace->u16Line,
                    pstTrace->u16Code, pstTrace->u16Arg1, pstTrace->u16Arg2 );

    memcpy( pstTLV->au8Data, pstTrace, pstTLV->u16Len );
    TLV_Write( pstTLV );
}

//---------------------------------------------------------------------------
void KA_Print( void )
{
    Debug_Symbol_t *pstSymbol = Symbol_Find_Obj_By_Name( "g_stKAData"
);
    if (!pstSymbol)
    {
        return;
    }

    uint16_t u16NamePtr = *((uint16_t*)&stCPU.pstRAM->au8RAM[ pstSymbol->
u32StartAddr ]);
    const char *szString = (const char*)&stCPU.pstRAM->au8RAM[ u16NamePtr ];

    strcpy( pstTLV->au8Data, szString );
    fprintf( stderr, "%s", szString );
}

//---------------------------------------------------------------------------
void KA_Trace_Init(void)
{
    pstTLV = TLV_Alloc(64);
}
```

4.109 ka_trace.h File Reference

Mark3 RTOS Kernel-Aware Trace and Print Functionality.

```
#include "kernel_aware.h"
#include "debug_sym.h"
#include "ka_trace.h"
#include "tlv_file.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
```

Functions

- void KA_EmitTrace (KernelAwareCommand_t eCmd_)

    KA_EmitTrace.
- void KA_Print (void)

    KA_Print.
- void KA_Trace_Init (void)

    KA_Trace_Init.

4.109.1 Detailed Description

Mark3 RTOS Kernel-Aware Trace and Print Functionality.

Definition in file ka_trace.h.

4.109.2 Function Documentation

4.109.2.1 void KA_EmitTrace (KernelAwareCommand_t *eCmd_*)

KA_EmitTrace.

Process a kernel trace event and emit the appropriate record into our TLV stream output

Parameters

*eCmd_*	Type of trace command being emitted.

Definition at line 47 of file ka_trace.c.

4.109.2.2 void KA_Print (void)

KA_Print.

Print a kernel string event to the console and TLV stream.

Definition at line 81 of file ka_trace.c.

4.109.2.3 void KA_Trace_Init (void)

KA_Trace_Init.

Initialize the local TLV buffers, etc. Must be called prior to use

Definition at line 97 of file ka_trace.c.

4.110 ka_trace.h

```
/*****************************************************************************
 *     (     (                      (    |
 *    )\ )  )\ )    (              )\ )  |
 *   (()/( (()/(    )\     (   (  (()/(  | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))    | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))   | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \  | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /  | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\  |
 *                                       | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __KA_TRACE__
#define __KA_TRACE__


#include "kernel_aware.h"
#include "debug_sym.h"

#include "ka_trace.h"
#include "tlv_file.h"

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

//---------------------------------------------------------------------------
void KA_EmitTrace( KernelAwareCommand_t eCmd_ );

//---------------------------------------------------------------------------
void KA_Print( void );

//---------------------------------------------------------------------------
void KA_Trace_Init( void );

#endif

```

4.111 kernel_aware.c File Reference

Mark3 RTOS Kernel-Aware debugger.

```
#include "kernel_aware.h"
#include "debug_sym.h"
#include "write_callout.h"
#include "interrupt_callout.h"
#include "ka_interrupt.h"
#include "ka_profile.h"
#include "ka_thread.h"
#include "ka_trace.h"
#include "ka_graphics.h"
#include "ka_joystick.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
```

Functions

- static bool **KA_Command** (uint16_t u16Addr_, uint8_t u8Data_)
- static bool **KA_Set** (uint16_t u16Addr_, uint8_t u8Data_)
- void KernelAware_Init (void)

KernelAware_Init.

4.111.1 Detailed Description

Mark3 RTOS Kernel-Aware debugger.

Definition in file kernel_aware.c.

4.111.2 Function Documentation

4.111.2.1 void KernelAware_Init (void)

KernelAware_Init.

Initialize special RTOS kernel-aware debugger functionality when selected. Currently this is tied to Mark3 RTOS (see kernel_aware.c implementation), but can be abstracted using this simple interface to any other RTOS kernel or environment (but why would you – Mark3 is awesome!).

Definition at line 69 of file kernel_aware.c.

4.112 kernel_aware.c

```
/****************************************************************************
 *     (     (                      (    |
 *    )\ )  )\ )    (              )\ ) |
 *   (()/( (()/(    )\     (   (  (()/( | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))   | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))   | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \  | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /  | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\  |
 *                                       | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#include "kernel_aware.h"
#include "debug_sym.h"
#include "write_callout.h"
#include "interrupt_callout.h"

#include "ka_interrupt.h"
#include "ka_profile.h"
#include "ka_thread.h"
#include "ka_trace.h"
#include "ka_graphics.h"
#include "ka_joystick.h"

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>


//---------------------------------------------------------------------------
static bool KA_Command( uint16_t u16Addr_, uint8_t u8Data_ )
{
    switch (u8Data_)
    {
    case KA_COMMAND_PROFILE_INIT:   KA_Command_Profile_Begin();     break;
    case KA_COMMAND_PROFILE_STOP:   KA_Command_Profile_Stop();      break;
    case KA_COMMAND_PROFILE_START:  KA_Command_Profile_Start();     break;
    case KA_COMMAND_PROFILE_REPORT: KA_Command_Profile_Report();    break;
    case KA_COMMAND_TRACE_0:
    case KA_COMMAND_TRACE_1:
    case KA_COMMAND_TRACE_2:        KA_EmitTrace(u8Data_);          break;
    case KA_COMMAND_PRINT:          KA_Print();                     break;
    default:
        break;

    }
    return true;
```

```
}

//---------------------------------------------------------------------------
static bool KA_Set( uint16_t u16Addr_, uint8_t u8Data_ )
{
    fprintf(stderr, "ADDR: [%04X], Data: [%02X]\n", u16Addr_, u8Data_ );
    stCPU.pstRAM->au8RAM[ u16Addr_ & 0xFFFF ] = 1;
    return false;
}

//---------------------------------------------------------------------------
void KernelAware_Init( void )
{
    Debug_Symbol_t *pstSymbol = 0;

    // Add a callout for profiling information (present in Mark3 Unit Tests)
    pstSymbol = Symbol_Find_Obj_By_Name( "g_u8KACommand" );
    if (pstSymbol)
    {
        // Ensure that we actually have the information we need at a valid address
        uint16_t u16CurrPtr = (uint16_t)(pstSymbol->u32StartAddr & 0x0000FFFF);
        printf( "found kernel-aware command @ %04X\n", u16CurrPtr );
        if (u16CurrPtr)
        {
            // Add a callback so that when profiling state changes, we do something.
            WriteCallout_Add( KA_Command , u16CurrPtr );
        }
    }
    else
    {
        printf( "Unable to find g_ucKACommand\n" );
    }


    // Set the kernel's "simulator aware" flag, to let it know to configure itself
    // appropriately.

    pstSymbol = Symbol_Find_Obj_By_Name( "g_bIsKernelAware" );
    if (pstSymbol)
    {
        fprintf( stderr, "Addr: %4X, Name: %s\n", pstSymbol->u32StartAddr, pstSymbol->szName );
        // Ensure that we actually have the information we need at a valid address
        uint16_t u16CurrPtr = (uint16_t)(pstSymbol->u32StartAddr);

        if (u16CurrPtr)
        {
            // Add a callout so that the kernel-aware flag is *always* set.
            fprintf( stderr, "Adding writeout\n" );
            WriteCallout_Add( KA_Set , u16CurrPtr );
            fprintf( stderr, "done\n" );
        }
    }
    else
    {
        printf( "Unable to find g_bIsKernelAware" );
    }


    KA_Interrupt_Init();
    KA_Thread_Init();
    KA_Profile_Init();
    KA_Trace_Init();

    KA_Graphics_Init();
    KA_Joystick_Init();

}
```

4.113 kernel_aware.h File Reference

Kernel-Aware debugger plugin interface.

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

Enumerations

- enum **KernelAwareCommand_t** {
KA_COMMAND_IDLE = 0, **KA_COMMAND_PROFILE_INIT**, **KA_COMMAND_PROFILE_START**, **KA_COMMAND_PROFILE_STOP**,
KA_COMMAND_PROFILE_REPORT, **KA_COMMAND_EXIT_SIMULATOR**, **KA_COMMAND_TRACE_0**, **KA_COMMAND_TRACE_1**,
KA_COMMAND_TRACE_2, **KA_COMMAND_PRINT** }

Functions

- void KernelAware_Init (void)

  KernelAware_Init.
- void **KA_Graphics_Init** (void) __attribute__((weak))
- void **KA_Joystick_Init** (void) __attribute__((weak))

4.113.1 Detailed Description

Kernel-Aware debugger plugin interface.

Definition in file kernel_aware.h.

4.113.2 Function Documentation

4.113.2.1 void KernelAware_Init (void)

KernelAware_Init.

Initialize special RTOS kernel-aware debugger functionality when selected. Currently this is tied to Mark3 RTOS (see kernel_aware.c implementation), but can be abstracted using this simple interface to any other RTOS kernel or environment (but why would you – Mark3 is awesome!).

Definition at line 69 of file kernel_aware.c.

4.114 kernel_aware.h

```
/****************************************************************************
 *     (      (                                   (       |
 *    )\ )   )\ )      (                         )\ )    |
 *   (()/(  (()/(     )\        (    (    (()/(     | -- [ Funkenstein ] -------------
 *    /(_))  /(_))((((_)()\    )\   /(_))        | -- [ Litle ] -------------------
 *   (_))_|(_))     )\ _ )\ ((_)((_)(_))       | -- [ AVR ] ---------------------
 *   | |_   | |      (_)_\(_)\ \ / / | _ \     | -- [ Virtual ] -----------------
 *   | __|  | |__     / _ \    \ V /  |   /     | -- [ Runtime ] -----------------
 *   |_|    |____| /_/ \_\    \_/    |_|_\     |
 *                                              | "Yeah, it does Arduino..."
 * ---------------------------------------------+------------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __KERNEL_AWARE_H__
#define __KERNEL_AWARE_H__

#include "elf_process.h"
#include "debug_sym.h"
#include "avr_cpu.h"

#include <stdint.h>

//---------------------------------------------------------------------------
typedef enum
{
    KA_COMMAND_IDLE = 0,
    KA_COMMAND_PROFILE_INIT,
    KA_COMMAND_PROFILE_START,
```

```
    KA_COMMAND_PROFILE_STOP,
    KA_COMMAND_PROFILE_REPORT,
    KA_COMMAND_EXIT_SIMULATOR,
    KA_COMMAND_TRACE_0,
    KA_COMMAND_TRACE_1,
    KA_COMMAND_TRACE_2,
    KA_COMMAND_PRINT
} KernelAwareCommand_t;

//---------------------------------------------------------------------------
void KernelAware_Init(void);

void KA_Graphics_Init( void )  __attribute__((weak));
void KA_Joystick_Init( void )  __attribute__((weak));
#endif
```

4.115 mega_eeprom.c File Reference

AVR atmega EEPROM plugin.

```
#include "mega_eeprom.h"
#include "avr_cpu.h"
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

Macros

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

Enumerations

- enum EEPROM_State_t {
  EEPROM_STATE_IDLE = 0, EEPROM_STATE_WRITE_ENABLE, EEPROM_STATE_READ, EEPROM_STATE_WRITE,
  EEPROM_STATES }
- enum EEPROM_Mode_t { EEPROM_MODE_ATOMIC = 0, EEPROM_MODE_ERASE, EEPROM_MODE_WRITE, **EEPROM_MODES** }

Functions

- static void **EEARH_Write** (uint8_t u8Addr_)
- static void **EEARL_Write** (uint8_t u8Addr_)
- static uint16_t **EEAR_Read** (void)
- static void **EEPE_Clear** (void)
- static void **EEPE_Set** (void)
- static bool **EEPE_Read** (void)
- static void **EERE_Clear** (void)
- static void **EERE_Set** (void)
- static bool **EERE_Read** (void)
- static void **EEMPE_Clear** (void)
- static void **EEMPE_Set** (void)
- static bool **EEMPE_Read** (void)
- static void **EERIE_Clear** (void)
- static void **EERIE_Set** (void)

- static bool **EERIE_Read** (void)
- static EEPROM_Mode_t **EEPM_Read** (void)
- static uint8_t **EEDR_Read** (void)
- static void **EEPROM_Init** (void *context_)
- static void **EEPROM_Read** (void *context_, uint8_t ucAddr_, uint8_t *pucValue_)
- static void EEPROM_Write (void *context_, uint8_t ucAddr_, uint8_t ucValue_)
- static void **EEPROM_Clock** (void *context_)

Variables

- static EEPROM_State_t **eState** = EEPROM_STATE_IDLE
- static uint32_t **u32CountDown** = 0
- AVRPeripheral **stEEPROM**

4.115.1 Detailed Description

AVR atmega EEPROM plugin.

Definition in file mega_eeprom.c.

4.115.2 Enumeration Type Documentation

4.115.2.1 enum EEPROM_Mode_t

Enumerator

EEPROM_MODE_ATOMIC Atomic Clear/Write operation.

EEPROM_MODE_ERASE Erase only.

EEPROM_MODE_WRITE Write only.

Definition at line 50 of file mega_eeprom.c.

4.115.2.2 enum EEPROM_State_t

Enumerator

EEPROM_STATE_IDLE EEPROM is idle.

EEPROM_STATE_WRITE_ENABLE EEPROM write is enabled (for 4 cycles)

EEPROM_STATE_READ EEPROM is reading a byte.

EEPROM_STATE_WRITE EEPROM is writing a byte.

Definition at line 38 of file mega_eeprom.c.

4.115.3 Function Documentation

4.115.3.1 static void EEPROM_Write (void * *context_*, uint8_t *ucAddr_*, uint8_t *ucValue_*) [static]

! ToDo - Fix the times to use RC-oscilator times, not CPU-clock times.

Definition at line 183 of file mega_eeprom.c.

4.115.4 Variable Documentation

4.115.4.1 AVRPeripheral stEEPROM

Initial value:

```
=
{
    EEPROM_Init,
    EEPROM_Read,
    EEPROM_Write,
    EEPROM_Clock,
    0,
    0x3F,
    0x3F
}
```

Definition at line 310 of file mega_eeprom.c.

4.116 mega_eeprom.c

```
/***************************************************************************
 *     (     (                      (     |
 *    )\ )  )\ )    (              )\ )   |
 *   (()/( (()/(    )\     (   (   (()/(  | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                        | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ***************************************************************************/
#include "mega_eeprom.h"

#include "avr_cpu.h"

#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#if 1
    #define DEBUG_PRINT(...)
#else
    #define DEBUG_PRINT     printf
#endif

//---------------------------------------------------------------------------
typedef enum
{
    EEPROM_STATE_IDLE = 0,
    EEPROM_STATE_WRITE_ENABLE,
    EEPROM_STATE_READ,
    EEPROM_STATE_WRITE,
    //--
    EEPROM_STATES
} EEPROM_State_t;


//---------------------------------------------------------------------------
typedef enum
{
    EEPROM_MODE_ATOMIC = 0,
    EEPROM_MODE_ERASE,
    EEPROM_MODE_WRITE,
    //----
    EEPROM_MODES
} EEPROM_Mode_t;

//---------------------------------------------------------------------------
static EEPROM_State_t eState = EEPROM_STATE_IDLE;
static uint32_t       u32CountDown = 0;

//---------------------------------------------------------------------------
static void EEARH_Write( uint8_t u8Addr_ )
```

```
{
    stCPU.pstRAM->stRegisters.EEARH = (u8Addr_ & 0x03);
}

//---------------------------------------------------------------------------
static void EEARL_Write( uint8_t u8Addr_ )
{
    stCPU.pstRAM->stRegisters.EEARL = u8Addr_;
}

//---------------------------------------------------------------------------
static uint16_t EEAR_Read( void )
{
    uint16_t u16Addr;
    u16Addr = ((uint16_t)(stCPU.pstRAM->stRegisters.EEARH) << 8) |
              (uint16_t)(stCPU.pstRAM->stRegisters.EEARL);
    return u16Addr;
}

//---------------------------------------------------------------------------
static void EEPE_Clear(void)
{
    stCPU.pstRAM->stRegisters.EECR.EEPE = 0;
}

//---------------------------------------------------------------------------
static void EEPE_Set(void)
{
    stCPU.pstRAM->stRegisters.EECR.EEPE = 1;
}

//---------------------------------------------------------------------------
static bool EEPE_Read(void)
{
    return (stCPU.pstRAM->stRegisters.EECR.EEPE == 1);
}
//---------------------------------------------------------------------------
static void EERE_Clear(void)
{
    stCPU.pstRAM->stRegisters.EECR.EERE = 0;
}

//---------------------------------------------------------------------------
static void EERE_Set(void)
{
    stCPU.pstRAM->stRegisters.EECR.EERE = 1;
}

//---------------------------------------------------------------------------
static bool EERE_Read(void)
{
    return (stCPU.pstRAM->stRegisters.EECR.EERE == 1);
}
//---------------------------------------------------------------------------
static void EEMPE_Clear(void)
{
    stCPU.pstRAM->stRegisters.EECR.EEMPE = 0;
}

//---------------------------------------------------------------------------
static void EEMPE_Set(void)
{
    stCPU.pstRAM->stRegisters.EECR.EEMPE = 1;
}

//---------------------------------------------------------------------------
static bool EEMPE_Read(void)
{
    return (stCPU.pstRAM->stRegisters.EECR.EEMPE == 1);
}

//---------------------------------------------------------------------------
static void EERIE_Clear(void)
{
    stCPU.pstRAM->stRegisters.EECR.EERIE = 0;
}

//---------------------------------------------------------------------------
static void EERIE_Set(void)
{
    stCPU.pstRAM->stRegisters.EECR.EERIE = 1;
}

//---------------------------------------------------------------------------
static bool EERIE_Read(void)
{
    return (stCPU.pstRAM->stRegisters.EECR.EERIE == 1);
```

```
}

//---------------------------------------------------------------------------
static EEPROM_Mode_t EEPM_Read(void)
{
    EEPROM_Mode_t eRet;
    eRet = (EEPROM_Mode_t)(stCPU.pstRAM->stRegisters.EECR.r & (0x30)) >> 4;
    return eRet;
}

//---------------------------------------------------------------------------
static uint8_t EEDR_Read(void)
{
    return stCPU.pstRAM->stRegisters.EEDR;
}

//---------------------------------------------------------------------------
static void EEPROM_Init(void *context_ )
{
    eState = EEPROM_STATE_IDLE;
    u32CountDown = 0;
}

//---------------------------------------------------------------------------
static void EEPROM_Read(void *context_, uint8_t ucAddr_, uint8_t *pucValue_ )
{
    DEBUG_PRINT( "EEPROM Read %2x\n", stCPU.pstRAM->stRegisters.EECR.r );
    *pucValue_ = stCPU.pstRAM->stRegisters.EECR.r;
}

//---------------------------------------------------------------------------
static void EEPROM_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
{
    // We're only interested in the EECR register.  If we really want to be
    // 100% CPU-accurate, we'd take into account a ton of additionl logic for
    // other peripherals (CPU SPM registers, etc.), but that's a lot of code
    // when pretty much everyone is going to be using the app note or the AVR
    // libc implementation, which is very much "sunny case" code.  In short,
    // this will handle incorrectly-implemented code incorrectly.

    stCPU.pstRAM->stRegisters.EECR.r |= (ucValue_ & 0x3F);

    switch (eState)
    {
        case EEPROM_STATE_IDLE:
        {
            if ((ucValue_ & 0x01) == 0x01)  // Read
            {
                // When the data is read, the data is available in the next instruction
                // but the CPU is halted for 4 cycles before it's executed.
                DEBUG_PRINT( "EEPROM Read\n" );
                eState = EEPROM_STATE_READ;
                u32CountDown = 4;

                stCPU.u16ExtraCycles += u32CountDown;
                stCPU.u64CycleCount += u32CountDown;

                // Read data at EEPROM address to EEPROM data register
                stCPU.pstRAM->stRegisters.EEDR = stCPU.pu8EEPROM[ EEAR_Read() ];
            }
            else if ((ucValue_ & 0x04) == 0x04) // Program Enable
            {
                // Must initiate a write within 4 cycles of enabling the EEPROM write bit
                DEBUG_PRINT( "EEPROM Write Enable \n" );
                eState = EEPROM_STATE_WRITE_ENABLE;
                u32CountDown = 4;
            }
        }
            break;

        case EEPROM_STATE_WRITE_ENABLE:
        {
            if ((ucValue_ & 0x02) == 0x02) // Value has EEPE
            {
                eState = EEPROM_STATE_WRITE;
                DEBUG_PRINT( "EEPROM Write\n" );
                switch ( EEPM_Read() )
                {
                    case EEPROM_MODE_ATOMIC:
                    {
                        stCPU.pu8EEPROM[ EEAR_Read() ] = EEDR_Read();
                        u32CountDown = 48000;
                    }
                        break;
                    case EEPROM_MODE_WRITE:
                    {
                        // EEPROM works by setting individual bits -- once a bit is set, it must be
```

```
                            // cleared before it can be reset.
                            stCPU.pu8EEPROM[ EEAR_Read() ] |= EEDR_Read();
                            u32CountDown = 25000;
                        }
                            break;
                        case EEPROM_MODE_ERASE:
                        {
                            // EEPROM is 0 when cleared
                            stCPU.pu8EEPROM[ EEAR_Read() ] = 0x00;
                            u32CountDown = 25000;
                        }
                            break;
                        default:
                            break;
                    }
                }
            }
            break;
        default:
            break;
    }
}

//---------------------------------------------------------------------------
static void EEPROM_Clock(void *context_)
{

    if (u32CountDown)
    {
        // DEBUG_PRINT( "EEPROM Clock %d\n", u32CountDown );

        u32CountDown--;
        if (!u32CountDown)
        {
            // We're only interested in the EECR register.
            switch (eState)
            {
                case EEPROM_STATE_WRITE:
                {
                    EEPE_Clear();
                    EERE_Clear();
                    EEMPE_Clear();

                    eState = EEPROM_STATE_IDLE;
                }
                    break;
                case EEPROM_STATE_READ:
                {
                    EEPE_Clear();
                    EERE_Clear();
                    EEMPE_Clear();

                    eState = EEPROM_STATE_IDLE;
                }
                    break;
                case EEPROM_STATE_WRITE_ENABLE:
                {
                    EEMPE_Clear();
                    EERE_Clear();
                    eState = EEPROM_STATE_IDLE;
                }
                    break;
                default:
                    break;
            }
        }
    }
}

//---------------------------------------------------------------------------
AVRPeripheral stEEPROM =
{
    EEPROM_Init,
    EEPROM_Read,
    EEPROM_Write,
    EEPROM_Clock,
    0,
    0x3F,
    0x3F
};

```

4.117 mega_eeprom.h File Reference

AVR atmega EEPROM plugin.

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

Variables

- AVRPeripheral **stEEPROM**

4.117.1 Detailed Description

AVR atmega EEPROM plugin.

Definition in file mega_eeprom.h.

4.118 mega_eeprom.h

```
/*******************************************************************************
 *      (      (                             (      |
 *     )\ )   )\ )     (                )\ )    |
 *    (()/(  (()/(     )\      (    (  (()/(    | -- [ Funkenstein ] -------------
 *     /(_))  /(_))((((_)()\   )\   /(_))     | -- [ Litle ] -----------------
 *    (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ---------------------
 *    | |_   | |    (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *    | __|  | |__   / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
 *    |_|    |____| /_/ \_\   \_/   |_|_\    |
 *                                           | "Yeah, it does Arduino..."
 * ---------------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ********************************************************************************/
#ifndef __MEGA_EEPROM_H__
#define __MEGA_EEPROM_H__

#include "avr_peripheral.h"


extern AVRPeripheral stEEPROM;

#endif // __MEGA_EEPROM_H__
```

4.119 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_SENS↩ E_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**

4.119.1 Detailed Description

ATMega External Interrupt Implementation.

Definition in file mega_eint.c.

4.119.2 Enumeration Type Documentation

4.119.2.1 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.

4.119.3 Function Documentation

4.119.3.1 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.

4.119.4 Variable Documentation

4.119.4.1 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.

4.119.4.2 AVRPeripheral stEINT_b

Initial value:

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

Definition at line 294 of file mega_eint.c.

4.120 mega_eint.c

```
/****************************************************************************
 *     (      (                          (      |
 *    )\ )   )\ )      (               )\ )    |
 *   (()/(  (()/(     )\       (    (  (()/(    | -- [ Funkenstein ] -------------
 *    /(_))  /(_))((((_)()\   )\  /(_))      | -- [ Litle ] -------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ---------------------
 *   | |_   | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *   | __|  | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
 *   |_|    |____|  /_/ \_\   \_/   |_|_\    |
 *                                            | "Yeah, it does Arduino..."
 * ---------------------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#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"

#define DEBUG_PRINT(...)

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


//---------------------------------------------------------------------------
```

```
static InterruptSense_t eINT0Sense;
static InterruptSense_t eINT1Sense;
static uint8_t ucLastINT0;
static uint8_t ucLastINT1;

//---------------------------------------------------------------------------
static void EINT_AckInt(  uint8_t ucVector_);

//---------------------------------------------------------------------------
static void EINT_Init(void *context_ )
{
    eINT0Sense = INT_SENSE_LOW;
    eINT1Sense = INT_SENSE_LOW;
    ucLastINT0 = 0;
    ucLastINT1 = 0;

    // Register interrupt callback functions
    CPU_RegisterInterruptCallback(EINT_AckInt, 0x01);
    CPU_RegisterInterruptCallback(EINT_AckInt, 0x02);
}

//---------------------------------------------------------------------------
static void EINT_Read(void *context_, uint8_t ucAddr_, uint8_t *pucValue_ )
{
    *pucValue_ = stCPU.pstRAM->au8RAM[ucAddr_];
}

//---------------------------------------------------------------------------
static void EICRA_Write( uint8_t ucValue_ )
{
    DEBUG_PRINT("EICRA Clock\n");
    ucValue_ &= 0x0F;     // Only the bottom 2 bits are valid.
    stCPU.pstRAM->stRegisters.EICRA.r = ucValue_;

    // Change local interrupt sense value.
    if ((stCPU.pstRAM->stRegisters.EICRA.ISC00 == 0) &&
        (stCPU.pstRAM->stRegisters.EICRA.ISC01 == 0))
    {
        DEBUG_PRINT("I0-low\n");
        eINT0Sense = INT_SENSE_LOW;
    }
    else if ((stCPU.pstRAM->stRegisters.EICRA.ISC00 == 1) &&
             (stCPU.pstRAM->stRegisters.EICRA.ISC01 == 0))
    {
        DEBUG_PRINT("I0-change\n");
        eINT0Sense = INT_SENSE_CHANGE;
    }
    else if ((stCPU.pstRAM->stRegisters.EICRA.ISC00 == 0) &&
             (stCPU.pstRAM->stRegisters.EICRA.ISC01 == 1))
    {
        DEBUG_PRINT("I0-fall\n");
        eINT0Sense = INT_SENSE_FALL;
    }
    else if ((stCPU.pstRAM->stRegisters.EICRA.ISC00 == 1) &&
             (stCPU.pstRAM->stRegisters.EICRA.ISC01 == 1))
    {
        DEBUG_PRINT("I0-risel\n");
        eINT0Sense = INT_SENSE_RISE;
    }

    if ((stCPU.pstRAM->stRegisters.EICRA.ISC10 == 0) &&
        (stCPU.pstRAM->stRegisters.EICRA.ISC11 == 0))
    {
        eINT1Sense = INT_SENSE_LOW;
    }
    else if ((stCPU.pstRAM->stRegisters.EICRA.ISC10 == 1) &&
             (stCPU.pstRAM->stRegisters.EICRA.ISC11 == 0))
    {
        eINT1Sense = INT_SENSE_CHANGE;
    }
    else if ((stCPU.pstRAM->stRegisters.EICRA.ISC10 == 0) &&
             (stCPU.pstRAM->stRegisters.EICRA.ISC11 == 1))
    {
        eINT1Sense = INT_SENSE_RISE;
    }
    else if ((stCPU.pstRAM->stRegisters.EICRA.ISC10 == 1) &&
             (stCPU.pstRAM->stRegisters.EICRA.ISC11 == 1))
    {
        eINT1Sense = INT_SENSE_FALL;
    }
    DEBUG_PRINT ("IntSense0,1: %d, %d\n", eINT0Sense, eINT1Sense);
    DEBUG_PRINT ("EICRA: %d, ISC00 : %d, ISC01 : %d, ISC10: %d, ISC11: %d\n",
                stCPU.pstRAM->stRegisters.EICRA.r,
                stCPU.pstRAM->stRegisters.EICRA.ISC00,
                stCPU.pstRAM->stRegisters.EICRA.ISC01,
                stCPU.pstRAM->stRegisters.EICRA.ISC10,
                stCPU.pstRAM->stRegisters.EICRA.ISC11
```

```
            );
}

//---------------------------------------------------------------------------
static void EIFR_Write( uint8_t ucValue_ )
{
    DEBUG_PRINT("EIFR Clock\n");
    ucValue_ &= 0x03;   // Only the bottom-2 bits are set
    stCPU.pstRAM->stRegisters.EIFR.r = ucValue_;
}

//---------------------------------------------------------------------------
static void EIMSK_Write( uint8_t ucValue_ )
{
    DEBUG_PRINT("EIMSK Write\n");
    ucValue_ &= 0x03;   // Only the bottom-2 bits are set
    stCPU.pstRAM->stRegisters.EIMSK.r = ucValue_;
}

//---------------------------------------------------------------------------
static void EINT_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
{
    DEBUG_PRINT("EINT Write\n");
    switch (ucAddr_)
    {
    case 0x69:  // EICRA
        EICRA_Write(ucValue_);
        break;
    case 0x3C:  // EIFR
        EIFR_Write(ucValue_);
        break;
    case 0x3D:  // EIMSK
        EIMSK_Write(ucValue_);
        break;
    default:
        break;
    }
}

//---------------------------------------------------------------------------
static void EINT_Clock(void *context_ )
{
    // Check to see if interrupts are enabled.  If so, check to see if the
    // interrupt mask is set, and then finally - whether or not an interupt
    // condition has occurred based on the interrupt sense mode.
    bool bSetINT0 = false;
    bool bSetINT1 = false;


    if (stCPU.pstRAM->stRegisters.EIMSK.INT0 == 1)
    {
        switch (eINT0Sense)
        {
        case INT_SENSE_LOW:
            if (stCPU.pstRAM->stRegisters.PORTD.PORT2 == 0)
            {
                DEBUG_PRINT(" SET INT0\n");
                bSetINT0 = true;
            }
            break;
        case INT_SENSE_CHANGE:
            if (stCPU.pstRAM->stRegisters.PORTD.PORT2 != ucLastINT0)
            {
                DEBUG_PRINT(" SET INT0\n");
                bSetINT0 = true;
            }
            break;
        case INT_SENSE_FALL:
            if ((stCPU.pstRAM->stRegisters.PORTD.PORT2 == 0) && (ucLastINT0 == 1))
            {
                DEBUG_PRINT(" SET INT0\n");
                bSetINT0 = true;
            }
            break;
        case INT_SENSE_RISE:
            if ((stCPU.pstRAM->stRegisters.PORTD.PORT2 == 1) && (ucLastINT0 == 0))
            {
                DEBUG_PRINT(" SET INT0\n");
                bSetINT0 = true;
            }
            break;
        }
    }
    if (stCPU.pstRAM->stRegisters.EIMSK.INT1 == 1)
    {
        switch (eINT0Sense)
        {
```

```
        case INT_SENSE_LOW:
            if (stCPU.pstRAM->stRegisters.PORTD.PORT3 == 0)
            {
                bSetINT1 = true;
            }
            break;
        case INT_SENSE_CHANGE:
            if (stCPU.pstRAM->stRegisters.PORTD.PORT3 != ucLastINT1)
            {
                bSetINT1 = true;
            }
            break;
        case INT_SENSE_FALL:
            if ((stCPU.pstRAM->stRegisters.PORTD.PORT3 == 0) && (ucLastINT1 == 1))
            {
                bSetINT1 = true;
            }
            break;
        case INT_SENSE_RISE:
            if ((stCPU.pstRAM->stRegisters.PORTD.PORT3 == 1) && (ucLastINT1 == 0))
            {
                bSetINT1 = true;
            }
            break;
        }
    }

    // Trigger interrupts where necessary
    if (bSetINT0)
    {
        stCPU.pstRAM->stRegisters.EIFR.INTF0 = 1;
        AVR_InterruptCandidate(0x01);
    }
    if (bSetINT1)
    {
        stCPU.pstRAM->stRegisters.EIFR.INTF1 = 1;
        AVR_InterruptCandidate(0x02);

    }
    // Update locally-cached copy of previous INT0/INT1 pin status.
    ucLastINT0 = stCPU.pstRAM->stRegisters.PORTD.PORT2;
    ucLastINT1 = stCPU.pstRAM->stRegisters.PORTD.PORT3;
}

//---------------------------------------------------------------------------
static void EINT_AckInt( uint8_t ucVector_)
{
    // We automatically clear the INTx flag as soon as the interrupt
    // is acknowledged.
    switch (ucVector_)
    {
    case 0x01:
        DEBUG_PRINT("INT0!\n");
        stCPU.pstRAM->stRegisters.EIFR.INTF0 = 0;
        break;
    case 0x02:
        DEBUG_PRINT("INT1!\n");
        stCPU.pstRAM->stRegisters.EIFR.INTF1 = 0;
        break;
    }
}

//---------------------------------------------------------------------------
AVRPeripheral stEINT_a =
{
    EINT_Init,
    EINT_Read,
    EINT_Write,
    EINT_Clock,
    NULL,
    0x69,
    0x69
};

//---------------------------------------------------------------------------
AVRPeripheral stEINT_b =
{
    NULL,
    EINT_Read,
    EINT_Write,
    NULL,
    NULL,
    0x3C,
    0x3D
};
```

4.121 mega_eint.h File Reference

ATMega External Interrupt Implementation.

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

Variables

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

4.121.1 Detailed Description

ATMega External Interrupt Implementation.

Definition in file mega_eint.h.

4.122 mega_eint.h

```
/**************************************************************************
 *     (      (                         (      |
 *    )\ )   )\ )      (              )\ )    |
 *   (()/(  (()/(     )\      (    (   (()/(    | -- [ Funkenstein ] -------------
 *    /(_))  /(_))((((_)()\   )\   /(_))      | -- [ Litle ] -------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ---------------------
 *   | |_   | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *   | __|  | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
 *   |_|    |____|  /_/ \_\   \_/   |_|_\    |
 *                                           | "Yeah, it does Arduino..."
 * ---------------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 *************************************************************************/
#ifndef __MEGA_EINT_H__
#define __MEGA_EINT_H__

#include "avr_peripheral.h"

extern AVRPeripheral stEINT_a;
extern AVRPeripheral stEINT_b;

#endif //__MEGA_EINT_H__
```

4.123 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**

4.123.1 Detailed Description

ATMega 16-bit timer implementation.

Definition in file mega_timer16.c.

4.123.2 Enumeration Type Documentation

4.123.2.1 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.

4.123.3 Function Documentation

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

! ToDo - Handle external timer generated events.

Definition at line 448 of file mega_timer16.c.

4.123.4 Variable Documentation

4.123.4.1 AVRPeripheral stTimer16

Initial value:

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

Definition at line 580 of file mega_timer16.c.

4.123.4.2 AVRPeripheral stTimer16a

Initial value:

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

Definition at line 592 of file mega_timer16.c.

4.123.4.3 AVRPeripheral stTimer16b

Initial value:

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

Definition at line 604 of file mega_timer16.c.

4.124 mega_timer16.c

```
/****************************************************************************
 *     (     (                     (     |
 *    )\ )  )\ )    (             )\ )   |
 *   (()/( (()/(   )\     (   (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))    | -- [ Litle ] -------------------
 *   (_))_|(_))  )\ _ )\ ((_)((_)(_))   | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \  | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /  | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\  |
 *                                       | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#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"

#define DEBUG_PRINT(...)

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
typedef enum
{
    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
```

```
} ClockSource_t;

//---------------------------------------------------------------------------
typedef enum
{
    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
} WaveformGeneratorMode_t;

//---------------------------------------------------------------------------
typedef enum
{
    COM_NORMAL,         // OCA1/B disconnected
    COM_TOGGLE_MATCH,   // Toggle on match
    COM_CLEAR_MATCH,
    COM_SET_MATCH
} CompareOutputMode_t;

//---------------------------------------------------------------------------
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;  // The 8-bit temporary register used in 16-bit register accesses
static uint16_t u8Count; // Internal 16-bit count register

//---------------------------------------------------------------------------
static void TCNT1_Increment()
{
    uint16_t u16NewVal = 0;

    u16NewVal = (stCPU.pstRAM->stRegisters.TCNT1H << 8 ) |
                 stCPU.pstRAM->stRegisters.TCNT1L;

    u16NewVal++;
    stCPU.pstRAM->stRegisters.TCNT1L = (u16NewVal & 0x00FF);
    stCPU.pstRAM->stRegisters.TCNT1H = (u16NewVal >> 8);
}

//---------------------------------------------------------------------------
static uint16_t TCNT1_Read()
{
    uint16_t u16Ret = 0;

    u16Ret = (stCPU.pstRAM->stRegisters.TCNT1H << 8 ) |
              stCPU.pstRAM->stRegisters.TCNT1L;
    return u16Ret;
}

//---------------------------------------------------------------------------
static void TCNT1_Clear()
{
    stCPU.pstRAM->stRegisters.TCNT1H = 0;
    stCPU.pstRAM->stRegisters.TCNT1L = 0;
}

//---------------------------------------------------------------------------
static uint16_t OCR1A_Read()
{
    uint16_t u16Ret = 0;

    u16Ret = (stCPU.pstRAM->stRegisters.OCR1AH << 8 ) |
              stCPU.pstRAM->stRegisters.OCR1AL;
    return u16Ret;
}

//---------------------------------------------------------------------------
static uint16_t OCR1B_Read()
{
```

```
    uint16_t u16Ret = 0;

    u16Ret = (stCPU.pstRAM->stRegisters.OCR1BH << 8 ) |
              stCPU.pstRAM->stRegisters.OCR1BL;
    return u16Ret;
}

//---------------------------------------------------------------------------
static uint16_t ICR1_Read()
{
    uint16_t u16Ret = 0;

    u16Ret = (stCPU.pstRAM->stRegisters.ICR1H << 8 ) |
              stCPU.pstRAM->stRegisters.ICR1L;
    return u16Ret;
}

//---------------------------------------------------------------------------
static bool Timer16_Is_TOIE1_Enabled()
{
    return (stCPU.pstRAM->stRegisters.TIMSK1.TOIE1 == 1);
}

//---------------------------------------------------------------------------
static bool Timer16_Is_OCIE1A_Enabled()
{
    return (stCPU.pstRAM->stRegisters.TIMSK1.OCIE1A == 1);
}

//---------------------------------------------------------------------------
static bool Timer16_Is_OCIE1B_Enabled()
{
    return (stCPU.pstRAM->stRegisters.TIMSK1.OCIE1B == 1);
}

//---------------------------------------------------------------------------
static bool Timer16_Is_ICIE1_Enabled()
{
    return (stCPU.pstRAM->stRegisters.TIMSK1.ICIE1 == 1);
}

//---------------------------------------------------------------------------
static void OV1_Ack(  uint8_t ucVector_)
{
    stCPU.pstRAM->stRegisters.TIFR1.TOV1 = 0;
}

//---------------------------------------------------------------------------
static void IC1_Ack(  uint8_t ucVector_)
{
    stCPU.pstRAM->stRegisters.TIFR1.ICF1 = 0;
}

//---------------------------------------------------------------------------
static void COMP1A_Ack(  uint8_t ucVector_)
{
    static uint64_t lastcycles = 0;
   // printf("COMP1A - Ack'd: %d delta\n", stCPU.u64CycleCount - lastcycles);
    lastcycles = stCPU.u64CycleCount;

    stCPU.pstRAM->stRegisters.TIFR1.OCF1A = 0;
}

//---------------------------------------------------------------------------
static void COMP1B_Ack(  uint8_t ucVector_)
{
    stCPU.pstRAM->stRegisters.TIFR1.OCF1B = 0;
}

//---------------------------------------------------------------------------
static void Timer16_Init(void *context_ )
{
    DEBUG_PRINT(stderr, "Timer16 Init\n");

    CPU_RegisterInterruptCallback( OV1_Ack, 0x0D);
    CPU_RegisterInterruptCallback( IC1_Ack, 0x0A);
    CPU_RegisterInterruptCallback( COMP1A_Ack, 0x0B);
    CPU_RegisterInterruptCallback( COMP1B_Ack, 0x0C);
}

//---------------------------------------------------------------------------
static void Timer16_Read(void *context_, uint8_t ucAddr_, uint8_t *pucValue_ )
{
    DEBUG_PRINT(stderr, "Timer16 Read: 0x%02x\n", ucAddr_);
    *pucValue_ = stCPU.pstRAM->au8RAM[ ucAddr_ ];
}

```

```
//---------------------------------------------------------------------------
static void TCCR1A_Write( uint8_t ucAddr_, uint8_t ucValue_)
{
    // Update the waveform generator mode (WGM11:10) bits.
    uint8_t u8WGMBits = ucValue_ & 0x03; // WGM11 and 10 are in bits 0,1
    uint8_t u8WGMTemp = (uint8_t)eWGM;
    u8WGMTemp &= ~(0x03);
    u8WGMTemp |= u8WGMBits;
    eWGM = (WaveformGeneratorMode_t)u8WGMTemp;

    // Update the memory-mapped register.
    stCPU.pstRAM->stRegisters.TCCR1A.r = ucValue_ & 0xF3;
}

//---------------------------------------------------------------------------
static void TCCR1B_Write( uint8_t ucAddr_, uint8_t ucValue_)
{
    // Update the waveform generator mode (WGM13:12) bits.
    uint8_t u8WGMBits = (ucValue_ >> 1) & 0x0C; // WGM13 and 12 are in register bits 3,4
    uint8_t u8WGMTemp = (uint8_t)eWGM;
    u8WGMTemp &= ~(0x0C);
    u8WGMTemp |= u8WGMBits;
    eWGM = (WaveformGeneratorMode_t)u8WGMTemp;

    // Update the clock-select bits
    uint8_t u8ClockSource = ucValue_ & 0x07; // clock select is last 3 bits in reg
    eClockSource = (ClockSource_t)u8ClockSource;
    switch (eClockSource)
    {
    case CLK_SRC_DIV_1:
        u16DivCycles = 1;
        break;
    case CLK_SRC_DIV_8:
        u16DivCycles = 8;
        break;
    case CLK_SRC_DIV_64:
        u16DivCycles = 64;
        break;
    case CLK_SRC_DIV_256:
        u16DivCycles = 256;
        break;
    case CLK_SRC_DIV_1024:
        u16DivCycles = 1024;
        break;
    default:
        u16DivCycles = 0;
        break;
    }

    // Update the memory-mapped register.
    stCPU.pstRAM->stRegisters.TCCR1B.r = ucValue_ & 0xDF; // Bit 5 is read-only
}

//---------------------------------------------------------------------------
static void TCCR1C_Write( uint8_t ucAddr_, uint8_t ucValue_)
{
    stCPU.pstRAM->stRegisters.TCCR1C.r = ucValue_;
}

//---------------------------------------------------------------------------
static void TCNT1L_Write( uint8_t ucAddr_, uint8_t ucValue_)
{
    // Writing the low-word forces the high-word to be stored from the internal
    // temp register... which is why the high byte must be written first.
    stCPU.pstRAM->stRegisters.TCNT1L = ucValue_;
    stCPU.pstRAM->stRegisters.TCNT1H = u8Temp;
}
//---------------------------------------------------------------------------
static void TCNT1H_Write( uint8_t ucAddr_, uint8_t ucValue_)
{
    u8Temp = ucValue_;
}
//---------------------------------------------------------------------------
static void ICR1L_Write( uint8_t ucAddr_, uint8_t ucValue_)
{
    // Writing the low-word forces the high-word to be stored from the internal
    // temp register... which is why the high byte must be written first.
    stCPU.pstRAM->stRegisters.ICR1L = ucValue_;
    stCPU.pstRAM->stRegisters.ICR1H = u8Temp;
}
//---------------------------------------------------------------------------
static void ICR1H_Write( uint8_t ucAddr_, uint8_t ucValue_)
{
    u8Temp = ucValue_;
}

//---------------------------------------------------------------------------
```

```
static void OCR1AL_Write( uint8_t ucAddr_, uint8_t ucValue_)
{
    // Writing the low-word forces the high-word to be stored from the internal
    // temp register... which is why the high byte must be written first.
    stCPU.pstRAM->stRegisters.OCR1AL = ucValue_;
    stCPU.pstRAM->stRegisters.OCR1AH = u8Temp;
}

//---------------------------------------------------------------------------
static void OCR1AH_Write( uint8_t ucAddr_, uint8_t ucValue_)
{
    u8Temp = ucValue_;
}

//---------------------------------------------------------------------------
static void OCR1BL_Write( uint8_t ucAddr_, uint8_t ucValue_)
{
    // Writing the low-word forces the high-word to be stored from the internal
    // temp register... which is why the high byte must be written first.
    stCPU.pstRAM->stRegisters.OCR1BL = ucValue_;
    stCPU.pstRAM->stRegisters.OCR1BH = u8Temp;
}

//---------------------------------------------------------------------------
static void OCR1BH_Write( uint8_t ucAddr_, uint8_t ucValue_)
{
    u8Temp = ucValue_;
}

//---------------------------------------------------------------------------
static void Timer16_IntFlagUpdate(void)
{
    if (stCPU.pstRAM->stRegisters.TIMSK1.TOIE1 == 1)
    {
        if (stCPU.pstRAM->stRegisters.TIFR1.TOV1 == 1)
        {
            DEBUG_PRINT(" TOV1 Interrupt Candidate\n" );
            AVR_InterruptCandidate(0x0D);
        }
        else
        {
            AVR_ClearCandidate(0x0D);
        }
    }

    if (stCPU.pstRAM->stRegisters.TIMSK1.OCIE1A == 1)
    {
        if (stCPU.pstRAM->stRegisters.TIFR1.OCF1A == 1)
        {
            DEBUG_PRINT(" OCF1A Interrupt Candidate\n" );
            AVR_InterruptCandidate(0x0B);
        }
        else
        {
            AVR_ClearCandidate(0x0B);
        }
    }

    if (stCPU.pstRAM->stRegisters.TIMSK1.OCIE1B == 1)
    {
        if (stCPU.pstRAM->stRegisters.TIFR1.OCF1B == 1)
        {
            DEBUG_PRINT(" OCF1B Interrupt Candidate\n" );
            AVR_InterruptCandidate(0x0C);
        }
        else
        {
            AVR_ClearCandidate(0x0C);
        }
    }

    if (stCPU.pstRAM->stRegisters.TIMSK1.ICIE1 == 1)
    {
        if (stCPU.pstRAM->stRegisters.TIFR1.ICF1 == 1)
        {
            DEBUG_PRINT(" ICF1 Interrupt Candidate\n" );
            AVR_InterruptCandidate(0x0A);
        }
        else
        {
            AVR_ClearCandidate(0x0A);
        }
    }
}

//---------------------------------------------------------------------------
// TIFR & TMSK
```

```
static void Timer16b_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
{
    stCPU.pstRAM->au8RAM[ucAddr_] = ucValue_;
    Timer16_IntFlagUpdate();
}

//---------------------------------------------------------------------------
static void Timer16_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
{
    switch (ucAddr_)
    {
    case 0x80:  //TCCR1A
        TCCR1A_Write(ucAddr_, ucValue_);
        break;
    case 0x81:  //TCCR1B
        TCCR1B_Write(ucAddr_, ucValue_);
        break;
    case 0x82:  //TCCR1C
        TCCR1C_Write(ucAddr_, ucValue_);
        break;
    case 0x83:  // Reserved
        break;
    case 0x84:  // TCNT1L
        TCNT1L_Write(ucAddr_, ucValue_);
        break;
    case 0x85:  // TCNT1H
        TCNT1H_Write(ucAddr_, ucValue_);
        break;
    case 0x86:  // ICR1L
        ICR1L_Write(ucAddr_, ucValue_);
        break;
    case 0x87:  // ICR1H
        ICR1H_Write(ucAddr_, ucValue_);
        break;
    case 0x88:  // OCR1AL
        OCR1AL_Write(ucAddr_, ucValue_);
        break;
    case 0x89:  // OCR1AH
        OCR1AH_Write(ucAddr_, ucValue_);
        break;
    case 0x8A:  // OCR1BL
        OCR1BL_Write(ucAddr_, ucValue_);
        break;
    case 0x8B:  // OCR1BH
        OCR1BH_Write(ucAddr_, ucValue_);
        break;
    default:
        break;
    }
}

//---------------------------------------------------------------------------
static void Timer16_Clock(void *context_ )
{
    if (eClockSource == CLK_SRC_OFF)
    {
        return;
    }

    // Handle clock division logic
    bool bUpdateTimer = false;
    switch (eClockSource)
    {
    case CLK_SRC_DIV_1:
    case CLK_SRC_DIV_8:
    case CLK_SRC_DIV_64:
    case CLK_SRC_DIV_256:
    case CLK_SRC_DIV_1024:
    {
        // Decrement the clock-divide value
        if (u16DivRemain)
        {
            //DEBUG_PRINT(" %d ticks remain\n", u16DivRemain);
            u16DivRemain--;
        }

        if (!u16DivRemain)
        {
            // clock-divider count hits zero, reset and trigger an update.
            //DEBUG_PRINT(" expire and reset\n");
            if (u16DivCycles)
            {
                u16DivRemain = u16DivCycles;
                bUpdateTimer = true;
            }
        }
    }
```

```
            break;
        default:
            break;
        }


        if (bUpdateTimer)
        {
            // Handle event flags on timer updates
            bool bOVF    = false;
            bool bCTCA   = false;
            bool bCTCB   = false;
            bool bICR    = false;
            bool bIntr   = false;

            //DEBUG_PRINT( " WGM Mode %d\n", eWGM );
            switch (eWGM)
            {
            case WGM_NORMAL:
            {
                DEBUG_PRINT(" Update Normal\n");
                TCNT1_Increment();
                if (TCNT1_Read() == 0)
                {
                    bOVF = true;
                }
            }
                break;
            case WGM_CTC_OCR:
            {
                DEBUG_PRINT(" Update CTC\n");
                TCNT1_Increment();
                if (TCNT1_Read() == 0)
                {
                    bOVF = true;
                }
                else
                {
                    bool bClearTCNT1 = false;
                    if (TCNT1_Read() == OCR1A_Read())
                    {
                        DEBUG_PRINT(" CTC1A Match\n" );
                        bCTCA = true;
                        bClearTCNT1 = true;
                    }
                    if (TCNT1_Read() == ICR1_Read())
                    {
                        DEBUG_PRINT(" ICR1 Match\n" );
                        bICR = true;
                        bClearTCNT1 = true;
                    }
                    if (bClearTCNT1)
                    {
                        TCNT1_Clear();
                    }
                }
            }
                break;
            default:
                break;
            }

            // Set interrupt flags if an appropriate transition has taken place
            if (bOVF)
            {
                DEBUG_PRINT(" TOV1 Set\n" );
                stCPU.pstRAM->stRegisters.TIFR1.TOV1 = 1;
                bIntr = true;
            }
            if (bCTCA)
            {
                DEBUG_PRINT(" OCF1A Set\n" );
                stCPU.pstRAM->stRegisters.TIFR1.OCF1A = 1;
                bIntr = true;
            }
            if (bCTCB)
            {
                DEBUG_PRINT(" OCF1B Set\n" );
                stCPU.pstRAM->stRegisters.TIFR1.OCF1B = 1;
                bIntr = true;
            }
            if (bICR)
            {
                DEBUG_PRINT(" ICF1 Set\n" );
                stCPU.pstRAM->stRegisters.TIFR1.ICF1 = 1;
                bIntr = true;
            }
```

```
        if (bIntr)
        {
            Timer16_IntFlagUpdate();
        }
    }
}

//---------------------------------------------------------------------------
AVRPeripheral stTimer16 =
{
    Timer16_Init,
    Timer16_Read,
    Timer16_Write,
    Timer16_Clock,
    0,
    0x80,
    0x8B
};

//---------------------------------------------------------------------------
AVRPeripheral stTimer16a =
{
    0,
    Timer16_Read,
    Timer16b_Write,
    0,
    0,
    0x36,
    0x36
};

//---------------------------------------------------------------------------
AVRPeripheral stTimer16b =
{
    0,
    Timer16_Read,
    Timer16b_Write,
    0,
    0,
    0x6F,
    0x6F
};
```

4.125 mega_timer16.h File Reference

ATMega 16-bit timer implementation.

`#include "avr_peripheral.h"`

Variables

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

4.125.1 Detailed Description

ATMega 16-bit timer implementation.

Definition in file mega_timer16.h.

4.126 mega_timer16.h

```
/****************************************************************************
 *     (     (                      (     |
 *    )\ )  )\ )      (              )\ )    |
 *   (()/( (()/(      )\      (   (   (()/(    | -- [ Funkenstein ] -------------
 *    /(_))  /(_))(((_)()\    )\   /(_))      | -- [ Litle ] -------------------
```

```
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                        | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __MEGA_TIMER16_H__
#define __MEGA_TIMER16_H__

#include "avr_peripheral.h"

extern AVRPeripheral stTimer16;
extern AVRPeripheral stTimer16a;
extern AVRPeripheral stTimer16b;

#endif //__MEGA_EINT_H__
```

4.127 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**

4.127.1 Detailed Description

ATMega 8-bit timer implementation.

Definition in file mega_timer8.c.

4.127.2 Enumeration Type Documentation

4.127.2.1 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.

4.127.3 Function Documentation

4.127.3.1 static void Timer8_Clock (void ∗ *context_*) [static]

! ToDo - Handle external timer generated events.

Definition at line 315 of file mega_timer8.c.

4.127.4 Variable Documentation

4.127.4.1 AVRPeripheral stTimer8

Initial value:

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

Definition at line 428 of file mega_timer8.c.

4.127.4.2 AVRPeripheral stTimer8a

Initial value:

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

Definition at line 441 of file mega_timer8.c.

4.127.4.3 AVRPeripheral stTimer8b

Initial value:

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

Definition at line 453 of file mega_timer8.c.

4.128 mega_timer8.c

```
/*****************************************************************************
 *     (     (                      (     |
 *    )\ )  )\ )    (             )\ )   |
 *   (()/( (()/(    )\     (   ( (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))    | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))   | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \  | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /  | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\  |
 *                                       | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
```

```
 ****************************************************************************/
#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"

#define DEBUG_PRINT(...)

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
typedef enum
{
    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
} ClockSource_t;

//---------------------------------------------------------------------------
typedef enum
{
    WGM_NORMAL,
    WGM_PWM_PC_FF,
    WGM_CTC_OCR,
    WGM_FAST_PWM_FF,
    WGM_RESERVED_1, // Not a valid mode
    WGM_PWM_PC_OCR,
    WGM_RESERVED_2, // Not a valid mode
    WGM_FAST_PWM_OCR
} WaveformGeneratorMode_t;

//---------------------------------------------------------------------------
typedef enum
{
    COM_NORMAL,         // OCA
    COM_TOGGLE_MATCH,   // Toggle on match
    COM_CLEAR_MATCH,
    COM_SET_MATCH
} CompareOutputMode_t;

//---------------------------------------------------------------------------
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;  // The 8-bit temporary register used in 16-bit register accesses
static uint16_t u8Count; // Internal 16-bit count register

//---------------------------------------------------------------------------
static void TCNT0_Increment()
{
    stCPU.pstRAM->stRegisters.TCNT0++;
}

//---------------------------------------------------------------------------
static uint8_t TCNT0_Read()
{
    return stCPU.pstRAM->stRegisters.TCNT0;
}

//---------------------------------------------------------------------------
static void TCNT0_Clear()
{
    stCPU.pstRAM->stRegisters.TCNT0 = 0;
}

//---------------------------------------------------------------------------
static uint8_t OCR0A_Read()
{
    return stCPU.pstRAM->stRegisters.OCR0A;
}

//---------------------------------------------------------------------------
static uint8_t OCR0B_Read()
```

```
{
    return stCPU.pstRAM->stRegisters.OCR0B;
}

//---------------------------------------------------------------------------
static bool Timer8_Is_TOIE0_Enabled()
{
    return (stCPU.pstRAM->stRegisters.TIMSK0.TOIE0 == 1);
}

//---------------------------------------------------------------------------
static bool Timer8_Is_OCIE0A_Enabled()
{
    return (stCPU.pstRAM->stRegisters.TIMSK0.OCIE0A == 1);
}

//---------------------------------------------------------------------------
static bool Timer8_Is_OCIE1B_Enabled()
{
    return (stCPU.pstRAM->stRegisters.TIMSK0.OCIE0B == 1);
}

//---------------------------------------------------------------------------
static void OV0_Ack(  uint8_t ucVector_)
{
    static uint64_t lastcycles = 0;
    stCPU.pstRAM->stRegisters.TIFR0.TOV0 = 0;
   // printf("OV0 - Ack'd: %d delta\n", stCPU.u64CycleCount - lastcycles);
    lastcycles = stCPU.u64CycleCount;
}

//---------------------------------------------------------------------------
static void COMP0A_Ack(  uint8_t ucVector_)
{
    stCPU.pstRAM->stRegisters.TIFR0.OCF0A = 0;
}

//---------------------------------------------------------------------------
static void COMP0B_Ack(  uint8_t ucVector_)
{
    stCPU.pstRAM->stRegisters.TIFR0.OCF0B = 0;
}

//---------------------------------------------------------------------------
static void Timer8_Init(void *context_ )
{
    DEBUG_PRINT( "Timer8 Init\n");
    CPU_RegisterInterruptCallback( OV0_Ack, 0x10);
    CPU_RegisterInterruptCallback( COMP0A_Ack, 0x0E);
    CPU_RegisterInterruptCallback( COMP0B_Ack, 0x0F);
}

//---------------------------------------------------------------------------
static void Timer8_Read(void *context_, uint8_t ucAddr_, uint8_t *pucValue_ )
{
    DEBUG_PRINT( "Timer8 Read: 0x%02x\n", ucAddr_);
    *pucValue_ = stCPU.pstRAM->au8RAM[ ucAddr_ ];
}

//---------------------------------------------------------------------------
static void TCCR0A_Write( uint8_t ucAddr_, uint8_t ucValue_)
{
    // Update the waveform generator mode (WGM1:0) bits.
    uint8_t u8WGMBits = ucValue_ & 0x03; // WGM1 and 0 are in bits 0,1
    uint8_t u8WGMTemp = (uint8_t)eWGM;
    u8WGMTemp &= ~(0x03);
    u8WGMTemp |= u8WGMBits;
    eWGM = (WaveformGeneratorMode_t)u8WGMTemp;

    // Update the memory-mapped register.
    stCPU.pstRAM->stRegisters.TCCR0A.r = ucValue_ & 0xF3;
}

//---------------------------------------------------------------------------
static void TCCR0B_Write( uint8_t ucAddr_, uint8_t ucValue_)
{
    // Update the waveform generator mode (WGM2) bit
    uint8_t u8WGMBits = (ucValue_ >> 1) & 0x04; // WGM2 is in bit 3 of the register
    uint8_t u8WGMTemp = (uint8_t)eWGM;
    u8WGMTemp &= ~(0x04);
    u8WGMTemp |= u8WGMBits;
    eWGM = (WaveformGeneratorMode_t)u8WGMTemp;

    // Update the clock-select bits
    uint8_t u8ClockSource = ucValue_ & 0x07; // clock select is last 3 bits in reg
    eClockSource = (ClockSource_t)u8ClockSource;
    switch (eClockSource)
```

```
    {
    case CLK_SRC_DIV_1:
        u16DivCycles = 1;
        break;
    case CLK_SRC_DIV_8:
        u16DivCycles = 8;
        break;
    case CLK_SRC_DIV_64:
        u16DivCycles = 64;
        break;
    case CLK_SRC_DIV_256:
        u16DivCycles = 256;
        break;
    case CLK_SRC_DIV_1024:
        u16DivCycles = 1024;
        break;
    default:
        u16DivCycles = 0;
        break;
    }
    DEBUG_PRINT(" ClockSource = %d, %d cycles\n", eClockSource, u16DivCycles);
    // Update the memory-mapped register.
    stCPU.pstRAM->stRegisters.TCCR0B.r = ucValue_ & 0xCF; // Bit 5&6 are read-only
}

//---------------------------------------------------------------------------
static void TCNT0_Write( uint8_t ucAddr_, uint8_t ucValue_)
{
    stCPU.pstRAM->stRegisters.TCNT0 = ucValue_;
}

//---------------------------------------------------------------------------
static void OCR0A_Write( uint8_t ucAddr_, uint8_t ucValue_)
{
    stCPU.pstRAM->stRegisters.OCR0A = ucValue_;
}

//---------------------------------------------------------------------------
static void OCR0B_Write( uint8_t ucAddr_, uint8_t ucValue_)
{
    stCPU.pstRAM->stRegisters.OCR0B = ucValue_;
}

//---------------------------------------------------------------------------
static void Timer8_IntFlagUpdate(void)
{
    if (stCPU.pstRAM->stRegisters.TIMSK0.TOIE0 == 1)
    {
        if (stCPU.pstRAM->stRegisters.TIFR0.TOV0 == 1)
        {
            DEBUG_PRINT(" TOV0 Interrupt Candidate\n" );
            AVR_InterruptCandidate(0x10);
        }
        else
        {
            AVR_ClearCandidate(0x10);
        }
    }
    if (stCPU.pstRAM->stRegisters.TIMSK0.OCIE0A == 1)
    {
        if (stCPU.pstRAM->stRegisters.TIFR0.OCF0A == 1)
        {
            DEBUG_PRINT(" OCF0A Interrupt Candidate\n" );
            AVR_InterruptCandidate(0x0E);
        }
        else
        {
            AVR_ClearCandidate(0x0E);
        }
    }
    if (stCPU.pstRAM->stRegisters.TIMSK0.OCIE0B == 1)
    {
        if (stCPU.pstRAM->stRegisters.TIFR0.OCF0B == 1)
        {
            DEBUG_PRINT(" OCF0B Interrupt Candidate\n" );
            AVR_InterruptCandidate(0x0F);
        }
        else
        {
            AVR_ClearCandidate(0x0F);
        }
    }
}

//---------------------------------------------------------------------------
static void Timer8b_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
{
```

```
    stCPU.pstRAM->au8RAM[ucAddr_] = ucValue_;
    Timer8_IntFlagUpdate();
}

//---------------------------------------------------------------------------
static void Timer8_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
{
    DEBUG_PRINT("Timer8_Write: %d=%d\n", ucAddr_, ucValue_);
    switch (ucAddr_)
    {
    case 0x44:  //TCCR1A
        TCCR0A_Write(ucAddr_, ucValue_);
        break;
    case 0x45:  //TCCR1B
        TCCR0B_Write(ucAddr_, ucValue_);
        break;
    case 0x46:  // TCNT0
        TCNT0_Write(ucAddr_, ucValue_);
        break;
    case 0x47:  // OCR0A
        OCR0A_Write(ucAddr_, ucValue_);
        break;
    case 0x48:  // OCR0B
        OCR0B_Write(ucAddr_, ucValue_);
        break;
    default:
        break;
    }
}

//---------------------------------------------------------------------------
static void Timer8_Clock(void *context_ )
{
    if (eClockSource == CLK_SRC_OFF)
    {
        return;
    }

    // Handle clock division logic
    bool bUpdateTimer = false;
    switch (eClockSource)
    {
    case CLK_SRC_DIV_1:
    case CLK_SRC_DIV_8:
    case CLK_SRC_DIV_64:
    case CLK_SRC_DIV_256:
    case CLK_SRC_DIV_1024:
    {
        // Decrement the clock-divide value
        if (u16DivRemain)
        {
            //DEBUG_PRINT(" %d ticks remain\n", u16DivRemain);
            u16DivRemain--;
        }

        if (!u16DivRemain)
        {
            // clock-divider count hits zero, reset and trigger an update.
            DEBUG_PRINT(" expire and reset\n");
            if (u16DivCycles)
            {
                u16DivRemain = u16DivCycles;
                bUpdateTimer = true;
            }
        }
    }
        break;
    default:
        break;
    }


    if (bUpdateTimer)
    {
        // Handle event flags on timer updates
        bool bOVF   = false;
        bool bCTCA  = false;
        bool bCTCB  = false;
        bool bIntr  = false;

        switch (eWGM)
        {
        case WGM_NORMAL:
        {
            DEBUG_PRINT(" Update Normal, TCNT = %d\n", TCNT0_Read());
            TCNT0_Increment();
            if (TCNT0_Read() == 0)
```

```
            {
                bOVF = true;
            }
        }
            break;
        case WGM_CTC_OCR:
        {
            DEBUG_PRINT(" Update CTC\n");
            TCNT0_Increment();
            if (TCNT0_Read() == 0)
            {
                bOVF = true;
            }
            else
            {
                if (TCNT0_Read() == OCR0A_Read())
                {
                    DEBUG_PRINT(" CTC0A Match\n" );
                    bCTCA = true;
                    TCNT0_Clear();
                }
            }
        }
            break;
        default:
            break;
        }

        // Set interrupt flags if an appropriate transition has taken place
        if (bOVF)
        {
            DEBUG_PRINT(" TOV0 Set\n" );
            stCPU.pstRAM->stRegisters.TIFR0.TOV0 = 1;
            bIntr = true;
        }
        if (bCTCA)
        {
            DEBUG_PRINT(" OCF0A Set\n" );
            stCPU.pstRAM->stRegisters.TIFR0.OCF0A = 1;
            bIntr = true;
        }
        if (bCTCB)
        {
            DEBUG_PRINT(" OCF0B Set\n" );
            stCPU.pstRAM->stRegisters.TIFR0.OCF0B = 1;
            bIntr = true;
        }

        if (bIntr)
        {
            Timer8_IntFlagUpdate();
        }
    }
}

//---------------------------------------------------------------------------
AVRPeripheral stTimer8 =
{
    Timer8_Init,
    Timer8_Read,
    Timer8_Write,
    Timer8_Clock,
    0,
    0x44,
    0x48
};


//---------------------------------------------------------------------------
AVRPeripheral stTimer8a =
{
    0,
    Timer8_Read,
    Timer8b_Write,
    0,
    0,
    0x35,
    0x35
};

//---------------------------------------------------------------------------
AVRPeripheral stTimer8b =
{
    0,
    Timer8_Read,
    Timer8b_Write,
    0,
```

```
    0,
    0x6E,
    0x6E
};
```

4.129 mega_timer8.h File Reference

ATMega 8-bit timer implementation.

`#include "avr_peripheral.h"`

Variables

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

4.129.1 Detailed Description

ATMega 8-bit timer implementation.

Definition in file mega_timer8.h.

4.130 mega_timer8.h

```
/*****************************************************************************
 *     (      (                        (     |
 *    )\ )   )\ )     (                 )\ )   |
 *   (()/(  (()/(    )\      (    (   (()/(   | -- [ Funkenstein ] -------------
 *    /(_))  /(_))((((_)()\   )\   /(_))      | -- [ Litle ] -------------------
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))      | -- [ AVR ] ---------------------
 *   | |_   | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *   | __|  | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
 *   |_|    |____|  /_/ \_\   \_/   |_|_\    |
 *                                           | "Yeah, it does Arduino..."
 * ------------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __MEGA_TIMER8_H__
#define __MEGA_TIMER8_H__

#include "avr_peripheral.h"

extern AVRPeripheral stTimer8;
extern AVRPeripheral stTimer8a;
extern AVRPeripheral stTimer8b;

#endif //__MEGA_EINT_H__
```

4.131 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**

4.131.1 Detailed Description

Implements an atmega UART plugin.

Definition in file mega_uart.c.

4.131.2 Macro Definition Documentation

4.131.2.1 #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.

4.131.3 Variable Documentation

4.131.3.1 AVRPeripheral stUART

Initial value:

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

Definition at line 436 of file mega_uart.c.

4.132 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 2014-15, 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
```

```
static uint8_t TXB = 0; // transmit buffer
static uint8_t TSR = 0; // transmit shift register.
static uint8_t RSR = 0; // receive shift register.

static uint32_t u32BaudTicks = 0;
static uint32_t u32TxTicksRemaining = 0;
static uint32_t u32RxTicksRemaining = 0;

//---------------------------------------------------------------------------
static void Echo_Tx()
{
    printf("%c", TSR);
}

//---------------------------------------------------------------------------
static void Echo_Rx()
{
    printf("%c", RSR);
}

//---------------------------------------------------------------------------
static bool UART_IsRxEnabled( void )
{
    //DEBUG_PRINT( "RxEnabled\n");
    return (stCPU.pstRAM->stRegisters.UCSR0B.RXEN0 == 1);
}

//---------------------------------------------------------------------------
static bool UART_IsTxEnabled( void )
{
    //DEBUG_PRINT( "TxEnabled\n");
    return (stCPU.pstRAM->stRegisters.UCSR0B.TXEN0 == 1);
}

//---------------------------------------------------------------------------
static bool UART_IsTxIntEnabled( void )
{
    return (stCPU.pstRAM->stRegisters.UCSR0B.TXCIE0 == 1);
}

//---------------------------------------------------------------------------
static bool UART_IsDREIntEnabled( void )
{
    return (stCPU.pstRAM->stRegisters.UCSR0B.UDRIE0 == 1);
}

//---------------------------------------------------------------------------
static bool UART_IsRxIntEnabled( void )
{
    return (stCPU.pstRAM->stRegisters.UCSR0B.RXCIE0 == 1);
}

//---------------------------------------------------------------------------
static bool UART_IsDoubleSpeed()
{
    return (stCPU.pstRAM->stRegisters.UCSR0A.U2X0 == 1);
}

//---------------------------------------------------------------------------
static void UART_SetDoubleSpeed()
{
    stCPU.pstRAM->stRegisters.UCSR0A.U2X0 = 1;
}

//---------------------------------------------------------------------------
static void UART_SetEmpty( void )
{
    stCPU.pstRAM->stRegisters.UCSR0A.UDRE0 = 1;
}

//---------------------------------------------------------------------------
static void UART_ClearEmpty( void )
{
    stCPU.pstRAM->stRegisters.UCSR0A.UDRE0 = 0;
}

//---------------------------------------------------------------------------
static bool UART_IsEmpty( void )
{
    return (stCPU.pstRAM->stRegisters.UCSR0A.UDRE0 == 1);
}

//---------------------------------------------------------------------------
static bool UART_IsTxComplete( void )
{
    return (stCPU.pstRAM->stRegisters.UCSR0A.TXC0 == 1);
}
```

```
//---------------------------------------------------------------------------
static void UART_TxComplete( void )
{
    stCPU.pstRAM->stRegisters.UCSR0A.TXC0 = 1;
}

//---------------------------------------------------------------------------
static bool UART_IsRxComplete( void )
{
    return (stCPU.pstRAM->stRegisters.UCSR0A.RXC0 == 1);
}

//---------------------------------------------------------------------------
static void UART_RxComplete( void )
{
    stCPU.pstRAM->stRegisters.UCSR0A.RXC0 = 1;
}

//---------------------------------------------------------------------------
static void TXC0_Callback(  uint8_t ucVector_ )
{
    // On TX Complete interrupt, automatically clear the TXC0 flag.
    stCPU.pstRAM->stRegisters.UCSR0A.TXC0 = 0;
}

//---------------------------------------------------------------------------
static void UART_Init(void *context_ )
{
    DEBUG_PRINT("UART Init\n");
    stCPU.pstRAM->stRegisters.UCSR0A.UDRE0 = 1;

    CPU_RegisterInterruptCallback( TXC0_Callback, 0x14); // TX Complete
}

//---------------------------------------------------------------------------
static void UART_Read(void *context_, uint8_t ucAddr_, uint8_t *pucValue_ )
{
    DEBUG_PRINT( "UART Read: 0x%02x == 0x%02X\n", ucAddr_, stCPU.pstRAM->au8RAM[ ucAddr_ ]);
    *pucValue_ = stCPU.pstRAM->au8RAM[ ucAddr_ ];
    switch (ucAddr_)
    {
        case 0xC6: // UDR0
            stCPU.pstRAM->stRegisters.UCSR0A.RXC0 = 0;
            break;
        default:
            break;
    }
}

//---------------------------------------------------------------------------
static void UART_WriteBaudReg()
{
    DEBUG_PRINT( "WriteBaud\n");
    uint16_t u16Baud =  (uint16_t)(stCPU.pstRAM->stRegisters.UBRR0L) |
                        ((uint16_t)(stCPU.pstRAM->stRegisters.UBRR0H) << 8);

    u32BaudTicks = u16Baud;
}

//---------------------------------------------------------------------------
static void UART_WriteDataReg()
{
    DEBUG_PRINT("UART Write UDR...\n");
    if (UART_IsTxEnabled())
    {
        DEBUG_PRINT("Enabled...\n");
        // Only set the baud timer if the UART is idle
        if (!u32TxTicksRemaining)
        {
            u32TxTicksRemaining = u32BaudTicks;
            if (UART_IsDoubleSpeed())
            {
                u32TxTicksRemaining >>= 1;
            }
        }

        // If the shift register is empty, load it immediately
        if (bTSR_Empty)
        {
            TSR = stCPU.pstRAM->stRegisters.UDR0;
            TXB = 0;
            bTSR_Empty = false;
            bUDR_Empty = true;
            UART_SetEmpty();

            if (UART_IsDREIntEnabled())
```

```
            {
                DEBUG_PRINT("DRE Interrupt\n");
                AVR_InterruptCandidate( 0x13 );
            }
        }
        else
        {
            TXB = stCPU.pstRAM->stRegisters.UDR0;
            bTSR_Empty = false;
            bUDR_Empty = false;
            UART_ClearEmpty();
        }
    }
    else
    {
        DEBUG_PRINT("Disabled...\n");
    }
}

//---------------------------------------------------------------------------
static void UART_WriteUCSR0A( uint8_t u8Value_)
{
    DEBUG_PRINT("UART Write UCSR0A...\n");
    uint8_t u8Reg = stCPU.pstRAM->stRegisters.UCSR0A.r;
    if (u8Value_ & 0x40) // TXC was set explicitly -- clear it in the SR.
    {
        u8Reg &= ~0x40;
    }
    u8Reg &= ~(0xBC);

    stCPU.pstRAM->stRegisters.UCSR0A.r |= u8Reg;

}

//---------------------------------------------------------------------------
static void UART_UpdateInterruptFlags(void)
{
    //DEBUG_PRINT("Check UART Interrupts\n");
    if (UART_IsTxIntEnabled())
    {
        if (UART_IsTxComplete())
        {
            DEBUG_PRINT("Enable TXC Interrupt\n");
            AVR_InterruptCandidate( 0x14 );
        }
        else
        {
            DEBUG_PRINT("Clear TXC Interrupt\n");
            AVR_ClearCandidate( 0x14 );
        }
    }
    if (UART_IsDREIntEnabled())
    {
        if( UART_IsEmpty())
        {
            DEBUG_PRINT("Enable DRE Interrupt\n");
            AVR_InterruptCandidate( 0x13 );
        }
        else
        {
            DEBUG_PRINT("Clear DRE Interrupt\n");
            AVR_ClearCandidate( 0x13 );
        }
    }
    if (UART_IsRxIntEnabled())
    {
        if (UART_IsRxComplete())
        {
            DEBUG_PRINT("Enable RXC Interrupt\n");
            AVR_InterruptCandidate( 0x12 );
        }
        else
        {
            DEBUG_PRINT("Clear RXC Interrupt\n");
            AVR_ClearCandidate( 0x12 );
        }
    }
}

//---------------------------------------------------------------------------
static void UART_WriteUCSR0B( uint8_t u8Value_)
{
    DEBUG_PRINT("Write UCSR0B = %02x\n", u8Value_);
    stCPU.pstRAM->stRegisters.UCSR0B.r = u8Value_;
    UART_UpdateInterruptFlags();
}

```

```
//---------------------------------------------------------------------------
static void UART_WriteUCSR0C( uint8_t u8Value_)
{
    DEBUG_PRINT("Write UCRS0C\n");
    stCPU.pstRAM->stRegisters.UCSR0C.r == u8Value_;
}

//---------------------------------------------------------------------------
static void UART_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
{
    DEBUG_PRINT("UART Write: %2X=%2X\n", ucAddr_, ucValue_ );
    switch (ucAddr_)
    {
    case 0xC0:  //UCSR0A
        UART_WriteUCSR0A( ucValue_ );
        break;
    case 0xC1:  //UCSR0B
        UART_WriteUCSR0B( ucValue_ );
        break;
    case 0xC2:  //UCSR0C
        UART_WriteUCSR0C( ucValue_ );
        break;
    case 0xC3:  // NA.
        break;
    case 0xC4:  //UBRR0L
    case 0xC5:  //UBRR0H
        DEBUG_PRINT("Write UBRR0x\n");
        stCPU.pstRAM->au8RAM[ ucAddr_ ] = ucValue_;
        UART_WriteBaudReg();
        break;
    case 0xC6:  //UDR0
        DEBUG_PRINT("Write UDR0\n");
        stCPU.pstRAM->au8RAM[ ucAddr_ ] = ucValue_;
        UART_WriteDataReg();
        break;
    default:
        break;
    }
}

//---------------------------------------------------------------------------
static void UART_TxClock(void *context_ )
{
    //DEBUG_PRINT("TX clock...\n");
    if (UART_IsTxEnabled() && u32TxTicksRemaining)
    {
        DEBUG_PRINT("Countdown %d ticks remain\n", u32TxTicksRemaining);
        u32TxTicksRemaining--;
        if (!u32TxTicksRemaining)
        {
            // Local echo of the freshly "shifted out" data to the terminal
            Echo_Tx();

            // If there's something queued in the TXB, reload the TSR
            // register, flag the UDR as empty, and TSR as full.
            if (!bUDR_Empty)
            {
                TSR = TXB;
                TXB = 0;
                bUDR_Empty = true;
                bTSR_Empty = false;

                UART_SetEmpty();

                if (UART_IsDREIntEnabled())
                {
                    DEBUG_PRINT("DRE Interrupt\n");
                    AVR_InterruptCandidate( 0x13 );
                }
            }
            // Nothing pending in the TXB?  Flag the TSR as empty, and
            // set the "Transmit complete" flag in the register.
            else
            {
                TXB = 0;
                TSR = 0;
                bTSR_Empty = true;

                UART_TxComplete();
                if (UART_IsTxIntEnabled())
                {
                    DEBUG_PRINT("TXC Interrupt\n");
                    AVR_InterruptCandidate( 0x14 );
                }
            }
        }
    }
```

```
}

//---------------------------------------------------------------------------
static void UART_RxClock(void *context_ )
{
    if (UART_IsRxEnabled() && u32RxTicksRemaining)
    {
        u32RxTicksRemaining--;
        if (!u32RxTicksRemaining)
        {
            // Local echo of the freshly "shifted in" data to the terminal
            Echo_Rx();

            // Move data from receive shift register into the receive buffer
            RXB = RSR;
            RSR = 0;

            // Set the RX Complete flag
            UART_RxComplete();
            if (UART_IsRxIntEnabled())
            {
                DEBUG_PRINT("RXC Interrupt\n");
                AVR_InterruptCandidate( 0x12 );
            }
        }
    }
}
//---------------------------------------------------------------------------
static void UART_Clock(void *context_ )
{
    // Handle Rx and TX clocks.
    UART_TxClock(context_);
    UART_RxClock(context_);
}

//---------------------------------------------------------------------------
AVRPeripheral stUART =
{
    UART_Init,
    UART_Read,
    UART_Write,
    UART_Clock,
    0,
    0xC0,
    0xC6
};
```

4.133 mega_uart.h File Reference

ATMega UART implementation.

`#include "avr_peripheral.h"`

Variables

- AVRPeripheral **stUART**

4.133.1 Detailed Description

ATMega UART implementation.

Definition in file mega_uart.h.

4.134 mega_uart.h

```
/****************************************************************************
 *     (     (                      (     |
 *    )\ )  )\ )    (              )\ )   |
 *   (()/( (()/(    )\     (   (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
```

```
 *   (_))_|(_))    )\ _ )\ ((_)((_)(_))    | -- [ AVR ] --------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                        | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __MEGA_UART_H__
#define __MEGA_UART_H__

#include "avr_peripheral.h"

extern AVRPeripheral stUART;

#endif //__MEGA_UART_H__
```

4.135 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_ELFFILE**, **OPTION_DEBUG**, **OPTION_GDB**, **OPTION_SILENT**, **OPTION_DISASM**, **OPTION_TRACE**, **OPTION_MARK3**, **OPTION_EXITRESET**, **OPTION_PROFILE**, 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.
- void Options_PrintUsage (void)

  Options_PrintUsage.

Variables

- static Option_t astAttributes [OPTION_NUM]

  Table of available commandline options.

4.135.1 Detailed Description

Module for managing command-line options.

Definition in file options.c.

4.135.2 Enumeration Type Documentation

4.135.2.1 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 44 of file options.c.

4.135.3 Function Documentation

4.135.3.1 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 97 of file options.c.

4.135.3.2 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 197 of file options.c.

4.135.3.3 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 186 of file options.c.

4.135.3.4 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 ast← Attributes table.

Parameters

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

Returns

The next index to process

Definition at line 124 of file options.c.

4.135.3.5 void Options_PrintUsage (void)

Options_PrintUsage.

Print a brief description of each command-line option and its usage.

Definition at line 204 of file options.c.

4.135.3.6 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 91 of file options.c.

4.135.4 Variable Documentation

4.135.4.1 Option_t astAttributes[OPTION_NUM] [static]

Initial value:

```
=
{
    {"--variant",   "Specify the CPU variant by model name (default - atmega328p)", NULL, false },
    {"--freq",      "Speed (in Hz) of the simulated CPU", NULL, false },
    {"--hexfile",   "Programming file (intel HEX binary). Mutually exclusive with --elffile ", NULL, false
      },
    {"--elffile",   "Programming file (ELF binary).  Mutually exclusive with --hexfile", NULL, false },
    {"--debug",     "Run simulator in interactive debug mode.  Mutually exclusive with --gdb", NULL, true }
      ,
    {"--gdb",       "Run simulator as a GDB remote, on the specified port.", NULL, false },
    {"--silent",    "Start without the flavr-banner print", NULL, true },
    {"--disasm",    "Disassemble programming file to standard output", NULL, true },
    {"--trace",     "Enable tracebuffer support when used in conjunction with --debug", NULL, true },
    {"--mark3",     "Enable Mark3 kernel-aware plugin", NULL, true },
    {"--exitreset", "Exit simulator if a jump-to-zero operation is encountered", NULL, true },
    {"--profile",   "Run with code profile and code coverage enabled", NULL, true },
}
```

Table of available commandline options.

Order must match enumeration defined above.

Definition at line 67 of file options.c.

4.136 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 2014-15, 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     const char *szDescription;
00035     char *szParameter;
00036     bool bStandalone;
00037 } Option_t;
00038
00039 //---------------------------------------------------------------------------
00044 typedef enum
00045 {
00046     OPTION_VARIANT,
00047     OPTION_FREQ,
00048     OPTION_HEXFILE,
00049     OPTION_ELFFILE,
00050     OPTION_DEBUG,
00051     OPTION_GDB,
00052     OPTION_SILENT,
00053     OPTION_DISASM,
00054     OPTION_TRACE,
00055     OPTION_MARK3,
00056     OPTION_EXITRESET,
00057     OPTION_PROFILE,
00058 //-- New options go here ^^^
00059     OPTION_NUM
00060 } OptionIndex_t;
00061
00062 //---------------------------------------------------------------------------
00067 static Option_t astAttributes[OPTION_NUM] =
00068 {
00069     {"--variant",   "Specify the CPU variant by model name (default - atmega328p)", NULL, false },
00070     {"--freq",      "Speed (in Hz) of the simulated CPU", NULL, false },
00071     {"--hexfile",   "Programming file (intel HEX binary). Mutually exclusive with --elffile ", NULL, false
      },
00072     {"--elffile",   "Programming file (ELF binary).  Mutually exclusive with --hexfile", NULL, false },
00073     {"--debug",     "Run simulator in interactive debug mode.  Mutually exclusive with --gdb", NULL, true }
      ,
00074     {"--gdb",       "Run simulator as a GDB remote, on the specified port.", NULL, false },
00075     {"--silent",    "Start without the flavr-banner print", NULL, true },
00076     {"--disasm",    "Disassemble programming file to standard output", NULL, true },
00077     {"--trace",     "Enable tracebuffer support when used in conjunction with --debug", NULL, true },
00078     {"--mark3",     "Enable Mark3 kernel-aware plugin", NULL, true },
00079     {"--exitreset", "Exit simulator if a jump-to-zero operation is encountered", NULL, true },
00080     {"--profile",   "Run with code profile and code coverage enabled", NULL, true },
00081 };
00082
00083 //---------------------------------------------------------------------------
00091 static void Options_SetDefaults( void )
00092 {
00093     astAttributes[ OPTION_VARIANT ].szParameter  = strdup( "atmega328p" );
00094     astAttributes[ OPTION_FREQ ].szParameter     = strdup( "16000000" );
00095 }
00096 //---------------------------------------------------------------------------
00097 const char *Options_GetByName (const char *szAttribute_)
```

```
{
    uint16_t j;

    // linear search for the correct option value.
    for (j = 0; j < OPTION_NUM; j++)
    {
        if (0 == strcmp(astAttributes[j].szAttribute, szAttribute_))
        {
            return (const char*)astAttributes[j].szParameter;
        }
    }
    return NULL;
}

//---------------------------------------------------------------------------
static uint16_t Options_ParseElement( int start_, int argc_, char **argv_ )
{
    // Parse out specific option parameter data for a given option attribute
    uint16_t i = start_;
    uint16_t j;

    while (i < argc_)
    {
        // linear search for the correct option value.
        for (j = 0; j < OPTION_NUM; j++)
        {
            if (0 == strcmp(astAttributes[j].szAttribute, argv_[i]))
            {
                // Match - is the option stand-alone, or does it take a parameter?
                if (astAttributes[j].bStandalone)
                {
                    // Standalone argument, auto-seed a "1" value for the parameter to
                    // indicate that the option was set on the commandline
                    astAttributes[j].szParameter = strdup("1");
                    return 1;
                }

                // ensure the user provided a parameter for this attribute
                if (i + 1 >= argc_)
                {
                    fprintf( stderr, "Error: Paramter expected for attribute %s", argv_[i] );
                    exit(-1);
                }
                else if (*(char*)argv_[i+1] == '-')
                {
                    fprintf( stderr, "Error: Paramter expected for attribute %s", argv_[i] );
                    exit(-1);
                }
                // Check to see if a parameter has already been set; if so, free the existing value
                if (NULL != astAttributes[j].szParameter)
                {
                    free(astAttributes[j].szParameter );
                }
                // fprintf( stderr, "Match: argv[i]=%s, argv[i+1]=%s\n", argv_[i], argv_[i+1] );
                astAttributes[j].szParameter = strdup(argv_[i+1]);
            }
        }
        // Read attribute + parameter combo, 2 tokens
        return 2;
    }

    // Unknown option - 1 token
    fprintf( stderr, "WARN: Invalid option \"%s\"", argv_[i] );

    return 1;
}

//---------------------------------------------------------------------------
static void Options_Parse(int argc_, char **argv_ )
{
    uint16_t i = 1;
    while (i < argc_)
    {
        // Parse out token from the command line array.
        i += Options_ParseElement( i, argc_, argv_ );
    }
}

//---------------------------------------------------------------------------
void Options_Init( int argc_, char **argv_ )
{
    Options_SetDefaults();
    Options_Parse( argc_, argv_ );
}

//---------------------------------------------------------------------------
void Options_PrintUsage(void)
```

```
{
    int i;
    printf("\n Usage:\n\n"
           "    flavr <options>\n\n Where <options> include:\n");
    for (i = 0; i < OPTION_NUM; i++)
    {
        printf( "  %14s: %s", astAttributes[i].szAttribute, astAttributes[i].szDescription );
        if (!astAttributes[i].bStandalone)
        {
            printf(" (takes an argument)" );
        }
        printf( "\n" );
    }
}
```

4.137 tlv_file.c File Reference

Tag-length-value file format used for encoding simulator run-time data (kernel-aware plugin data, code profiling statistics, etc.).

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include "tlv_file.h"
```

Functions

- void TLV_WriteInit (const char ∗szPath_)

    TLV_WriteInit.
- void **TLV_WriteFinish** (void)
- TLV_t ∗ TLV_Alloc (uint16_t u16Len_)

    TLV_Alloc.
- void TLV_Free (TLV_t ∗pstTLV_)

    TLV_Free.
- int TLV_Write (TLV_t ∗pstData_)

    TLV_Write.
- int TLV_ReadInit (const char ∗szPath_, uint8_t ∗∗pu8Buffer_)

    TLV_ReadInit.
- int TLV_Read (TLV_t ∗pstTLV_, uint8_t ∗pu8Buffer_, int iIndex_)

    TLV_Read.
- void TLV_ReadFinish (uint8_t ∗pu8Buffer_)

    TLV_ReadFinish.

Variables

- static FILE ∗ **fMyFile** = NULL

4.137.1 Detailed Description

Tag-length-value file format used for encoding simulator run-time data (kernel-aware plugin data, code profiling statistics, etc.).

Definition in file tlv_file.c.

4.137.2 Function Documentation

4.137.2.1 TLV_t* TLV_Alloc (uint16_t *u16Len_*)

TLV_Alloc.

Dynamically allocate an appropriately-sized TLV buffer struct with a large enough data array to store u16Len_ bytes of data.

Parameters

*u16Len_*	Length of the data array to allocate

Returns

Pointer to a newly-allocated object, or NULL on error

Definition at line 55 of file tlv_file.c.

4.137.2.2 void TLV_Free (TLV_t * *pstTLV_*)

TLV_Free.

Free a previously-allocated TLV object.

Parameters

*pstTLV_*	Pointer to a valid, previously-allocated TLV object

Definition at line 61 of file tlv_file.c.

4.137.2.3 int TLV_Read (TLV_t * *pstTLV_*, uint8_t * *pu8Buffer_*, int *iIndex_*)

TLV_Read.

Read an entry from a local copy of the TLV buffer into a user-provided TLV pointer.

Parameters

*pstTLV_*	Pointer to a valid TLV object, with a buffer large enough to hold the largest data object we may encounter.
*pu8Buffer_*	Pointer to a buffer containing the contents of the TLV input file.
*iIndex_*	Byte index at whch to start reading TLV data.

Returns

Number of bytes read into the TLV struct

! ToDo – add checks around buffer usage

Definition at line 102 of file tlv_file.c.

4.137.2.4 void TLV_ReadFinish (uint8_t * *pu8Buffer_*)

TLV_ReadFinish.

Dispose of the in-ram copy of the TLV read buffer, allocated from TLV_ReadInit

Parameters

*pu8Buffer_*	Pointer to the previously allocated TLV ram buffer

Definition at line 113 of file tlv_file.c.

4.137.2.5 int TLV_ReadInit (const char ∗ *szPath_*, uint8_t ∗∗ *pu8Buffer_*)

TLV_ReadInit.

Open the tlv-formatted binary specified in the szPath_ argument, and read its contents into a newly-allocated buffer, which is passed back to the user by the double-pointer pu8Buffer_ argument..

Parameters

*szPath_*	Path to the file to open
*pu8Buffer_*	Pointer which will be assigned to the newly-created buffer.

Returns

size of the newly-created buffer (in bytes), or 0 on error.

Definition at line 76 of file tlv_file.c.

4.137.2.6 int TLV_Write (TLV_t ∗ *pstData_*)

TLV_Write.

Write a TLV record to the active file stream.

Parameters

*pstData_*	Pointer to a valid TLV object to log

Returns

-1 on error, number of bytes written on success.

Definition at line 67 of file tlv_file.c.

4.137.2.7 void TLV_WriteInit (const char ∗ *szPath_*)

TLV_WriteInit.

Initialize the TLV file used to store profiling and diagnostics information in an efficient binary format. Must be called before logging TLV data.

Parameters

*szPath_*	Name of the TLV output file to create

Definition at line 36 of file tlv_file.c.

4.138 tlv_file.c

```
/*****************************************************************************
 *     (     (                      (    |
 *    )\ )  )\ )   (              )\ )  |
 *   (()/( (()/(   )\     (   (  (()/(  | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))   | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))  | -- [ AVR ] ---------------------
```

```
 *   | |_  | |     (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |_____| /_/ \_\   \_/   |_|_\   |
 *                                         | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>

#include "tlv_file.h"

//---------------------------------------------------------------------------
static FILE *fMyFile = NULL;

//---------------------------------------------------------------------------
void TLV_WriteInit( const char *szPath_ )
{
    if (!fMyFile)
    {
        fMyFile = fopen( szPath_, "wb" );
    }
}

//---------------------------------------------------------------------------
void TLV_WriteFinish( void )
{
    if (fMyFile)
    {
        fclose(fMyFile);
    }
    fMyFile = NULL;
}

//---------------------------------------------------------------------------
TLV_t *TLV_Alloc( uint16_t u16Len_ )
{
    return (TLV_t*)(malloc(sizeof(TLV_t) + u16Len_ - 1));
}

//---------------------------------------------------------------------------
void TLV_Free( TLV_t *pstTLV_ )
{
    free( pstTLV_ );
}

//---------------------------------------------------------------------------
int TLV_Write( TLV_t *pstData_ )
{
    if (fMyFile)
    {
        return fwrite( (void*)pstData_, sizeof(uint8_t), sizeof(TLV_t) + pstData_->u16Len - 1, fMyFile );
    }
    return -1;
}
//---------------------------------------------------------------------------
int TLV_ReadInit( const char *szPath_, uint8_t **pu8Buffer_ )
{
    FILE *fReadFile = fopen( szPath_, "rb" );
    struct stat stStat;

    if (!fReadFile)
    {
        fprintf(stderr, "Unable to open tlv for input!\n" );
        return 0;
    }

    stat( szPath_, &stStat );
    *pu8Buffer_ = (uint8_t*)malloc( stStat.st_size );
    if (!pu8Buffer_)
    {
        fclose(fReadFile);
        fprintf(stderr, "Unable to allocate local tlv read buffer!\n" );
        return 0;
    }
    fread(*pu8Buffer_, 1, stStat.st_size, fReadFile );

    fclose(fReadFile);
    return stStat.st_size;
}
```

```
//---------------------------------------------------------------------------
int TLV_Read( TLV_t *pstTLV_, uint8_t *pu8Buffer_, int iIndex_)
{
    TLV_t *pstStreamTLV = (TLV_t*)&(pu8Buffer_[iIndex_]);
    pstTLV_->eTag = pstStreamTLV->eTag;
    pstTLV_->u16Len = pstStreamTLV->u16Len;
    memcpy( pstTLV_->au8Data, pstStreamTLV->au8Data, pstTLV_->u16Len );
    return (sizeof(TLV_t) + pstTLV_->u16Len - 1);
}

//---------------------------------------------------------------------------
void TLV_ReadFinish ( uint8_t *pu8Buffer_ )
{
    if (pu8Buffer_)
    {
        free( pu8Buffer_ );
    }
}
```

4.139 tlv_file.h File Reference

Tag-length-value file format used for encoding simulator run-time data (kernel-aware plugin data, code profiling statistics, etc.).

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

Data Structures

- struct TLV_t

Enumerations

- enum FlavrTag_t { TAG_KERNEL_AWARE_INTERRUPT, TAG_KERNEL_AWARE_CONTEXT_SWITCH, TAG_KERNEL_AWARE_PRINT, TAG_KERNEL_AWARE_TRACE_0, TAG_KERNEL_AWARE_TRACE_1, TAG_KERNEL_AWARE_TRACE_2, TAG_KERNEL_AWARE_PROFILE, TAG_KERNEL_AWARE_THREAD_PROFILE_EPOCH, TAG_KERNEL_AWARE_THREAD_PROFILE_GLOBAL, TAG_CODE_PROFILE_FUNCTION_EPOCH, TAG_CODE_PROFILE_FUNCTION_GLOBAL, TAG_CODE_COVERAGE_FUNCTION_EPOCH, TAG_CODE_COVERAGE_FUNCTION_GLOBAL, TAG_CODE_COVERAGE_GLOBAL, TAG_CODE_COVERAGE_ADDRESS, **TAG_COUNT** }

Functions

- void TLV_WriteInit (const char *szPath_)

  TLV_WriteInit.
- void **TLV_WriteFinish** (void)
- TLV_t * TLV_Alloc (uint16_t u16Len_)

  TLV_Alloc.
- void TLV_Free (TLV_t *pstTLV_)

  TLV_Free.
- int TLV_Write (TLV_t *pstData_)

  TLV_Write.
- int TLV_ReadInit (const char *szPath_, uint8_t **pu8Buffer_)

TLV_ReadInit.

- int TLV_Read (TLV_t ∗pstTLV_, uint8_t ∗pu8Buffer_, int iIndex_)

    TLV_Read.

- void TLV_ReadFinish (uint8_t ∗pu8Buffer_)

    TLV_ReadFinish.

4.139.1 Detailed Description

Tag-length-value file format used for encoding simulator run-time data (kernel-aware plugin data, code profiling statistics, etc.).

Definition in file tlv_file.h.

4.139.2 Enumeration Type Documentation

4.139.2.1 enum FlavrTag_t

Enumerator

TAG_KERNEL_AWARE_INTERRUPT Kernel-aware plugin generated interrupt events.

TAG_KERNEL_AWARE_CONTEXT_SWITCH Kernel-aware plugin generated context switch events.

TAG_KERNEL_AWARE_PRINT Prints generated from kernel-aware debugger.

TAG_KERNEL_AWARE_TRACE_0 Kernel trace events.

TAG_KERNEL_AWARE_TRACE_1 Kernel trace events, 1 argument.

TAG_KERNEL_AWARE_TRACE_2 Kernel trace events, 2 arguments.

TAG_KERNEL_AWARE_PROFILE Kernel-aware profiling events.

TAG_KERNEL_AWARE_THREAD_PROFILE_EPOCH Epoch-based thread profiling (i.e. CPU use per thread, per epoch)

TAG_KERNEL_AWARE_THREAD_PROFILE_GLOBAL Global thread profiling (i.e. CPU use per thread, cumulative)

TAG_CODE_PROFILE_FUNCTION_EPOCH CPU Profiling for a given function (per epoch)

TAG_CODE_PROFILE_FUNCTION_GLOBAL CPU Profiling for a given function (cumulative)

TAG_CODE_COVERAGE_FUNCTION_EPOCH Code coverage for a given function (per epoch)

TAG_CODE_COVERAGE_FUNCTION_GLOBAL Code coverage for a given function (cumulative)

TAG_CODE_COVERAGE_GLOBAL Global code coverage (cumulative)

TAG_CODE_COVERAGE_ADDRESS Code coverage stats for a given address (cumulative)

Definition at line 31 of file tlv_file.h.

4.139.3 Function Documentation

4.139.3.1 TLV_t∗ TLV_Alloc (uint16_t *u16Len_*)

TLV_Alloc.

Dynamically allocate an appropriately-sized TLV buffer struct with a large enough data array to store u16Len_ bytes of data.

Parameters

*u16Len_*	Length of the data array to allocate

Returns

Pointer to a newly-allocated object, or NULL on error

Definition at line 55 of file tlv_file.c.

4.139.3.2 void TLV_Free (TLV_t * *pstTLV_*)

TLV_Free.

Free a previously-allocated TLV object.

Parameters

*pstTLV_*	Pointer to a valid, previously-allocated TLV object

Definition at line 61 of file tlv_file.c.

4.139.3.3 int TLV_Read (TLV_t * *pstTLV_*, uint8_t * *pu8Buffer_*, int *iIndex_*)

TLV_Read.

Read an entry from a local copy of the TLV buffer into a user-provided TLV pointer.

Parameters

*pstTLV_*	Pointer to a valid TLV object, with a buffer large enough to hold the largest data object we may encounter.
*pu8Buffer_*	Pointer to a buffer containing the contents of the TLV input file.
*iIndex_*	Byte index at whch to start reading TLV data.

Returns

Number of bytes read into the TLV struct

! ToDo – add checks around buffer usage

Definition at line 102 of file tlv_file.c.

4.139.3.4 void TLV_ReadFinish (uint8_t * *pu8Buffer_*)

TLV_ReadFinish.

Dispose of the in-ram copy of the TLV read buffer, allocated from TLV_ReadInit

Parameters

*pu8Buffer_*	Pointer to the previously allocated TLV ram buffer

Definition at line 113 of file tlv_file.c.

4.139.3.5 int TLV_ReadInit (const char * *szPath_*, uint8_t ** *pu8Buffer_*)

TLV_ReadInit.

Open the tlv-formatted binary specified in the szPath_ argument, and read its contents into a newly-allocated buffer, which is passed back to the user by the double-pointer pu8Buffer_ argument..

Parameters

*szPath_*	Path to the file to open
*pu8Buffer_*	Pointer which will be assigned to the newly-created buffer.

Returns

size of the newly-created buffer (in bytes), or 0 on error.

Definition at line 76 of file tlv_file.c.

4.139.3.6 int TLV_Write (TLV_t ∗ *pstData_*)

TLV_Write.

Write a TLV record to the active file stream.

Parameters

*pstData_*	Pointer to a valid TLV object to log

Returns

-1 on error, number of bytes written on success.

Definition at line 67 of file tlv_file.c.

4.139.3.7 void TLV_WriteInit (const char ∗ *szPath_*)

TLV_WriteInit.

Initialize the TLV file used to store profiling and diagnostics information in an efficient binary format. Must be called before logging TLV data.

Parameters

*szPath_*	Name of the TLV output file to create

Definition at line 36 of file tlv_file.c.

4.140 tlv_file.h

```
/*****************************************************************************
 *     (      (                     (      |
 *    )\ )   )\ )      (            )\ )   |
 *   (()/(  (()/(     )\      (   (  (()/(   | -- [ Funkenstein ] -------------
 *    /(_))  /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\    |
 *                                         | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __TLV_FILE_H__
#define __TLV_FILE_H__

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//---------------------------------------------------------------------------
typedef enum
{
```

```
    TAG_KERNEL_AWARE_INTERRUPT,
    TAG_KERNEL_AWARE_CONTEXT_SWITCH,
    TAG_KERNEL_AWARE_PRINT,
    TAG_KERNEL_AWARE_TRACE_0,
    TAG_KERNEL_AWARE_TRACE_1,
    TAG_KERNEL_AWARE_TRACE_2,
    TAG_KERNEL_AWARE_PROFILE,
    TAG_KERNEL_AWARE_THREAD_PROFILE_EPOCH,
    TAG_KERNEL_AWARE_THREAD_PROFILE_GLOBAL,
    TAG_CODE_PROFILE_FUNCTION_EPOCH,
    TAG_CODE_PROFILE_FUNCTION_GLOBAL,
    TAG_CODE_COVERAGE_FUNCTION_EPOCH,
    TAG_CODE_COVERAGE_FUNCTION_GLOBAL,
    TAG_CODE_COVERAGE_GLOBAL,
    TAG_CODE_COVERAGE_ADDRESS,
//---
    TAG_COUNT
} FlavrTag_t;

//---------------------------------------------------------------------------
typedef struct
{
    FlavrTag_t eTag;
    uint16_t   u16Len;
    uint8_t    au8Data[1];
} TLV_t;

//---------------------------------------------------------------------------
void TLV_WriteInit( const char *szPath_ );

void TLV_WriteFinish( void );
//---------------------------------------------------------------------------
TLV_t *TLV_Alloc( uint16_t u16Len_ );

//---------------------------------------------------------------------------
void TLV_Free( TLV_t *pstTLV_ );

//---------------------------------------------------------------------------
int TLV_Write( TLV_t *pstData_ );

//---------------------------------------------------------------------------
int TLV_ReadInit( const char *szPath_, uint8_t **pu8Buffer_ );

//---------------------------------------------------------------------------
int TLV_Read( TLV_t *pstTLV_, uint8_t *pu8Buffer_, int iIndex_);

//---------------------------------------------------------------------------
void TLV_ReadFinish( uint8_t *pu8Buffer_ );

#endif
```

4.141 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 u32Element_)

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

4.141.1 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.

4.141.2 Function Documentation

4.141.2.1 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.

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

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

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.

4.141.2.3 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 120 of file trace_buffer.c.

4.141.2.4 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.

4.141.2.5 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.

4.142 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 2014-15, 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++;
```

```
    if (pstTraceBuffer_->u32Index >= CONFIG_TRACEBUFFER_SIZE)
    {
        pstTraceBuffer_->u32Index = 0;
    }
}

//---------------------------------------------------------------------------
void TraceBuffer_LoadElement( TraceBuffer_t *pstTraceBuffer_, TraceElement_t *pstElement_, uint32_t u32Element_ )
{
    TraceElement_t *pstSourceElement = &pstTraceBuffer_->astTraceStep[ pstTraceBuffer_->u32Index ];

    memcpy(pstElement_, pstSourceElement, sizeof(*pstElement_));
}

//---------------------------------------------------------------------------
void TraceBuffer_PrintElement( TraceElement_t *pstElement_, TracePrintFormat_t eFormat_  )
{
    printf( "[%08d] 0x%04X:0x%04X: ",
            pstElement_->u64Counter, pstElement_->u16PC, pstElement_->u16OpCode );
    if (eFormat_ & TRACE_PRINT_DISASSEMBLY)
    {
        uint16_t u16TempPC = stCPU.u16PC;
        stCPU.u16PC = pstElement_->u16PC;

        AVR_Disasm pfOp = AVR_Disasm_Function( pstElement_->u16OpCode );

        char szBuf[256];
        AVR_Decode( pstElement_->u16OpCode );
        pfOp( szBuf );
        printf( "%s", szBuf );

        stCPU.u16PC = u16TempPC;
    }

    if (eFormat_ & TRACE_PRINT_COMPACT)
    {
        printf( "%04X ", pstElement_->u16SP );

        int i;
        for (i = 0; i < 32; i++)
        {
            printf( "%02X ", pstElement_->stCoreRegs.r[i] );
        }
        printf( "\n" );
    }
    if (eFormat_ & TRACE_PRINT_REGISTERS)
    {
        uint8_t i;
        for (i = 0; i < 32; i++)
        {
            printf( "[R%02d] = 0x%02X\n", i, pstElement_->stCoreRegs.r[i] );
        }
        printf("[SP]   = 0x%04X\n", pstElement_->u16SP );
        printf("[PC]   = 0x%04X\n", (uint16_t)pstElement_->u16PC );
        printf("[SREG]= 0x%02X", pstElement_->u8SR );
        printf( "\n" );
    }
}

//---------------------------------------------------------------------------
void TraceBuffer_Print( TraceBuffer_t *pstTraceBuffer_, TracePrintFormat_t eFormat_ )
{
    int i;
    for (i = pstTraceBuffer_->u32Index; i < CONFIG_TRACEBUFFER_SIZE; i++)
    {
        TraceBuffer_PrintElement(&pstTraceBuffer_->astTraceStep[i], eFormat_ );
    }
    for (i = 0; i < pstTraceBuffer_->u32Index; i++)
    {
        TraceBuffer_PrintElement(&pstTraceBuffer_->astTraceStep[i], eFormat_ );
    }
}
```

4.143 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 defining the CPU's running state at each tracebuffer sample point.
- struct TraceBuffer_t

  Implements a circular buffer of trace elements, sized according to the compile-time configuration.

Enumerations

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

  Enumerated values defining the various formats for printing/displaying tracebuffer information.

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 output element.
- 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.

4.143.1 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.

4.143.2 Function Documentation

4.143.2.1 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.

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

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

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.

4.143.2.3 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 120 of file trace_buffer.c.

4.143.2.4 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.

4.143.2.5 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.

4.144 trace_buffer.h

```
/***************************************************************************
 *     (     (                      (     |
 *    )\ )  )\ )    (             )\ )    |
 *   (()/( (()/(    )\     (    (   (()/(   | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                        | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __TRACE_BUFFER_H__
#define __TRACE_BUFFER_H__

#include <stdint.h>

#include "emu_config.h"
#include "avr_cpu.h"

//---------------------------------------------------------------------------
typedef struct
{
    uint64_t    u64Counter;
    uint64_t    u64CycleCount;
    uint16_t    u16OpCode;
    uint16_t    u16PC;
    uint16_t    u16SP;
    uint8_t     u8SR;

    AVR_CoreRegisters stCoreRegs;

} TraceElement_t;

//---------------------------------------------------------------------------
typedef struct
{
    TraceElement_t  astTraceStep[ CONFIG_TRACEBUFFER_SIZE ];
    uint32_t        u32Index;
} TraceBuffer_t;

//---------------------------------------------------------------------------
typedef enum
{
    TRACE_PRINT_COMPACT     = 1,
    TRACE_PRINT_REGISTERS   = 2,
    TRACE_PRINT_DISASSEMBLY = 4
} TracePrintFormat_t;

//---------------------------------------------------------------------------
void TraceBuffer_Init( TraceBuffer_t *pstTraceBuffer_ );

//---------------------------------------------------------------------------
void TraceBuffer_StoreFromCPU( TraceBuffer_t *pstTraceBuffer_ );

//---------------------------------------------------------------------------
void TraceBuffer_LoadElement( TraceBuffer_t *pstTraceBuffer_,
      TraceElement_t *pstElement_, uint32_t u32Element_ );

//---------------------------------------------------------------------------
void TraceBuffer_PrintElement( TraceElement_t *pstElement_,
      TracePrintFormat_t eFormat_ );

//---------------------------------------------------------------------------
void TraceBuffer_Print( TraceBuffer_t *pstTraceBuffer_,
      TracePrintFormat_t eFormat_ );

#endif
```

4.145 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** []

4.145.1 Detailed Description

Module containing a table of device variants supported by flavr.

Definition in file variant.c.

4.145.2 Function Documentation

4.145.2.1 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.

4.145.3 Variable Documentation

4.145.3.1 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.

4.146 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 2014-15, 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 }
```

```
};

//---------------------------------------------------------------------------
const AVR_Variant_t *Variant_GetByName( const char *szName_ )
{
    AVR_Variant_t *pstVariant = astVariants;
    while (pstVariant->szName)
    {
        if (0 == strcmp(pstVariant->szName, szName_ ) )
        {
            return pstVariant;
        }
        pstVariant++;
    }
    return NULL;
}

```

4.147 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.

Functions

- const AVR_Variant_t * Variant_GetByName (const char *szName_)

  Variant_GetByName.

4.147.1 Detailed Description

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

Definition in file variant.h.

4.147.2 Function Documentation

4.147.2.1 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.

4.148 variant.h

```
/*****************************************************************************
 *     (     (                      (     |
 *    )\ )  )\ )    (             )\ )    |
 *   (()/( (()/(    )\     (   ( (()/(    | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))     | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))    | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \   | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /   | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\   |
 *                                        | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __VARIANT_H__
#define __VARIANT_H__

//---------------------------------------------------------------------------
typedef struct
{
    const char  *szName;

    uint32_t    u32RAMSize;
    uint32_t    u32ROMSize;
    uint32_t    u32EESize;

    const uint8_t *u8Descriptors;

} AVR_Variant_t;

//---------------------------------------------------------------------------
const AVR_Variant_t *Variant_GetByName( const char *szName_ );

#endif
```

4.149 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.

4.149.1 Detailed Description

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

Definition in file watchpoint.c.

4.149.2 Function Documentation

4.149.2.1 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.

4.149.2.2 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.

4.149.2.3 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.

4.150 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 2014-15, 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 {
```

```
    // Don't add multiple watchpoints at the same address
    if (WatchPoint_EnabledAtAddress( u16Addr_ ))
    {
        return;
    }

    WatchPoint_t *pstNewWatch = NULL;

    pstNewWatch = (WatchPoint_t*)malloc( sizeof(WatchPoint_t) );

    pstNewWatch->next = stCPU.pstWatchPoints;
    pstNewWatch->prev = NULL;

    pstNewWatch->u16Addr = u16Addr_;

    if (stCPU.pstWatchPoints)
    {
        WatchPoint_t *pstTemp = stCPU.pstWatchPoints;
        pstTemp->prev = pstNewWatch;
    }
    stCPU.pstWatchPoints = pstNewWatch;
}

//---------------------------------------------------------------------------
void WatchPoint_Delete( uint16_t u16Addr_ )
{
    WatchPoint_t *pstTemp = stCPU.pstWatchPoints;

    while (pstTemp)
    {
        if (pstTemp->u16Addr == u16Addr_)
        {
            // Remove node -- reconnect surrounding elements
            WatchPoint_t *pstNext = pstTemp->next;
            if (pstNext)
            {
                pstNext->prev = pstTemp->prev;
            }

            WatchPoint_t *pstPrev = pstTemp->prev;
            if (pstPrev)
            {
                pstPrev->next = pstTemp->next;
            }

            // Adjust list-head if necessary
            if (pstTemp == stCPU.pstWatchPoints)
            {
                stCPU.pstWatchPoints = pstNext;
            }

            // Free the node/iterate to next node.
            pstPrev = pstTemp;
            pstTemp = pstTemp->next;
            free(pstPrev);
        }
        else
        {
            pstTemp = pstTemp->next;
        }
    }
}

//---------------------------------------------------------------------------
bool WatchPoint_EnabledAtAddress( uint16_t u16Addr_ )
{
    WatchPoint_t *pstTemp = stCPU.pstWatchPoints;

    while (pstTemp)
    {
        if (pstTemp->u16Addr == u16Addr_)
        {
            return true;
        }
        pstTemp = pstTemp->next;
    }
    return false;
}
```

4.151 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.

4.151.1 Detailed Description

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

Definition in file watchpoint.h.

4.151.2 Function Documentation

4.151.2.1 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.

4.151.2.2 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.

4.151.2.3 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.

4.152 watchpoint.h

```
/*****************************************************************************
 *     (     (                      (     |
 *    )\ )  )\ )    (              )\ )  |
 *   (()/( (()/(    )\     (   (  (()/(  | -- [ Funkenstein ] -------------
 *    /(_)) /(_))((((_)()\  )\  /(_))   | -- [ Litle ] -------------------
 *   (_))_|(_))   )\ _ )\ ((_)((_)(_))   | -- [ AVR ] ---------------------
 *   | |_  | |    (_)_\(_)\ \ / / | _ \  | -- [ Virtual ] -----------------
 *   | __| | |__   / _ \   \ V /  |   /  | -- [ Runtime ] -----------------
 *   |_|   |____| /_/ \_\   \_/   |_|_\  |
 *                                       | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __WATCHPOINT_H__
#define __WATCHPOINT_H__

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

#include "avr_cpu.h"

//---------------------------------------------------------------------------
typedef struct _WatchPoint
{
    struct _WatchPoint *next;
    struct _WatchPoint *prev;

    uint16_t    u16Addr;
} WatchPoint_t;

//---------------------------------------------------------------------------
void WatchPoint_Insert( uint16_t u16Addr_ );

//---------------------------------------------------------------------------
void WatchPoint_Delete( uint16_t u16Addr_ );

//---------------------------------------------------------------------------
bool WatchPoint_EnabledAtAddress( uint16_t u16Addr_ );

#endif
```

4.153 write_callout.h File Reference

Extended emulator functionality allowing for functions to be triggered based on RAM-write operations.

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

Typedefs

- typedef bool(∗ WriteCalloutFunc)(uint16_t u16Addr_, uint8_t u8Data_)

  Function pointer type for memory-write callout handlers.

Functions

- void WriteCallout_Add (WriteCalloutFunc pfCallout_, uint16_t u16Addr_)

  WriteCallout_Add.
- bool WriteCallout_Run (uint16_t u16Addr_, uint8_t u8Data_)

  WriteCallout_Run.

4.153.1 Detailed Description

Extended emulator functionality allowing for functions to be triggered based on RAM-write operations.

Definition in file write_callout.h.

4.153.2 Function Documentation

4.153.2.1 void WriteCallout_Add (WriteCalloutFunc *pfCallout_*, uint16_t *u16Addr_*)

WriteCallout_Add.

Registers a specific function to be called whenever a specific address in memory is modified. Multiple functions can be registered at the same location in memory.

Parameters

*pfCallout_*	- Pointer to the callout function
*u16Addr_*	- Address in RAM that triggers the callout when written

Definition at line 60 of file write_callout.c.

4.153.2.2 bool WriteCallout_Run (uint16_t *u16Addr_*, uint8_t *u8Data_*)

WriteCallout_Run.

Function called by the AVR CPU core whenever a word in memory is written. This searches the list of write callouts and executes any callouts registered at the specific address.

Parameters

*u16Addr_*	- Address in RAM currently being modified
*u8Data_*	- Data that will be written to the address

Returns

false - bypass CPU's own write function for this memory write.

Definition at line 77 of file write_callout.c.

4.154 write_callout.h

```
/*****************************************************************************
 *     (     (                       (     |
 *    )\ )  )\ )    (              )\ )   |
```

```
 *    (()/( (()/(     )\       (    (   (()/(    | -- [ Funkenstein ] -------------
 *     /(_)) /(_))((((_)()\  )\  /(_))      | -- [ Litle ] -------------------
 *    (_))_|(_))    )\ _ )\ ((_)((_)(_))     | -- [ AVR ] ---------------------
 *    | |_  | |     (_)_\(_)\ \ / / | _ \    | -- [ Virtual ] -----------------
 *    | __| | |__    / _ \   \ V /  |   /    | -- [ Runtime ] -----------------
 *    |_|   |____| /_/ \_\   \_/   |_|_\    |
 *                                          | "Yeah, it does Arduino..."
 * ---------------------------------------+----------------------------------
 * (c) Copyright 2014-15, Funkenstein Software Consulting, All rights reserved
 *     See license.txt for details
 ****************************************************************************/
#ifndef __WRITE_CALLOUT_H__
#define __WRITE_CALLOUT_H__

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

//---------------------------------------------------------------------------
typedef bool (*WriteCalloutFunc)(uint16_t u16Addr_, uint8_t u8Data_);

//---------------------------------------------------------------------------
void WriteCallout_Add( WriteCalloutFunc pfCallout_, uint16_t u16Addr_ );

//---------------------------------------------------------------------------
bool WriteCallout_Run( uint16_t u16Addr_, uint8_t u8Data_ );


#endif
```