

fIAVR - Funkenstein Little AVR Virtual Runtime

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Data Structure Index</b>	<b>1</b>
1.1	Data Structures . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Data Structure Documentation</b>	<b>7</b>
3.1	_BreakPoint Struct Reference . . . . .	7
3.1.1	Detailed Description . . . . .	7
3.2	_IOClockList Struct Reference . . . . .	7
3.2.1	Detailed Description . . . . .	8
3.3	_IOReaderList Struct Reference . . . . .	8
3.3.1	Detailed Description . . . . .	8
3.4	_IOWriterList Struct Reference . . . . .	8
3.4.1	Detailed Description . . . . .	8
3.5	_WatchPoint Struct Reference . . . . .	9
3.5.1	Detailed Description . . . . .	9
3.5.2	Field Documentation . . . . .	9
3.5.2.1	u16Addr . . . . .	9
3.6	AddressCoverageTLV_t Struct Reference . . . . .	9
3.6.1	Detailed Description . . . . .	9
3.7	AVR_CoreRegisters Struct Reference . . . . .	10
3.7.1	Detailed Description . . . . .	11
3.8	AVR_CPU Struct Reference . . . . .	11

3.8.1 Detailed Description . . . . .	12
3.9 AVR_CPU_Config_t Struct Reference . . . . .	13
3.9.1 Detailed Description . . . . .	13
3.10 AVR_Feature_Map_t Struct Reference . . . . .	13
3.10.1 Detailed Description . . . . .	13
3.11 AVR_RAM_t Struct Reference . . . . .	13
3.11.1 Detailed Description . . . . .	14
3.12 AVR_Variant_t Struct Reference . . . . .	14
3.12.1 Detailed Description . . . . .	14
3.12.2 Field Documentation . . . . .	15
3.12.2.1 szName . . . . .	15
3.13 AVR_Vector_Map_t Struct Reference . . . . .	15
3.13.1 Detailed Description . . . . .	16
3.14 AVRPeripheral Struct Reference . . . . .	16
3.14.1 Detailed Description . . . . .	16
3.15 AVRRegisterFile Struct Reference . . . . .	16
3.15.1 Detailed Description . . . . .	21
3.16 Debug_Symbol_t Struct Reference . . . . .	21
3.16.1 Detailed Description . . . . .	21
3.17 DrawPoint_t Struct Reference . . . . .	22
3.17.1 Detailed Description . . . . .	22
3.18 FunctionCoverageTLV_t Struct Reference . . . . .	22
3.18.1 Detailed Description . . . . .	22
3.19 FunctionProfileTLV_t Struct Reference . . . . .	22
3.19.1 Detailed Description . . . . .	23
3.20 GDBCommandMap_t Struct Reference . . . . .	23
3.20.1 Detailed Description . . . . .	23
3.21 HEX_Record_t Struct Reference . . . . .	23
3.21.1 Detailed Description . . . . .	24
3.22 Interactive_Command_t Struct Reference . . . . .	24

3.22.1 Detailed Description . . . . .	24
3.23 Interrupt_Callout_ Struct Reference . . . . .	24
3.23.1 Detailed Description . . . . .	25
3.24 KernelAwareTrace_t Struct Reference . . . . .	25
3.24.1 Detailed Description . . . . .	25
3.25 Mark3_Context_t Struct Reference . . . . .	25
3.25.1 Detailed Description . . . . .	25
3.26 Mark3_Thread_Info_t Struct Reference . . . . .	26
3.26.1 Detailed Description . . . . .	26
3.27 Mark3_Thread_t Struct Reference . . . . .	26
3.27.1 Detailed Description . . . . .	27
3.28 Mark3ContextSwitch_TLV_t Struct Reference . . . . .	27
3.28.1 Detailed Description . . . . .	27
3.29 Mark3Interrupt_TLV_t Struct Reference . . . . .	27
3.29.1 Detailed Description . . . . .	27
3.30 Mark3Profile_TLV_t Struct Reference . . . . .	28
3.30.1 Detailed Description . . . . .	28
3.31 Option_t Struct Reference . . . . .	28
3.31.1 Detailed Description . . . . .	28
3.31.2 Field Documentation . . . . .	28
3.31.2.1 szAttribute . . . . .	29
3.32 Profile_t Struct Reference . . . . .	29
3.32.1 Detailed Description . . . . .	29
3.33 TLV_t Struct Reference . . . . .	29
3.33.1 Detailed Description . . . . .	30
3.34 TraceBuffer_t Struct Reference . . . . .	30
3.34.1 Detailed Description . . . . .	30
3.35 TraceElement_t Struct Reference . . . . .	30
3.35.1 Detailed Description . . . . .	31
3.36 Write_Callout_ Struct Reference . . . . .	31
3.36.1 Detailed Description . . . . .	31

<b>4 File Documentation</b>	<b>33</b>
4.1 src/avr_cpu/avr_coreregs.h File Reference	33
4.1.1 Detailed Description	33
4.2 avr_coreregs.h	33
4.3 src/avr_cpu/avr_cpu.c File Reference	35
4.3.1 Detailed Description	35
4.3.2 Function Documentation	36
4.3.2.1 CPU_AddPeriph()	36
4.3.2.2 CPU_Fetch()	36
4.3.2.3 CPU_Init()	36
4.3.2.4 CPU_RegisterInterruptCallback()	37
4.3.2.5 CPU_RunCycle()	37
4.4 avr_cpu.c	37
4.5 src/avr_cpu/avr_cpu.h File Reference	41
4.5.1 Detailed Description	41
4.5.2 Function Documentation	42
4.5.2.1 CPU_AddPeriph()	42
4.5.2.2 CPU_Fetch()	42
4.5.2.3 CPU_Init()	42
4.5.2.4 CPU_RegisterInterruptCallback()	43
4.5.2.5 CPU_RunCycle()	43
4.6 avr_cpu.h	43
4.7 src/avr_cpu/avr_cpu_print.c File Reference	45
4.7.1 Detailed Description	46
4.7.2 Function Documentation	46
4.7.2.1 print_core_regs()	46
4.7.2.2 print_io_reg()	46
4.7.2.3 print_io_reg_with_name()	47
4.7.2.4 print_ram()	47
4.7.2.5 print_rom()	47

4.8	<a href="#">avr_cpu_print.c</a>	48
4.9	<a href="#">src/avr_cpu/avr_cpu_print.h</a> File Reference	51
4.9.1	Detailed Description	51
4.9.2	Function Documentation	51
4.9.2.1	<a href="#">print_core_regs()</a>	51
4.9.2.2	<a href="#">print_io_reg()</a>	51
4.9.2.3	<a href="#">print_io_reg_with_name()</a>	52
4.9.2.4	<a href="#">print_ram()</a>	52
4.9.2.5	<a href="#">print_rom()</a>	53
4.10	<a href="#">avr_cpu_print.h</a>	53
4.11	<a href="#">src/avr_cpu/avr_disasm.c</a> File Reference	53
4.11.1	Detailed Description	57
4.11.2	Function Documentation	57
4.11.2.1	<a href="#">AVR_Disasm_Function()</a>	57
4.12	<a href="#">avr_disasm.c</a>	57
4.13	<a href="#">src/avr_cpu/avr_disasm.h</a> File Reference	79
4.13.1	Detailed Description	79
4.13.2	Function Documentation	79
4.13.2.1	<a href="#">AVR_Disasm_Function()</a>	79
4.14	<a href="#">avr_disasm.h</a>	80
4.15	<a href="#">src/avr_cpu/avr_interrupt.c</a> File Reference	80
4.15.1	Detailed Description	81
4.15.2	Function Documentation	81
4.15.2.1	<a href="#">AVR_ClearCandidate()</a>	81
4.15.2.2	<a href="#">AVR_Interrupt()</a>	81
4.15.2.3	<a href="#">AVR_InterruptCandidate()</a>	81
4.16	<a href="#">avr_interrupt.c</a>	82
4.17	<a href="#">src/avr_cpu/avr_interrupt.h</a> File Reference	83
4.17.1	Detailed Description	84
4.17.2	Function Documentation	84

4.17.2.1	AVR_ClearCandidate()	84
4.17.2.2	AVR_Interrupt()	84
4.17.2.3	AVR_InterruptCandidate()	84
4.18	avr_interrupt.h	85
4.19	src/avr_cpu/avr_io.c File Reference	85
4.19.1	Detailed Description	86
4.19.2	Function Documentation	86
4.19.2.1	IO_AddClocker()	86
4.19.2.2	IO_AddReader()	86
4.19.2.3	IO_AddWriter()	87
4.19.2.4	IO_Clock()	87
4.19.2.5	IO_Read()	87
4.19.2.6	IO_Write()	88
4.20	avr_io.c	88
4.21	src/avr_cpu/avr_io.h File Reference	89
4.21.1	Detailed Description	90
4.21.2	Function Documentation	90
4.21.2.1	IO_AddClocker()	90
4.21.2.2	IO_AddReader()	91
4.21.2.3	IO_AddWriter()	91
4.21.2.4	IO_Clock()	91
4.21.2.5	IO_Read()	92
4.21.2.6	IO_Write()	92
4.22	avr_io.h	92
4.23	src/avr_cpu/avr_op_cycles.c File Reference	93
4.23.1	Detailed Description	96
4.23.2	Function Documentation	96
4.23.2.1	AVR_Opcode_Cycles()	96
4.23.2.2	AVR_Opcode_Cycles_CALL()	97
4.23.2.3	AVR_Opcode_Cycles_CBI()	97



4.23.2.4	AVR_Opcode_Cycles_ICALL()	97
4.23.2.5	AVR_Opcode_Cycles_LD_Z_Indirect_Postinc()	97
4.23.2.6	AVR_Opcode_Cycles_LD_Z_Indirect_Predec()	98
4.23.2.7	AVR_Opcode_Cycles_RCALL()	98
4.23.2.8	AVR_Opcode_Cycles_RET()	98
4.23.2.9	AVR_Opcode_Cycles_RETI()	98
4.23.2.10	AVR_Opcode_Cycles_SBI()	98
4.23.2.11	AVR_Opcode_Cycles_SPM()	99
4.23.2.12	AVR_Opcode_Cycles_SPM_Z_Postinc2()	99
4.23.2.13	AVR_Opcode_Cycles_ST_X_Indirect()	99
4.23.2.14	AVR_Opcode_Cycles_ST_X_Indirect_Postinc()	99
4.23.2.15	AVR_Opcode_Cycles_ST_X_Indirect_Predec()	99
4.23.2.16	AVR_Opcode_Cycles_ST_Y_Indirect()	100
4.23.2.17	AVR_Opcode_Cycles_ST_Y_Indirect_Postinc()	100
4.23.2.18	AVR_Opcode_Cycles_ST_Y_Indirect_Predec()	100
4.23.2.19	AVR_Opcode_Cycles_ST_Z_Indirect()	100
4.23.2.20	AVR_Opcode_Cycles_ST_Z_Indirect_Postinc()	100
4.23.2.21	AVR_Opcode_Cycles_ST_Z_Indirect_Predec()	101
4.23.2.22	AVR_Opcode_Cycles_STD_Y()	101
4.23.2.23	AVR_Opcode_Cycles_STD_Z()	101
4.24	avr_op_cycles.c	101
4.25	src/avr_cpu/avr_op_cycles.h File Reference	114
4.25.1	Detailed Description	114
4.25.2	Function Documentation	114
4.25.2.1	AVR_Opcode_Cycles()	114
4.26	avr_op_cycles.h	115
4.27	src/avr_cpu/avr_op_decode.c File Reference	115
4.27.1	Detailed Description	116
4.27.2	Function Documentation	116
4.27.2.1	AVR_Decode()	116

4.27.2.2 AVR_Decoder_Function()	117
4.28 avr_op_decode.c	118
4.29 src/avr_cpu/avr_op_decode.h File Reference	122
4.29.1 Detailed Description	122
4.29.2 Function Documentation	122
4.29.2.1 AVR_Decode()	123
4.29.2.2 AVR_Decoder_Function()	123
4.30 avr_op_decode.h	124
4.31 src/avr_cpu/avr_op_size.c File Reference	125
4.31.1 Detailed Description	125
4.31.2 Function Documentation	125
4.31.2.1 AVR_Opcode_Size()	125
4.32 avr_op_size.c	126
4.33 src/avr_cpu/avr_op_size.h File Reference	129
4.33.1 Detailed Description	129
4.33.2 Function Documentation	129
4.33.2.1 AVR_Opcode_Size()	129
4.34 avr_op_size.h	130
4.35 src/avr_cpu/avr_opcodes.c File Reference	130
4.35.1 Detailed Description	134
4.35.2 Function Documentation	134
4.35.2.1 AVR_Opcode_DES()	134
4.35.2.2 AVR_Opcode_EICALL()	134
4.35.2.3 AVR_Opcode_EI JMP()	134
4.35.2.4 AVR_Opcode_Function()	134
4.35.2.5 AVR_Opcode_SPM()	135
4.35.2.6 AVR_Opcode_SPM_Z_Postinc2()	135
4.35.2.7 AVR_RunOpcode()	135
4.36 avr_opcodes.c	136
4.37 src/avr_cpu/avr_opcodes.h File Reference	160

4.37.1 Detailed Description . . . . .	160
4.37.2 Function Documentation . . . . .	161
4.37.2.1 AVR_Opcode_Function() . . . . .	161
4.37.2.2 AVR_RunOpcode() . . . . .	161
4.38 avr_opcodes.h . . . . .	161
4.39 src/avr_cpu/avr_registerfile.h File Reference . . . . .	162
4.39.1 Detailed Description . . . . .	162
4.40 avr_registerfile.h . . . . .	162
4.41 src/avr_cpu/interrupt_callout.c File Reference . . . . .	166
4.41.1 Detailed Description . . . . .	167
4.41.2 Function Documentation . . . . .	167
4.41.2.1 InterruptCallout_Add() . . . . .	167
4.41.2.2 InterruptCallout_Run() . . . . .	167
4.42 interrupt_callout.c . . . . .	168
4.43 src/avr_cpu/interrupt_callout.h File Reference . . . . .	168
4.43.1 Detailed Description . . . . .	169
4.43.2 Function Documentation . . . . .	169
4.43.2.1 InterruptCallout_Add() . . . . .	169
4.43.2.2 InterruptCallout_Run() . . . . .	169
4.44 interrupt_callout.h . . . . .	170
4.45 src/avr_cpu/write_callout.h File Reference . . . . .	170
4.45.1 Detailed Description . . . . .	171
4.45.2 Function Documentation . . . . .	171
4.45.2.1 WriteCallout_Add() . . . . .	171
4.45.2.2 WriteCallout_Run() . . . . .	171
4.46 write_callout.h . . . . .	172
4.47 src/config/emu_config.h File Reference . . . . .	172
4.47.1 Detailed Description . . . . .	172
4.47.2 Macro Definition Documentation . . . . .	173
4.47.2.1 CONFIG_TRACEBUFFER_SIZE . . . . .	173

4.47.2.2	FEATURE_USE_JUMPTABLES	173
4.48	emu_config.h	173
4.49	src/config/options.c File Reference	174
4.49.1	Detailed Description	174
4.49.2	Enumeration Type Documentation	175
4.49.2.1	OptionIndex_t	175
4.49.3	Function Documentation	175
4.49.3.1	Options_GetByName()	175
4.49.3.2	Options_Init()	175
4.49.3.3	Options_Parse()	176
4.49.3.4	Options_ParseElement()	176
4.49.3.5	Options_PrintUsage()	177
4.49.3.6	Options_SetDefaults()	177
4.49.4	Variable Documentation	177
4.49.4.1	astAttributes	177
4.50	options.c	178
4.51	src/config/variant.c File Reference	180
4.51.1	Detailed Description	180
4.51.2	Function Documentation	180
4.51.2.1	Variant_GetByName()	180
4.51.3	Variable Documentation	181
4.51.3.1	astVariants	181
4.51.3.2	stLargeAtMegaFeatures	181
4.51.3.3	stMediumAtMegaFeatures	182
4.51.3.4	stSmallAtMegaFeatures	182
4.52	variant.c	182
4.53	src/config/variant.h File Reference	185
4.53.1	Detailed Description	185
4.53.2	Function Documentation	185
4.53.2.1	Variant_GetByName()	185

4.54	variant.h . . . . .	186
4.55	src/debug/breakpoint.c File Reference . . . . .	187
4.55.1	Detailed Description . . . . .	187
4.55.2	Function Documentation . . . . .	187
4.55.2.1	BreakPoint_Delete() . . . . .	188
4.55.2.2	BreakPoint_EnabledAtAddress() . . . . .	189
4.55.2.3	BreakPoint_Insert() . . . . .	189
4.56	breakpoint.c . . . . .	190
4.57	src/debug/breakpoint.h File Reference . . . . .	191
4.57.1	Detailed Description . . . . .	191
4.57.2	Function Documentation . . . . .	192
4.57.2.1	BreakPoint_Delete() . . . . .	192
4.57.2.2	BreakPoint_EnabledAtAddress() . . . . .	192
4.57.2.3	BreakPoint_Insert() . . . . .	192
4.58	breakpoint.h . . . . .	193
4.59	src/debug/code_profile.c File Reference . . . . .	193
4.59.1	Detailed Description . . . . .	194
4.59.2	Function Documentation . . . . .	194
4.59.2.1	Profile_Hit() . . . . .	194
4.59.2.2	Profile_Init() . . . . .	195
4.59.2.3	Profile_Print() . . . . .	195
4.60	code_profile.c . . . . .	195
4.61	src/debug/code_profile.h File Reference . . . . .	199
4.61.1	Detailed Description . . . . .	199
4.61.2	Function Documentation . . . . .	199
4.61.2.1	Profile_Hit() . . . . .	200
4.61.2.2	Profile_Init() . . . . .	200
4.61.2.3	Profile_Print() . . . . .	200
4.62	code_profile.h . . . . .	201
4.63	src/debug/debug_sym.c File Reference . . . . .	201

4.63.1 Detailed Description . . . . .	202
4.63.2 Function Documentation . . . . .	202
4.63.2.1 Symbol_Add_Func() . . . . .	202
4.63.2.2 Symbol_Add_Obj() . . . . .	202
4.63.2.3 Symbol_Find_Func_By_Name() . . . . .	203
4.63.2.4 Symbol_Find_Obj_By_Name() . . . . .	203
4.63.2.5 Symbol_Func_At_Index() . . . . .	204
4.63.2.6 Symbol_Get_Func_Count() . . . . .	204
4.63.2.7 Symbol_Get_Obj_Count() . . . . .	205
4.63.2.8 Symbol_Obj_At_Index() . . . . .	205
4.64 debug_sym.c . . . . .	205
4.65 src/debug/debug_sym.h File Reference . . . . .	207
4.65.1 Detailed Description . . . . .	208
4.65.2 Function Documentation . . . . .	208
4.65.2.1 Symbol_Add_Func() . . . . .	208
4.65.2.2 Symbol_Add_Obj() . . . . .	208
4.65.2.3 Symbol_Find_Func_By_Name() . . . . .	209
4.65.2.4 Symbol_Find_Obj_By_Name() . . . . .	209
4.65.2.5 Symbol_Func_At_Index() . . . . .	210
4.65.2.6 Symbol_Get_Func_Count() . . . . .	210
4.65.2.7 Symbol_Get_Obj_Count() . . . . .	210
4.65.2.8 Symbol_Obj_At_Index() . . . . .	210
4.66 debug_sym.h . . . . .	211
4.67 src/debug/elf_print.c File Reference . . . . .	212
4.67.1 Function Documentation . . . . .	212
4.67.1.1 ELF_PrintHeader() . . . . .	212
4.67.1.2 ELF_PrintProgramHeaders() . . . . .	212
4.67.1.3 ELF_PrintSections() . . . . .	213
4.67.1.4 ELF_PrintSymbols() . . . . .	213
4.68 elf_print.c . . . . .	213

4.69	src/debug/elf_print.h File Reference	217
4.69.1	Detailed Description	217
4.69.2	Function Documentation	217
4.69.2.1	ELF_PrintHeader()	217
4.69.2.2	ELF_PrintProgramHeaders()	218
4.69.2.3	ELF_PrintSections()	218
4.69.2.4	ELF_PrintSymbols()	218
4.70	elf_print.h	219
4.71	src/debug/elf_types.h File Reference	219
4.71.1	Detailed Description	220
4.72	elf_types.h	221
4.73	src/debug/interactive.c File Reference	222
4.73.1	Detailed Description	224
4.73.2	Typedef Documentation	224
4.73.2.1	Interactive_Handler	224
4.73.3	Function Documentation	225
4.73.3.1	Interactive_Break()	225
4.73.3.2	Interactive_BreakFunc()	225
4.73.3.3	Interactive_CheckAndExecute()	226
4.73.3.4	Interactive_Continue()	226
4.73.3.5	Interactive_Disasm()	226
4.73.3.6	Interactive_EE()	227
4.73.3.7	Interactive_Help()	227
4.73.3.8	Interactive_Init()	228
4.73.3.9	Interactive_ListFunc()	228
4.73.3.10	Interactive_ListObj()	228
4.73.3.11	Interactive_Quit()	229
4.73.3.12	Interactive_RAM()	229
4.73.3.13	Interactive_Registers()	230
4.73.3.14	Interactive_ROM()	230

4.73.3.15 Interactive_Set()	231
4.73.3.16 Interactive_Step()	231
4.73.3.17 Interactive_Trace()	231
4.73.3.18 Interactive_Watch()	232
4.73.3.19 Interactive_WatchObj()	232
4.73.4 Variable Documentation	233
4.73.4.1 astCommands	233
4.74 interactive.c	233
4.75 src/debug/interactive.h File Reference	241
4.75.1 Detailed Description	241
4.75.2 Function Documentation	241
4.75.2.1 Interactive_CheckAndExecute()	242
4.75.2.2 Interactive_Init()	242
4.75.2.3 Interactive_Set()	242
4.76 interactive.h	242
4.77 src/debug/trace_buffer.c File Reference	243
4.77.1 Detailed Description	243
4.77.2 Function Documentation	244
4.77.2.1 TraceBuffer_Init()	244
4.77.2.2 TraceBuffer_LoadElement()	244
4.77.2.3 TraceBuffer_Print()	244
4.77.2.4 TraceBuffer_PrintElement()	245
4.77.2.5 TraceBuffer_StoreFromCPU()	245
4.78 trace_buffer.c	245
4.79 src/debug/trace_buffer.h File Reference	247
4.79.1 Detailed Description	248
4.79.2 Function Documentation	248
4.79.2.1 TraceBuffer_Init()	248
4.79.2.2 TraceBuffer_LoadElement()	248
4.79.2.3 TraceBuffer_Print()	249



4.79.2.4	TraceBuffer_PrintElement()	249
4.79.2.5	TraceBuffer_StoreFromCPU()	250
4.80	trace_buffer.h	250
4.81	src/debug/watchpoint.c File Reference	251
4.81.1	Detailed Description	251
4.81.2	Function Documentation	251
4.81.2.1	WatchPoint_Delete()	251
4.81.2.2	WatchPoint_EnabledAtAddress()	252
4.81.2.3	WatchPoint_Insert()	252
4.82	watchpoint.c	253
4.83	src/debug/watchpoint.h File Reference	254
4.83.1	Detailed Description	254
4.83.2	Function Documentation	255
4.83.2.1	WatchPoint_Delete()	255
4.83.2.2	WatchPoint_EnabledAtAddress()	255
4.83.2.3	WatchPoint_Insert()	255
4.84	watchpoint.h	256
4.85	src/flavr.c File Reference	256
4.85.1	Detailed Description	257
4.86	flavr.c	258
4.87	src/kernel_aware/ka_graphics.c File Reference	262
4.87.1	Detailed Description	263
4.88	ka_graphics.c	263
4.89	src/kernel_aware/ka_graphics.h File Reference	264
4.89.1	Detailed Description	265
4.90	ka_graphics.h	265
4.91	src/kernel_aware/ka_interrupt.c File Reference	265
4.91.1	Detailed Description	266
4.91.2	Function Documentation	266
4.91.2.1	KA_Interrupt_Init()	266

4.92	<a href="#">ka_interrupt.c</a>	266
4.93	<a href="#">src/kernel_aware/ka_interrupt.h File Reference</a>	267
4.93.1	Detailed Description	267
4.93.2	Function Documentation	267
4.93.2.1	<a href="#">KA_Interrupt_Init()</a>	268
4.94	<a href="#">ka_interrupt.h</a>	268
4.95	<a href="#">src/kernel_aware/ka_joystick.c File Reference</a>	268
4.95.1	Detailed Description	269
4.96	<a href="#">ka_joystick.c</a>	269
4.97	<a href="#">src/kernel_aware/ka_joystick.h File Reference</a>	271
4.97.1	Detailed Description	271
4.98	<a href="#">ka_joystick.h</a>	271
4.99	<a href="#">src/kernel_aware/ka_profile.c File Reference</a>	271
4.99.1	Detailed Description	272
4.99.2	Function Documentation	272
4.99.2.1	<a href="#">KA_Profile_Init()</a>	272
4.99.3	Variable Documentation	273
4.99.3.1	<a href="#">u64ProfileEpochStart</a>	273
4.100	<a href="#">ka_profile.c</a>	273
4.101	<a href="#">src/kernel_aware/ka_profile.h File Reference</a>	274
4.101.1	Detailed Description	275
4.101.2	Function Documentation	275
4.101.2.1	<a href="#">KA_Profile_Init()</a>	275
4.102	<a href="#">ka_profile.h</a>	275
4.103	<a href="#">src/kernel_aware/ka_thread.c File Reference</a>	276
4.103.1	Detailed Description	277
4.104	<a href="#">ka_thread.c</a>	277
4.105	<a href="#">src/kernel_aware/ka_thread.h File Reference</a>	283
4.105.1	Detailed Description	283
4.106	<a href="#">ka_thread.h</a>	284

4.107src/kernel_aware/ka_trace.c File Reference . . . . .	284
4.107.1 Detailed Description . . . . .	285
4.107.2 Function Documentation . . . . .	285
4.107.2.1 KA_EmitTrace() . . . . .	285
4.107.2.2 KA_Print() . . . . .	285
4.107.2.3 KA_Trace_Init() . . . . .	286
4.108ka_trace.c . . . . .	286
4.109src/kernel_aware/ka_trace.h File Reference . . . . .	287
4.109.1 Detailed Description . . . . .	287
4.109.2 Function Documentation . . . . .	288
4.109.2.1 KA_EmitTrace() . . . . .	288
4.109.2.2 KA_Print() . . . . .	288
4.109.2.3 KA_Trace_Init() . . . . .	288
4.110ka_trace.h . . . . .	289
4.111src/kernel_aware/kernel_aware.c File Reference . . . . .	289
4.111.1 Detailed Description . . . . .	290
4.111.2 Function Documentation . . . . .	290
4.111.2.1 KernelAware_Init() . . . . .	290
4.112kernel_aware.c . . . . .	290
4.113src/kernel_aware/kernel_aware.h File Reference . . . . .	291
4.113.1 Detailed Description . . . . .	292
4.113.2 Function Documentation . . . . .	292
4.113.2.1 KernelAware_Init() . . . . .	292
4.114kernel_aware.h . . . . .	293
4.115src/kernel_aware/tlv_file.c File Reference . . . . .	293
4.115.1 Detailed Description . . . . .	294
4.115.2 Function Documentation . . . . .	294
4.115.2.1 TLV_Alloc() . . . . .	294
4.115.2.2 TLV_Free() . . . . .	295
4.115.2.3 TLV_Read() . . . . .	295

4.115.2.4 TLV_ReadFinish()	295
4.115.2.5 TLV_ReadInit()	296
4.115.2.6 TLV_Write()	296
4.115.2.7 TLV_WriteInit()	297
4.116tlv_file.c	297
4.117src/kernel_aware/tlv_file.h File Reference	298
4.117.1 Detailed Description	299
4.117.2 Enumeration Type Documentation	299
4.117.2.1 FlavrTag_t	299
4.117.3 Function Documentation	300
4.117.3.1 TLV_Alloc()	300
4.117.3.2 TLV_Free()	301
4.117.3.3 TLV_Read()	301
4.117.3.4 TLV_ReadFinish()	301
4.117.3.5 TLV_ReadInit()	302
4.117.3.6 TLV_Write()	302
4.117.3.7 TLV_WriteInit()	303
4.118tlv_file.h	303
4.119src/loader/avr_loader.c File Reference	304
4.119.1 Detailed Description	305
4.119.2 Function Documentation	305
4.119.2.1 AVR_Load_ELF()	305
4.119.2.2 AVR_Load_HEX()	305
4.120avr_loader.c	306
4.121src/loader/avr_loader.h File Reference	308
4.121.1 Detailed Description	308
4.121.2 Function Documentation	308
4.121.2.1 AVR_Load_ELF()	308
4.121.2.2 AVR_Load_HEX()	309
4.122avr_loader.h	309

4.123src/loader/elf_process.c File Reference . . . . .	310
4.123.1 Detailed Description . . . . .	310
4.123.2 Function Documentation . . . . .	310
4.123.2.1 ELF_GetHeaderStringTableOffset() . . . . .	310
4.123.2.2 ELF_GetSymbolStringTableOffset() . . . . .	311
4.123.2.3 ELF_GetSymbolTableOffset() . . . . .	311
4.123.2.4 ELF_LoadFromFile() . . . . .	312
4.124elf_process.c . . . . .	312
4.125src/loader/elf_process.h File Reference . . . . .	314
4.125.1 Detailed Description . . . . .	314
4.125.2 Function Documentation . . . . .	314
4.125.2.1 ELF_GetHeaderStringTableOffset() . . . . .	314
4.125.2.2 ELF_GetSymbolStringTableOffset() . . . . .	315
4.125.2.3 ELF_GetSymbolTableOffset() . . . . .	315
4.125.2.4 ELF_LoadFromFile() . . . . .	316
4.126elf_process.h . . . . .	316
4.127src/loader/intel_hex.c File Reference . . . . .	317
4.127.1 Detailed Description . . . . .	317
4.127.2 Function Documentation . . . . .	317
4.127.2.1 HEX_Print_Record() . . . . .	317
4.127.2.2 HEX_Read_Record() . . . . .	318
4.128intel_hex.c . . . . .	318
4.129src/loader/intel_hex.h File Reference . . . . .	321
4.129.1 Detailed Description . . . . .	322
4.129.2 Function Documentation . . . . .	322
4.129.2.1 HEX_Print_Record() . . . . .	322
4.129.2.2 HEX_Read_Record() . . . . .	322
4.130intel_hex.h . . . . .	323
4.131src/peripheral/avr_peripheral.h File Reference . . . . .	324
4.131.1 Detailed Description . . . . .	324

4.132	avr_peripheral.h	324
4.133	src/peripheral/avr_periphregs.h File Reference	325
4.133.1	Detailed Description	326
4.134	avr_periphregs.h	326
4.135	src/peripheral/mega_eeprom.c File Reference	338
4.135.1	Detailed Description	339
4.135.2	Enumeration Type Documentation	339
4.135.2.1	EEPROM_Mode_t	339
4.135.2.2	EEPROM_State_t	340
4.135.3	Function Documentation	340
4.135.3.1	EEPROM_Write()	340
4.135.4	Variable Documentation	340
4.135.4.1	stEEPROM	340
4.136	mega_eeprom.c	341
4.137	src/peripheral/mega_eeprom.h File Reference	344
4.137.1	Detailed Description	344
4.138	mega_eeprom.h	345
4.139	src/peripheral/mega_eint.c File Reference	345
4.139.1	Detailed Description	346
4.139.2	Enumeration Type Documentation	346
4.139.2.1	InterruptSense_t	346
4.139.3	Function Documentation	346
4.139.3.1	EINT_Clock()	346
4.139.4	Variable Documentation	347
4.139.4.1	stEINT_a	347
4.139.4.2	stEINT_b	347
4.140	mega_eint.c	348
4.141	src/peripheral/mega_eint.h File Reference	352
4.141.1	Detailed Description	352
4.142	mega_eint.h	352

4.143src/peripheral/mega_timer16.c File Reference	353
4.143.1 Detailed Description	354
4.143.2 Enumeration Type Documentation	354
4.143.2.1 ClockSource_t	355
4.143.3 Function Documentation	355
4.143.3.1 Timer16_Clock()	355
4.143.4 Variable Documentation	355
4.143.4.1 stTimer16	355
4.143.4.2 stTimer16a	356
4.143.4.3 stTimer16b	356
4.144mega_timer16.c	356
4.145src/peripheral/mega_timer16.h File Reference	363
4.145.1 Detailed Description	364
4.146mega_timer16.h	364
4.147src/peripheral/mega_timer8.c File Reference	364
4.147.1 Detailed Description	366
4.147.2 Enumeration Type Documentation	366
4.147.2.1 ClockSource_t	366
4.147.3 Function Documentation	366
4.147.3.1 Timer8_Clock()	366
4.147.4 Variable Documentation	366
4.147.4.1 stTimer8	366
4.147.4.2 stTimer8a	367
4.147.4.3 stTimer8b	367
4.148mega_timer8.c	367
4.149src/peripheral/mega_timer8.h File Reference	373
4.149.1 Detailed Description	373
4.150mega_timer8.h	373
4.151src/peripheral/mega_uart.c File Reference	373
4.151.1 Detailed Description	375
4.151.2 Macro Definition Documentation	375
4.151.2.1 DEBUG_PRINT	375
4.151.3 Variable Documentation	375
4.151.3.1 stUART	375
4.152mega_uart.c	376
4.153src/peripheral/mega_uart.h File Reference	383
4.153.1 Detailed Description	383
4.154mega_uart.h	383





# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">_BreakPoint</a>	Node-structure for a linked-list of breakpoint addresses . . . . .	7
<a href="#">_IOClockList</a>	. . . . .	7
<a href="#">_IOReaderList</a>	. . . . .	8
<a href="#">_IOWriterList</a>	. . . . .	8
<a href="#">_WatchPoint</a>	. . . . .	9
<a href="#">AddressCoverageTLV_t</a>	. . . . .	9
<a href="#">AVR_CoreRegisters</a>	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 . . . . .	10
<a href="#">AVR_CPU</a>	This structure effectively represents an entire simulated AVR CPU - all memories, registers (memory-mapped or internal), peripherals and housekeeping information . . . . .	11
<a href="#">AVR_CPU_Config_t</a>	Struct defining parameters used to initialize the AVR CPU structure on startup . . . . .	13
<a href="#">AVR_Feature_Map_t</a>	. . . . .	13
<a href="#">AVR_RAM_t</a>	Union structure mapping the first 256 bytes of IO address space to an array of bytes used to represent CPU RAM . . . . .	13
<a href="#">AVR_Variant_t</a>	This struct contains the information necessary to effectively describe an AVR Microcontroller variant among the rest of the code . . . . .	14
<a href="#">AVR_Vector_Map_t</a>	. . . . .	15
<a href="#">AVRPeripheral</a>	. . . . .	16
<a href="#">AVRRegisterFile</a>	The first 256 bytes of the AVR memory space is composed of the core 32 general-purpose registers (R0-R31), and 224 bytes for the remaining I/O registers (used by peripherals) . . . . .	16
<a href="#">Debug_Symbol_t</a>	. . . . .	21
<a href="#">DrawPoint_t</a>	. . . . .	22
<a href="#">FunctionCoverageTLV_t</a>	. . . . .	22
<a href="#">FunctionProfileTLV_t</a>	. . . . .	22
<a href="#">GDBCommandMap_t</a>	. . . . .	23
<a href="#">HEX_Record_t</a>	Data type used to represent a single Intel Hex Record . . . . .	23
<a href="#">Interactive_Command_t</a>	Struct type used to map debugger command-line inputs to command handlers . . . . .	24

<a href="#">Interrupt_Callout_</a> . . . . .	24
<a href="#">KernelAwareTrace_t</a> . . . . .	25
<a href="#">Mark3_Context_t</a> . . . . .	25
<a href="#">Mark3_Thread_Info_t</a> . . . . .	26
<a href="#">Mark3_Thread_t</a> . . . . .	26
<a href="#">Mark3ContextSwitch_TLV_t</a> . . . . .	27
<a href="#">Mark3Interrupt_TLV_t</a> . . . . .	27
<a href="#">Mark3Profile_TLV_t</a> . . . . .	28
<a href="#">Option_t</a>	
Local data structure used to define a command-line option . . . . .	28
<a href="#">Profile_t</a> . . . . .	29
<a href="#">TLV_t</a> . . . . .	29
<a href="#">TraceBuffer_t</a>	
Implements a circular buffer of trace elements, sized according to the compile-time configuration	30
<a href="#">TraceElement_t</a>	
Struct defining the CPU's running state at each tracebuffer sample point . . . . .	30
<a href="#">Write_Callout_</a> . . . . .	31

## Chapter 2

# File Index

### 2.1 File List

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

src/ <a href="#">flavr.c</a>	Main AVR emulator entrypoint, commandline-use with built-in interactive debugger . . . . .	256
src/avr_cpu/ <a href="#">avr_coreregs.h</a>	Module containing struct definition for the core AVR registers . . . . .	33
src/avr_cpu/ <a href="#">avr_cpu.c</a>	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 . . . . .	35
src/avr_cpu/ <a href="#">avr_cpu.h</a>	AVR CPU Emulator - virtual AVR structure declarations and functions used to drive the emulator (fetch/decode/execute) . . . . .	41
src/avr_cpu/ <a href="#">avr_cpu_print.c</a>	Helper module used to print the contents of a virtual AVR's internal registers and memory . . . . .	45
src/avr_cpu/ <a href="#">avr_cpu_print.h</a>	Helper module used to print the contents of a virtual AVR's internal registers and memory . . . . .	51
src/avr_cpu/ <a href="#">avr_disasm.c</a>	AVR Disassembler Implementation . . . . .	53
src/avr_cpu/ <a href="#">avr_disasm.h</a>	AVR Disassembler Implementation . . . . .	79
src/avr_cpu/ <a href="#">avr_interrupt.c</a>	CPU Interrupt management . . . . .	80
src/avr_cpu/ <a href="#">avr_interrupt.h</a>	AVR CPU Interrupt management - functionality responsible for arming/ disarming/processing CPU interrupts generated during the course of normal operation . . . . .	83
src/avr_cpu/ <a href="#">avr_io.c</a>	Interface to connect I/O register updates to their corresponding peripheral plugins . . . . .	85
src/avr_cpu/ <a href="#">avr_io.h</a>	Interface to connect I/O register updates to their corresponding peripheral plugins . . . . .	89
src/avr_cpu/ <a href="#">avr_op_cycles.c</a>	Opcode cycle counting functions . . . . .	93
src/avr_cpu/ <a href="#">avr_op_cycles.h</a>	Opcode cycle counting functions . . . . .	114
src/avr_cpu/ <a href="#">avr_op_decode.c</a>	Module providing logic to decode AVR CPU Opcodes . . . . .	115
src/avr_cpu/ <a href="#">avr_op_decode.h</a>	Module providing logic to decode AVR CPU Opcodes . . . . .	122

src/avr_cpu/avr_op_size.c	
Module providing opcode sizes . . . . .	125
src/avr_cpu/avr_op_size.h	
Module providing an interface to lookup the size of an opcode . . . . .	129
src/avr_cpu/avr_opcodes.c	
AVR CPU - Opcode implementation . . . . .	130
src/avr_cpu/avr_opcodes.h	
AVR CPU - Opcode interface . . . . .	160
src/avr_cpu/avr_registerfile.h	
Module providing a mapping of IO memory to the AVR register file . . . . .	162
src/avr_cpu/interrupt_callout.c	
Module providing functionality allowing emulator extensions to be triggered on interrupts . . . . .	166
src/avr_cpu/interrupt_callout.h	
Module providing functionality allowing emulator extensions to be triggered on interrupts . . . . .	168
src/avr_cpu/write_callout.c . . . . .	??
src/avr_cpu/write_callout.h	
Extended emulator functionality allowing for functions to be triggered based on RAM-write operations . . . . .	170
src/config/emu_config.h	
Configuration file - used to configure features used by the emulator at build-time . . . . .	172
src/config/options.c	
Module for managing command-line options . . . . .	174
src/config/options.h . . . . .	??
src/config/variant.c	
Module containing a table of device variants supported by flavr . . . . .	180
src/config/variant.h	
Module containing a lookup table of device variants supported by flavr . . . . .	185
src/debug/breakpoint.c	
Implements instruction breakpoints for debugging based on code path . . . . .	187
src/debug/breakpoint.h	
Implements instruction breakpoints for debugging based on code path . . . . .	191
src/debug/code_profile.c	
Code profiling (exeuction and coverage) functionality . . . . .	193
src/debug/code_profile.h	
Code profiling (exeuction and coverage) functionality . . . . .	199
src/debug/debug_sym.c	
Symbolic debugging support for data and functions . . . . .	201
src/debug/debug_sym.h	
Symbolic debugging support for data and functions . . . . .	207
src/debug/elf_print.c . . . . .	212
src/debug/elf_print.h	
Functions to print information from ELF files . . . . .	217
src/debug/elf_types.h	
Defines and types used by ELF loader and supporting functionality . . . . .	219
src/debug/gdb_rsp.c . . . . .	??
src/debug/gdb_rsp.h . . . . .	??
src/debug/interactive.c	
Interactive debugging support . . . . .	222
src/debug/interactive.h	
Interactive debugging support . . . . .	241
src/debug/trace_buffer.c	
Implements a circular buffer containing a history of recently executed instructions, along with core register context for each . . . . .	243
src/debug/trace_buffer.h	
Implements a circular buffer containing a history of recently executed instructions, along with core register context for each . . . . .	247

src/debug/ <a href="#">watchpoint.c</a>	Implements data watchpoints for debugging running programs based on reads/writes to a given memory address . . . . .	251
src/debug/ <a href="#">watchpoint.h</a>	Implements data watchpoints for debugging running programs based on reads/writes to a given memory address . . . . .	254
src/kernel_aware/ <a href="#">ka_graphics.c</a>	Mark3 RTOS Kernel-Aware graphics library . . . . .	262
src/kernel_aware/ <a href="#">ka_graphics.h</a>	Mark3 RTOS Kernel-Aware graphics library . . . . .	264
src/kernel_aware/ <a href="#">ka_interrupt.c</a>	Mark3 RTOS Kernel-Aware Interrupt Logging . . . . .	265
src/kernel_aware/ <a href="#">ka_interrupt.h</a>	Mark3 RTOS Kernel-Aware Interrupt Logging . . . . .	267
src/kernel_aware/ <a href="#">ka_joystick.c</a>	Mark3 RTOS Kernel-Aware graphics library . . . . .	268
src/kernel_aware/ <a href="#">ka_joystick.h</a>	Mark3 RTOS Kernel-Aware graphics library . . . . .	271
src/kernel_aware/ <a href="#">ka_profile.c</a>	Mark3 RTOS Kernel-Aware Profiling . . . . .	271
src/kernel_aware/ <a href="#">ka_profile.h</a>	Mark3 RTOS Kernel-Aware Profiling . . . . .	274
src/kernel_aware/ <a href="#">ka_stubs.c</a>		??
src/kernel_aware/ <a href="#">ka_thread.c</a>	Mark3 RTOS Kernel-Aware Thread Profiling . . . . .	276
src/kernel_aware/ <a href="#">ka_thread.h</a>	Mark3 RTOS Kernel-Aware Thread Profiling . . . . .	283
src/kernel_aware/ <a href="#">ka_trace.c</a>	Mark3 RTOS Kernel-Aware Trace functionality . . . . .	284
src/kernel_aware/ <a href="#">ka_trace.h</a>	Mark3 RTOS Kernel-Aware Trace and Print Functionality . . . . .	287
src/kernel_aware/ <a href="#">kernel_aware.c</a>	Mark3 RTOS Kernel-Aware debugger . . . . .	289
src/kernel_aware/ <a href="#">kernel_aware.h</a>	Kernel-Aware debugger plugin interface . . . . .	291
src/kernel_aware/ <a href="#">tlv_file.c</a>	Tag-length-value file format used for encoding simulator run-time data (kernel-aware plugin data, code profiling statistics, etc.) . . . . .	293
src/kernel_aware/ <a href="#">tlv_file.h</a>	Tag-length-value file format used for encoding simulator run-time data (kernel-aware plugin data, code profiling statistics, etc.) . . . . .	298
src/loader/ <a href="#">avr_loader.c</a>	Functions to load intel-formatted programming files into a virtual AVR . . . . .	304
src/loader/ <a href="#">avr_loader.h</a>	Functions to load intel hex or elf binaries into a virtual AVR . . . . .	308
src/loader/ <a href="#">elf_process.c</a>	Functions used to process ELF Binaries . . . . .	310
src/loader/ <a href="#">elf_process.h</a>	Functions used to process ELF Binaries . . . . .	314
src/loader/ <a href="#">intel_hex.c</a>	Module for decoding Intel hex formatted programming files . . . . .	317
src/loader/ <a href="#">intel_hex.h</a>	Module for decoding Intel hex formatted programming files . . . . .	321
src/peripheral/ <a href="#">avr_peripheral.h</a>	Interfaces for creating AVR peripheral plugins . . . . .	324
src/peripheral/ <a href="#">avr_periphregs.h</a>	Module defining bitfield/register definitions for memory-mapped peripherals located within IO memory space . . . . .	325

src/peripheral/ <a href="#">mega_eeprom.c</a>	
AVR atmega EEPROM plugin . . . . .	338
src/peripheral/ <a href="#">mega_eeprom.h</a>	
AVR atmega EEPROM plugin . . . . .	344
src/peripheral/ <a href="#">mega_eint.c</a>	
ATMega External Interrupt Implementation . . . . .	345
src/peripheral/ <a href="#">mega_eint.h</a>	
ATMega External Interrupt Implementation . . . . .	352
src/peripheral/ <a href="#">mega_timer16.c</a>	
ATMega 16-bit timer implementation . . . . .	353
src/peripheral/ <a href="#">mega_timer16.h</a>	
ATMega 16-bit timer implementation . . . . .	363
src/peripheral/ <a href="#">mega_timer8.c</a>	
ATMega 8-bit timer implementation . . . . .	364
src/peripheral/ <a href="#">mega_timer8.h</a>	
ATMega 8-bit timer implementation . . . . .	373
src/peripheral/ <a href="#">mega_uart.c</a>	
Implements an atmega UART plugin . . . . .	373
src/peripheral/ <a href="#">mega_uart.h</a>	
ATMega UART implementation . . . . .	383

## 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.*
- uint32\_t `u32Addr`  
*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:

- [src/debug/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:

- [src/avr\\_cpu/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:

- [src/avr\\_cpu/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:

- [src/avr\\_cpu/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 `u16Addr`

```
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:

- [src/debug/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:

- [src/debug/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

- ```

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
    }
}
};

```

### 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:

- [src/avr\\_cpu/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**
- uint32\_t **u32PC**
- 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**
- 
- union {
 uint32\_t **k**
 int32\_t **k\_s**
 };
- 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**
- const [AVR\\_Vector\\_Map\\_t](#) \* **pstVectorMap**
- const [AVR\\_Feature\\_Map\\_t](#) \* **pstFeatureMap**

### 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 64 of file [avr\\_cpu.h](#).

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

- src/avr\_cpu/[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**
- const [AVR\\_Vector\\_Map\\_t](#) \* **pstVectorMap**
- const [AVR\\_Feature\\_Map\\_t](#) \* **pstFeatureMap**

### 3.9.1 Detailed Description

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

Definition at line 151 of file [avr\\_cpu.h](#).

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

- [src/avr\\_cpu/avr\\_cpu.h](#)

## 3.10 AVR\_Feature\_Map\_t Struct Reference

### Data Fields

- bool **bHasTimer3**
- bool **bHasUSART1**
- bool **bHasInt2**
- bool **bHasPCInt3**

### 3.10.1 Detailed Description

Definition at line 70 of file [variant.h](#).

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

- [src/config/variant.h](#)

## 3.11 AVR\_RAM\_t Struct Reference

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

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

## Data Fields

- ```
union {
    AVRRegisterFile stRegisters
    uint8_t au8RAM [sizeof(AVRRegisterFile)]
};
```

### 3.11.1 Detailed Description

union structure mapping the first 256 bytes of IO address space to an array 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 48 of file [avr\\_cpu.h](#).

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

- [src/avr\\_cpu/avr\\_cpu.h](#)

## 3.12 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 [AVR\\_Feature\\_Map\\_t](#) \* [pstFeatures](#)  
*CPU Feature flags.*
- const [AVR\\_Vector\\_Map\\_t](#) \* [pstVectors](#)  
*Interrupt vector mappings.*

### 3.12.1 Detailed Description

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

Definition at line 82 of file [variant.h](#).

### 3.12.2 Field Documentation

#### 3.12.2.1 szName

```
const char* AVR_Variant_t::szName
```

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

"atmega328p")

Definition at line 84 of file [variant.h](#).

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

- [src/config/variant.h](#)

## 3.13 AVR\_Vector\_Map\_t Struct Reference

### Data Fields

- uint8\_t **RESET**
- uint8\_t **INT0**
- uint8\_t **INT1**
- uint8\_t **INT2**
- uint8\_t **PCINT0**
- uint8\_t **PCINT1**
- uint8\_t **PCINT2**
- uint8\_t **PCINT3**
- uint8\_t **WDT**
- uint8\_t **TIMER2\_COMPA**
- uint8\_t **TIMER2\_COMPB**
- uint8\_t **TIMER2\_OVF**
- uint8\_t **TIMER1\_CAPT**
- uint8\_t **TIMER1\_COMPA**
- uint8\_t **TIMER1\_COMPB**
- uint8\_t **TIMER1\_OVF**
- uint8\_t **TIMER0\_COMPA**
- uint8\_t **TIMER0\_COMPB**
- uint8\_t **TIMER0\_OVF**
- uint8\_t **SPI\_STC**
- uint8\_t **USART0\_RX**
- uint8\_t **USART0\_UDRE**
- uint8\_t **USART0\_TX**
- uint8\_t **ANALOG\_COMP**
- uint8\_t **ADC**
- uint8\_t **EE\_READY**
- uint8\_t **TWI**
- uint8\_t **SPM\_READY**
- uint8\_t **USART1\_RX**
- uint8\_t **USART1\_UDRE**
- uint8\_t **USART1\_TX**
- uint8\_t **TIMER3\_CAPT**
- uint8\_t **TIMER3\_COMPA**
- uint8\_t **TIMER3\_COMPB**
- uint8\_t **TIMER3\_OVF**

### 3.13.1 Detailed Description

Definition at line 31 of file [variant.h](#).

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

- [src/config/variant.h](#)

## 3.14 AVRPeripheral Struct Reference

### Data Fields

- PeriphInit **pfInit**
- PeriphRead **pfRead**
- PeriphWrite **pfWrite**
- PeriphClock **pfClock**
- void \* **pvContext**
- uint8\_t **u8AddrStart**
- uint8\_t **u8AddrEnd**

### 3.14.1 Detailed Description

Definition at line 41 of file [avr\\_peripheral.h](#).

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

- [src/peripheral/avr\\_peripheral.h](#)

## 3.15 AVRRegisterFile Struct Reference

The first 256 bytes of the AVR memory space is composed of the core 32 general-purpose 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 **RAMPZ**
- 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 **ADMUX**
- 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**

### 3.15.1 Detailed Description

The first 256 bytes of the AVR memory space is composed of the core 32 general-purpose 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 peripherals found on the atmega328p.

Definition at line 38 of file [avr\\_registerfile.h](#).

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

- src/avr\_cpu/[avr\\_registerfile.h](#)

## 3.16 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.*

### 3.16.1 Detailed Description

Definition at line 36 of file [debug\\_sym.h](#).

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

- src/debug/[debug\\_sym.h](#)

## 3.17 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.*

### 3.17.1 Detailed Description

Definition at line 39 of file [ka\\_graphics.c](#).

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

- [src/kernel\\_aware/ka\\_graphics.c](#)

## 3.18 FunctionCoverageTLV\_t Struct Reference

### Data Fields

- uint32\_t **u32FunctionSize**
- uint32\_t **u32AddressesHit**
- char **szSymName** [256]

### 3.18.1 Detailed Description

Definition at line 47 of file [code\\_profile.c](#).

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

- [src/debug/code\\_profile.c](#)

## 3.19 FunctionProfileTLV\_t Struct Reference

### Data Fields

- uint64\_t **u64CyclesTotal**
- uint64\_t **u64CPUCycles**
- char **szSymName** [256]

### 3.19.1 Detailed Description

Definition at line 39 of file [code\\_profile.c](#).

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

- [src/debug/code\\_profile.c](#)

## 3.20 GDBCommandMap\_t Struct Reference

### Data Fields

- GDBCommandType\_t **eCmd**
- const char \* **szToken**
- GDBCommandHandler\_t **pfHandler**

### 3.20.1 Detailed Description

Definition at line 66 of file [gdb\\_rsp.c](#).

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

- [src/debug/gdb\\_rsp.c](#)

## 3.21 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.*

### 3.21.1 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:

- [src/loader/intel\\_hex.h](#)

## 3.22 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.*

### 3.22.1 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:

- [src/debug/interactive.c](#)

## 3.23 Interrupt\_Callout\_ Struct Reference

### Data Fields

- `struct Interrupt\_Callout\_ * pstNext`  
*Next interrupt callout.*
- `InterruptCalloutFunc pfCallout`  
*Callout function.*



### 3.23.1 Detailed Description

Definition at line 29 of file [interrupt\\_callout.c](#).

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

- [src/avr\\_cpu/interrupt\\_callout.c](#)

## 3.24 KernelAwareTrace\_t Struct Reference

### Data Fields

- uint16\_t **u16File**
- uint16\_t **u16Line**
- uint16\_t **u16Code**
- uint16\_t **u16Arg1**
- uint16\_t **u16Arg2**

### 3.24.1 Detailed Description

Definition at line 34 of file [ka\\_trace.c](#).

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

- [src/kernel\\_aware/ka\\_trace.c](#)

## 3.25 Mark3\_Context\_t Struct Reference

### Data Fields

- uint8\_t **SPH**
- uint8\_t **SPL**
- uint8\_t **r** [32]
- uint8\_t **SREG**
- uint16\_t **PC**

### 3.25.1 Detailed Description

Definition at line 26 of file [ka\\_thread.h](#).

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

- [src/kernel\\_aware/ka\\_thread.h](#)

## 3.26 Mark3\_Thread\_Info\_t Struct Reference

### Data Fields

- [Mark3\\_Thread\\_t](#) \* **pstThread**
- [uint8\\_t](#) **u8ThreadID**
- [uint64\\_t](#) **u64TotalCycles**
- [uint64\\_t](#) **u64EpockCycles**
- [bool](#) **bActive**

### 3.26.1 Detailed Description

Definition at line 84 of file [ka\\_thread.c](#).

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

- [src/kernel\\_aware/ka\\_thread.c](#)

## 3.27 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 inheritance)*
- [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 entypoint.*
- [uint16\\_t](#) **u16Quantum**  
*Thread quantum (in milliseconds)*

### 3.27.1 Detailed Description

Definition at line 41 of file [ka\\_thread.c](#).

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

- [src/kernel\\_aware/ka\\_thread.c](#)

## 3.28 Mark3ContextSwitch\_TLV\_t Struct Reference

### Data Fields

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

### 3.28.1 Detailed Description

Definition at line 94 of file [ka\\_thread.c](#).

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

- [src/kernel\\_aware/ka\\_thread.c](#)

## 3.29 Mark3Interrupt\_TLV\_t Struct Reference

### Data Fields

- `uint64_t u64TimeStamp`
- `uint8_t u8Vector`
- `bool bEntry`

### 3.29.1 Detailed Description

Definition at line 38 of file [ka\\_interrupt.c](#).

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

- [src/kernel\\_aware/ka\\_interrupt.c](#)

### 3.30 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.*

#### 3.30.1 Detailed Description

Definition at line 44 of file [ka\\_profile.c](#).

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

- [src/kernel\\_aware/ka\\_profile.c](#)

### 3.31 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)*

#### 3.31.1 Detailed Description

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

Definition at line 31 of file [options.c](#).

#### 3.31.2 Field Documentation

### 3.31.2.1 szAttribute

```
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:

- [src/config/options.c](#)

## 3.32 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.*

### 3.32.1 Detailed Description

Definition at line 31 of file [code\\_profile.c](#).

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

- [src/debug/code\\_profile.c](#)

## 3.33 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)*

### 3.33.1 Detailed Description

Definition at line 53 of file [tlv\\_file.h](#).

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

- [src/kernel\\_aware/tlv\\_file.h](#)

## 3.34 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.*

### 3.34.1 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:

- [src/debug/trace\\_buffer.h](#)

## 3.35 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 u32PC`  
*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*

### 3.35.1 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:

- [src/debug/trace\\_buffer.h](#)

## 3.36 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.*

### 3.36.1 Detailed Description

Definition at line 31 of file [write\\_callout.c](#).

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

- [src/avr\\_cpu/write\\_callout.c](#)





## Chapter 4

# File Documentation

### 4.1 src/avr\_cpu/avr\_coreregs.h File Reference

Module containing struct definition for the core AVR registers.

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

#### Data Structures

- struct [AVR\\_CoreRegisters](#)

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

#### 4.1.1 Detailed Description

Module containing struct definition for the core AVR registers.

Definition in file [avr\\_coreregs.h](#).

### 4.2 avr\_coreregs.h

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( ()/(      (      | -- [ Funkenstein ] -----
00005 *      /( ) )/( ) )((( ( )\ )\ /( ) ) -- [ Little ] -----
00006 *      ( )_ ( ) )      )\ _ )\ ( ( ( ) ( ) ) -- [ AVR ] -----
00007 *      | _ | |      ( )_ \ ( ) \ \ / / | _ \ -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ V / | _ / -- [ Runtime ] -----
00009 *      | _ | | _ _ | / _ \ \ \ \ / | _ \
00010 *      | "Yeah, it does Arduino..."
00011 *      -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __AVR_COREREG_H__
00022 #define __AVR_COREREG_H__
00023
00024 #include <stdint.h>
00025
```

```
00038 typedef struct
00039 {
00040     union
00041     {
00042         uint8_t r[32];
00043         uint16_t r_word[16];
00044         struct
00045         {
00046             uint16_t r1_0;
00047             uint16_t r3_2;
00048             uint16_t r5_4;
00049             uint16_t r7_6;
00050             uint16_t r9_8;
00051             uint16_t r11_10;
00052             uint16_t r13_12;
00053             uint16_t r15_14;
00054             uint16_t r17_16;
00055             uint16_t r19_18;
00056             uint16_t r21_20;
00057             uint16_t r23_22;
00058             uint16_t r25_24;
00059             uint16_t r27_26;
00060             uint16_t r29_28;
00061             uint16_t r31_30;
00062         };
00063         struct
00064         {
00065             uint8_t r0;
00066             uint8_t r1;
00067             uint8_t r2;
00068             uint8_t r3;
00069             uint8_t r4;
00070             uint8_t r5;
00071             uint8_t r6;
00072             uint8_t r7;
00073             uint8_t r8;
00074             uint8_t r9;
00075             uint8_t r10;
00076             uint8_t r11;
00077             uint8_t r12;
00078             uint8_t r13;
00079             uint8_t r14;
00080             uint8_t r15;
00081             uint8_t r16;
00082             uint8_t r17;
00083             uint8_t r18;
00084             uint8_t r19;
00085             uint8_t r20;
00086             uint8_t r21;
00087             uint8_t r22;
00088             uint8_t r23;
00089             uint8_t r24;
00090             uint8_t r25;
00091             union
00092             {
00093                 uint16_t X;
00094                 struct
00095                 {
00096                     uint8_t r26;
00097                     uint8_t r27;
00098                 };
00099             };
00100             union
00101             {
00102                 uint16_t Y;
00103                 struct
00104                 {
00105                     uint8_t r28;
00106                     uint8_t r29;
00107                 };
00108             };
00109             union
00110             {
00111                 uint16_t Z;
00112                 struct
00113                 {
00114                     uint8_t r30;
00115                     uint8_t r31;
00116                 };
00117             };
00118         };
00119     };
00120 } AVR_CoreRegisters;
00121
00122 #endif
```

## 4.3 src/avr\_cpu/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.*
- 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 pflntAck\_, uint8\_t ucVector\_)
  - CPU\_RegisterInterruptCallback.*

### Variables

- **AVR\_CPU stCPU**

#### 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 CPU\_AddPeriph()

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

CPU\_AddPeriph Add a new I/O Peripheral to the CPU.

#### Parameters

<i>pstPeriph_</i>	Pointer to an initialized AVR Peripheral object to be associated with this CPU.
-------------------	---

Definition at line 268 of file [avr\\_cpu.c](#).

### 4.3.2.2 CPU\_Fetch()

```
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 CPU\_Init()

```
void CPU_Init (
    AVR_CPU_Config_t * pstConfig_ )
```

CPU\_Init Initialize the CPU object and its associated data.

#### Parameters

<i>pstConfig_</i>	Pointer to an initialized <a href="#">AVR_CPU_Config_t</a> struct
-------------------	---

Definition at line 227 of file [avr\\_cpu.c](#).

#### 4.3.2.4 CPU\_RegisterInterruptCallback()

```
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

<i>pfIntAck_</i>	Callback function to register
<i>ucVector_</i>	Interrupt vector index to install handler at

Definition at line 286 of file [avr\\_cpu.c](#).

#### 4.3.2.5 CPU\_RunCycle()

```
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](#).

## 4.4 avr\_cpu.c

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      )\ ) |
00004 *      ((/( ((/(      \      ( ((/( | -- [ Funkenstein ] -----
00005 *      /( ) /( ) ((( ( )\ )\ /( ) | -- [ Little ] -----
00006 *      ( ) ( )      )\ _ )\ ( ( ( ) ( ) | -- [ AVR ] -----
00007 *      | | _ | |      ( ) _\ ( )\ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ | / _ \ \ / \ / | _ \ | | -- [ Runtime ] -----
00009 *      | _ | | _ | / _ \ \ / \ / | _ \ | |
00010 *      | _ | | _ | / _ \ \ / \ / | _ \ | | "Yeah, it does Arduino..."
00011 *      +-----+
00012 * (c) Copyright 2014-17, 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.u32PC;
00090     if (PC >= 16384)
00091     {
00092         return 0xFFFF;
00093     }
00094     return stCPU.pul6ROM[ stCPU.u32PC ];
00095 }
00096
00097 //-----
00098 static void CPU_GetOpCycles( uint16_t OP_ )
00099 {
00100     #if FEATURE_USE_JUMPTABLES
00101         stCPU.ul6ExtraCycles = au8OpCycles[ OP_ ];
00102     #else
00103         stCPU.ul6ExtraCycles = AVR_Opcode_Cycles( OP_ );
00104     #endif
00105 }
00106
00107 //-----
00108 static void CPU_GetOpSize( uint16_t OP_ )
00109 {
00110     #if FEATURE_USE_JUMPTABLES
00111         stCPU.ul6ExtraPC = au8OpSizes[ OP_ ];
00112     #else
00113         stCPU.ul6ExtraPC = AVR_Opcode_Size( OP_ );
00114     #endif
00115 }
00116
00117 //-----
00118 static void CPU_PeripheralCycle( void )
00119 {
00120     IO_Clock();
00121 }
00122
00123 //-----
00124 void CPU_RunCycle( void )
00125 {

```

```

00126     uint16_t OP;
00127
00128     if (!stCPU.bAsleep)
00129     {
00130
00131         OP = CPU_Fetch();
00132
00133         // From the first word fetched, figure out how big this opcode is
00134         // (either 16 or 32-bit)
00135         CPU_GetOpSize( OP );
00136
00137         // Based on the first word fetched, figure out the minimum number of
00138         // CPU cycles required to execute the instruction fetched.
00139         CPU_GetOpCycles( OP );
00140
00141         // Decode the instruction, load internal registers with appropriate
00142         // values.
00143         CPU_Decode( OP );
00144
00145         // Execute the instruction that was just decoded
00146         CPU_Execute( OP );
00147
00148         // Update the PC based on the size of the instruction + whatever
00149         // modifications occurred during the execution cycle.
00150         stCPU.u32PC += stCPU.u16ExtraPC;
00151
00152         // Add CPU clock cycles to the global cycle counter based on
00153         // the minimum instruction time, plus whatever modifiers are applied
00154         // during execution of the instruction.
00155         stCPU.u64CycleCount += stCPU.u16ExtraCycles;
00156
00157         // Cycle-accurate peripheral clocking -- one iteration for each
00158         // peripheral for each CPU cycle of the instruction.
00159         // Note that CPU Interrupts are generated in the peripheral
00160         // phase of the instruction cycle.
00161         while (stCPU.u16ExtraCycles--)
00162         {
00163             CPU_PeripheralCycle();
00164         }
00165
00166         // Increment the "total executed instruction counter"
00167         stCPU.u64InstructionCount++;
00168     }
00169     else
00170     {
00171         // CPU is asleep, just NOP and wait until we hit an interrupt.
00172         stCPU.u64CycleCount++;
00173         CPU_PeripheralCycle();
00174     }
00175 }
00176
00177 // Check to see if there are any pending interrupts - if so, vector
00178 // to the appropriate location. This has no effect if no interrupts
00179 // are pending
00180 AVR_Interrupt();
00181 }
00182
00183
00184 #if FEATURE_USE_JUMPTABLES
00185 //-----
00186 static void CPU_BuildDecodeTable(void)
00187 {
00188     uint32_t i;
00189     for (i = 0; i < 65536; i++)
00190     {
00191         astDecoders[i] = AVR_Decoder_Function(i);
00192     }
00193 }
00194
00195 //-----
00196 static void CPU_BuildOpcodeTable(void)
00197 {
00198     uint32_t i;
00199     for (i = 0; i < 65536; i++)
00200     {
00201         astOpCodes[i] = AVR_Opcode_Function(i);
00202     }
00203 }
00204
00205 //-----
00206 static void CPU_BuildSizeTable(void)
00207 {
00208     uint32_t i;
00209     for (i = 0; i < 65536; i++)
00210     {
00211         au8OpSizes[i] = AVR_Opcode_Size(i);
00212     }

```

```

00213 }
00214
00215 //-----
00216 static void CPU_BuildCycleTable(void)
00217 {
00218     uint32_t i;
00219     for (i = 0; i < 65536; i++)
00220     {
00221         au8OpCycles[i] = AVR_Opcode_Cycles(i);
00222     }
00223 }
00224 #endif
00225
00226 //-----
00227 void CPU_Init( AVR_CPU_Config_t *pstConfig_ )
00228 {
00229     memset( &stCPU, 0, sizeof(stCPU));
00230     pstConfig_>u32RAMSize += 256;
00231
00232     stCPU.bExitOnReset = pstConfig_>bExitOnReset;
00233
00234     // Dynamically allocate memory for RAM, ROM, and EEPROM buffers
00235     stCPU.pu8EEPROM = (uint8_t*)malloc( pstConfig_>u32EESize );
00236     stCPU.pu16ROM    = (uint16_t*)malloc( pstConfig_>u32ROMSize );
00237     stCPU.pstRAM      = (AVR_RAM_t*)malloc( pstConfig_>u32RAMSize );
00238
00239     stCPU.u32ROMSize = pstConfig_>u32ROMSize;
00240     stCPU.u32RAMSize = pstConfig_>u32RAMSize;
00241     stCPU.u32EEPROMSize = pstConfig_>u32EESize;
00242
00243     memset( stCPU.pu8EEPROM, 0, pstConfig_>u32EESize );
00244     memset( stCPU.pu16ROM, 0, pstConfig_>u32ROMSize );
00245     memset( stCPU.pstRAM, 0, pstConfig_>u32RAMSize );
00246
00247     // Set the base stack pointer to top-of-ram.
00248     uint16_t ul6InitialStack = 256 + pstConfig_>u32RAMSize - 1;
00249     stCPU.pstRAM->stRegisters.SPH.r = (uint8_t)(ul6InitialStack >> 8);
00250     stCPU.pstRAM->stRegisters.SPL.r = (uint8_t)(ul6InitialStack & 0xFF);
00251
00252     // Reset the interrupt priority register
00253     stCPU.u8IntPriority = 255;
00254
00255     // Copy in part-specific interrupt vector and feature tables
00256     stCPU.pstVectorMap = pstConfig_>pstVectorMap;
00257     stCPU.pstFeatureMap = pstConfig_>pstFeatureMap;
00258
00259     #if FEATURE_USE_JUMPTABLES
00260     CPU_BuildCycleTable();
00261     CPU_BuildSizeTable();
00262     CPU_BuildOpcodeTable();
00263     CPU_BuildDecodeTable();
00264     #endif
00265 }
00266
00267 //-----
00268 void CPU_AddPeriph( AVRPeripheral *pstPeriph_ )
00269 {
00270     IO_AddClocker( pstPeriph_ );
00271
00272     uint8_t i;
00273     for (i = pstPeriph_>u8AddrStart; i <= pstPeriph_>u8AddrEnd; i++)
00274     {
00275         IO_AddReader( pstPeriph_, i );
00276         IO_AddWriter( pstPeriph_, i );
00277     }
00278
00279     if (pstPeriph_>pfInit)
00280     {
00281         pstPeriph_>pfInit( pstPeriph_>pvContext );
00282     }
00283 }
00284
00285 //-----
00286 void CPU_RegisterInterruptCallback( InterruptAck pfIntAck_, uint8_t ucVector_
)
00287 {
00288     if (ucVector_ >= 32)
00289     {
00290         return;
00291     }
00292
00293     stCPU.apfInterruptCallbacks[ ucVector_ ] = pfIntAck_;
00294 }

```



## 4.5 src/avr\_cpu/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 "variant.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 array 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 pflntAck\_, 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 CPU\_AddPeriph()

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

CPU\_AddPeriph Add a new I/O Peripheral to the CPU.

#### Parameters

<i>pst↔ Periph_</i>	Pointer to an initialized AVR Peripheral object to be associated with this CPU.
-------------------------	---

Definition at line 268 of file [avr\\_cpu.c](#).

### 4.5.2.2 CPU\_Fetch()

```
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 CPU\_Init()

```
void CPU_Init (
    AVR_CPU_Config_t * pstConfig_ )
```

CPU\_Init Initialize the CPU object and its associated data.

#### Parameters

<i>pst↔ Config_</i>	Pointer to an initialized <a href="#">AVR_CPU_Config_t</a> struct
-------------------------	---

Definition at line 227 of file [avr\\_cpu.c](#).

#### 4.5.2.4 CPU\_RegisterInterruptCallback()

```
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

<i>pfIntAck_</i>	Callback function to register
<i>ucVector_</i>	Interrupt vector index to install handler at

Definition at line 286 of file [avr\\_cpu.c](#).

#### 4.5.2.5 CPU\_RunCycle()

```
void CPU_RunCycle (
    void )
```

CPU\_RunCycle Run a CPU instruction cycle.

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

Definition at line 124 of file [avr\\_cpu.c](#).

## 4.6 avr\_cpu.h

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      )\ ) |
00004 *      ((/( ((/(      \      ( ((/( | -- [ Funkenstein ] -----
00005 *      /( ) /( ) ((( ( )\ )\ /( ) | -- [ Little ] -----
00006 *      ( ) _ ( )      )\ _ )\ ( ( ( ( ) | -- [ AVR ] -----
00007 *      | _ | | _      ( ) _ ( ) \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ / / | _ \ | -- [ Runtime ] -----
00009 *      | _ | | _      / _ \ \ \ / / | _ \ |
00010 *      | _ | | _      / _ \ \ \ / / | _ \ | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00022 #ifndef __AVR_CPU_H__
00023 #define __AVR_CPU_H__
00024
00025 #include <stdint.h>
00026 #include <stdbool.h>
00027
```

```

00028 #include "emu_config.h"
00029
00030 #include "avr_peripheral.h"
00031 #include "avr_periphregs.h"
00032 #include "avr_coreregs.h"
00033 #include "avr_registerfile.h"
00034 #include "avr_io.h"
00035 #include "variant.h"
00036
00037 #include "watchpoint.h"
00038 #include "breakpoint.h"
00039
00040 //-----
00048 typedef struct
00049 {
00050     union
00051     {
00052         AVRRegisterFile stRegisters;
00053         uint8_t au8RAM[ sizeof(AVRRegisterFile) ];
00054     };
00055 } AVR_RAM_t;
00056
00057 //-----
00064 typedef struct
00065 {
00066     //-----
00067     // Jump tables for peripheral read/write functions. This implementation uses
00068     // a table with function pointer arrays, enabling multiple peripherals to
00069     // monitor reads/writes at particular addresses efficiently.
00070     //-----
00071     IOReaderList *apstPeriphReadTable[CONFIG_IO_ADDRESS_BYTES];
00072     IOWriterList *apstPeriphWriteTable[CONFIG_IO_ADDRESS_BYTES];
00073     IOClockList *pstClockList;
00074
00075     //-----
00076     // List of data watchpoints
00077     struct _WatchPoint *pstWatchPoints;
00078
00079     //-----
00080     // List of instruction breakpoints
00081     struct _BreakPoint *pstBreakPoints;
00082
00083     //-----
00084     // Internal CPU Registers (not exposed via IO space)
00085     uint32_t u32PC; // Program counter is not memory mapped, unlike all others
00086
00087     //-----
00088     // Emulator variables
00089     uint64_t u64InstructionCount; // Total Executed instructions
00090     uint64_t u64CycleCount; // Cycle Counter
00091     uint32_t u32CoreFreq; // CPU Frequency (Hz)
00092     uint32_t u32WDTCCount; // Current watchdog timer count
00093     uint16_t u16ExtraPC; // Offset to add to the PC after executing an instruction
00094     uint16_t u16ExtraCycles; // CPU Cycles to add for the current instruction
00095
00096     bool bAsleep; // Whether or not the CPU is sleeping (wake by interrupt)
00097     //-----
00098     // Temporary registers used for optimizing opcodes - for various addressing modes
00099     uint16_t *Rd16;
00100     uint8_t *Rd; // Destination register (in some cases, also source)
00101
00102     uint16_t *Rr16;
00103     uint8_t *Rr; // Source register
00104
00105     uint16_t K; // Constant data
00106     union
00107     {
00108         uint32_t k; // Constant address
00109         int32_t k_s; // Signed, constant address
00110     };
00111
00112     uint8_t A; // IO location address
00113     uint8_t b; // Bit in a register file (3-bits wide)
00114     uint8_t s; // Bit in the status register (3-bits wide)
00115     uint8_t q; // Displacement for direct addressing (6-bits)
00116
00117     //-----
00118     // Setting up regions of memory for general-purpose RAM (shared with the
00119     // IO space from 0-0xFF), ROM/FLASH, and EEPROM.
00120     //-----
00121     uint16_t *pu16ROM;
00122     uint8_t *pu8EEPROM;
00123     AVR_RAM_t *pstRAM;
00124
00125     uint32_t u32ROMSize;
00126     uint32_t u32EEPROMSize;
00127     uint32_t u32RAMSize;

```

```

00128
00129 //-----
00130 uint8_t      u8IntPriority; // Priority of pending interrupts this cycle
00131 uint32_t     u32IntFlags;   // Bitmask for the 32 interrupts
00132
00133 //-----
00134 InterruptAck apfInterruptCallbacks[32]; // Interrupt callbacks
00135
00136 //-----
00137 bool         bExitOnReset; // Flag indicating behavior when we jump to 0. true == exit emulator
00138 bool         bProfile;     // Flag indicating that CPU is running with active code profiling
00139
00140 //-----
00141 const AVR_Vector_Map_t *pstVectorMap; // part-specific interrupt vector map
00142 const AVR_Feature_Map_t *pstFeatureMap; // part-specific feature map
00143 } AVR_CPU;
00144
00145
00146 //-----
00151 typedef struct
00152 {
00153     uint32_t u32ROMSize;
00154     uint32_t u32RAMSize;
00155     uint32_t u32EESize;
00156     bool     bExitOnReset;
00157     const AVR_Vector_Map_t *pstVectorMap; // part-specific interrupt vector map
00158     const AVR_Feature_Map_t *pstFeatureMap; // part-specific feature map
00159 } AVR_CPU_Config_t;
00160
00161
00162 //-----
00168 void CPU_Init( AVR_CPU_Config_t *pstConfig_ );
00169
00170 //-----
00176 uint16_t CPU_Fetch( void );
00177
00178 //-----
00184 void CPU_RunCycle( void );
00185
00186 //-----
00193 void CPU_AddPeriph( AVRPeripheral *pstPeriph_ );
00194
00195 //-----
00206 void CPU_RegisterInterruptCallback( InterruptAck pfIntAck_, uint8_t ucVector_
    );
00207
00208
00209 extern AVR_CPU stCPU;
00210
00211 #endif

```

## 4.7 src/avr\_cpu/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](#) (uint32\_t u32Start\_, 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 [print\\_core\\_regs\(\)](#)

```
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 [print\\_io\\_reg\(\)](#)

```
void print_io_reg (
    uint8_t u8Addr_ )
```

[print\\_io\\_reg](#)

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

#### Parameters

<i>u8↵ Addr_</i>	Address of the IO register to display
----------------------	---------------------------------------

Definition at line 116 of file [avr\\_cpu\\_print.c](#).

#### 4.7.2.3 print\_io\_reg\_with\_name()

```
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

<i>u8Addr_</i>	Address of the IO register to display
<i>szName_</i>	"Friendly name" of the register.

Definition at line 122 of file [avr\\_cpu\\_print.c](#).

#### 4.7.2.4 print\_ram()

```
void print_ram (
    uint16_t u16Start_,
    uint16_t u16Span_ )
```

`print_ram`

Display a block of RAM on the console.

##### Parameters

<i>u16Start_</i>	Start address
<i>u16Span_</i>	Number of bytes to display

Definition at line 128 of file [avr\\_cpu\\_print.c](#).

#### 4.7.2.5 print\_rom()

```
void print_rom (
    uint32_t u32Start_,
    uint16_t u16Span_ )
```





```

00066     PRINT_FUNC("H");
00067 }
00068 else
00069 {
00070     PRINT_FUNC("-");
00071 }
00072 if (1 == stCPU.pstRAM->stRegisters.SREG.S)
00073 {
00074     PRINT_FUNC("S");
00075 }
00076 else
00077 {
00078     PRINT_FUNC("-");
00079 }
00080 if (1 == stCPU.pstRAM->stRegisters.SREG.V)
00081 {
00082     PRINT_FUNC("V");
00083 }
00084 else
00085 {
00086     PRINT_FUNC("-");
00087 }
00088 if (1 == stCPU.pstRAM->stRegisters.SREG.N)
00089 {
00090     PRINT_FUNC("N");
00091 }
00092 else
00093 {
00094     PRINT_FUNC("-");
00095 }
00096 if (1 == stCPU.pstRAM->stRegisters.SREG.Z)
00097 {
00098     PRINT_FUNC("Z");
00099 }
00100 else
00101 {
00102     PRINT_FUNC("-");
00103 }
00104 if (1 == stCPU.pstRAM->stRegisters.SREG.C)
00105 {
00106     PRINT_FUNC("C");
00107 }
00108 else
00109 {
00110     PRINT_FUNC("-");
00111 }
00112 PRINT_FUNC("]\n");
00113 }
00114
00115 //-----
00116 void print_io_reg( uint8_t u8Addr_ )
00117 {
00118     PRINT_FUNC( "[IO%02X]= 0x%02X\n", u8Addr_, stCPU.pstRAM->au8RAM[u8Addr_] );
00119 }
00120
00121 //-----
00122 void print_io_reg_with_name( uint8_t u8Addr_, const char *szName_ )
00123 {
00124     PRINT_FUNC( "[%s]= 0x%02X\n", szName_, stCPU.pstRAM->au8RAM[u8Addr_] );
00125 }
00126
00127 //-----
00128 void print_ram( uint16_t u16Start_, uint16_t u16Span_ )
00129 {
00130     uint16_t i, j;
00131
00132     while (u16Span_)
00133     {
00134         // Print the current memory address
00135         PRINT_FUNC( "[0x%04X]", u16Start_ );
00136         if (u16Span_ < RAM_DISPLAY_SPAN)
00137         {
00138             j = u16Span_;
00139         }
00140         else
00141         {
00142             j = RAM_DISPLAY_SPAN;
00143         }
00144
00145         // Print a divider, followed by the ASCII codes for each char
00146         PRINT_FUNC( "| " );
00147         for (i = 0; i < j; i++)
00148         {
00149             uint8_t u8Char = stCPU.pstRAM->au8RAM[u16Start_ + i];
00150             if (u8Char < 32)
00151             {
00152                 u8Char = '.';

```

```

00153         }
00154
00155         PRINT_FUNC( " %c", u8Char );
00156     }
00157     i = j;
00158     while (i < RAM_DISPLAY_SPAN)
00159     {
00160         PRINT_FUNC( "   ");
00161         i++;
00162     }
00163
00164     // Print a divider, followed by the HEX code for each char
00165     PRINT_FUNC( " |" );
00166     for (i = 0; i < j; i++)
00167     {
00168         PRINT_FUNC( " %02X", stCPU.pstRAM->au8RAM[u16Start_ + i]);
00169     }
00170
00171     if (u16Span_ < RAM_DISPLAY_SPAN)
00172     {
00173         u16Span_ = 0;
00174     }
00175     else
00176     {
00177         u16Span_ -= RAM_DISPLAY_SPAN;
00178     }
00179     u16Start_ += RAM_DISPLAY_SPAN;
00180     PRINT_FUNC( "\n" );
00181 }
00182 }
00183
00184 //-----
00185 void print_rom( uint32_t u32Start_, uint16_t u16Span_ )
00186 {
00187     uint16_t i, j;
00188
00189     while (u16Span_)
00190     {
00191         // Print the current memory address
00192         PRINT_FUNC( "[0x%04X]", u32Start_ );
00193         if (u16Span_ < ROM_DISPLAY_SPAN)
00194         {
00195             j = u16Span_;
00196         }
00197         else
00198         {
00199             j = ROM_DISPLAY_SPAN;
00200         }
00201
00202         // Print a divider, followed by the ASCII codes for each char
00203         PRINT_FUNC( " |" );
00204         for (i = 0; i < j; i++)
00205         {
00206             uint16_t u16Val = stCPU.pu16ROM[u32Start_ + i];
00207             uint8_t u8High = u16Val >> 8;
00208             uint8_t u8Low = u16Val & 0x00FF;
00209
00210             if (u8High < 32)
00211             {
00212                 u8High = '.';
00213             }
00214             if (u8Low < 32)
00215             {
00216                 u8Low = '.';
00217             }
00218
00219             PRINT_FUNC( " %c%c", u8High, u8Low );
00220         }
00221         i = j;
00222         while (i < ROM_DISPLAY_SPAN)
00223         {
00224             PRINT_FUNC( "   ");
00225             i++;
00226         }
00227
00228         // Print a divider, followed by the HEX code for each char
00229         PRINT_FUNC( " |" );
00230         for (i = 0; i < j; i++)
00231         {
00232             PRINT_FUNC( " %04X", stCPU.pu16ROM[u32Start_ + i]);
00233         }
00234
00235         if (u16Span_ < ROM_DISPLAY_SPAN)
00236         {
00237             u16Span_ = 0;
00238         }
00239         else

```

```

00240     {
00241         u16Span_ -= ROM_DISPLAY_SPAN;
00242     }
00243     u32Start_ += ROM_DISPLAY_SPAN;
00244     PRINT_FUNC( "\n" );
00245 }
00246 }

```

## 4.9 src/avr\_cpu/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](#) (uint32\_t u32Start\_, 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 [print\\_core\\_regs\(\)](#)

```

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 [print\\_io\\_reg\(\)](#)

```

void print_io_reg (
    uint8_t u8Addr_ )

```

[print\\_io\\_reg](#)

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

**Parameters**

<i>u8↔ Addr_</i>	Address of the IO register to display
----------------------	---------------------------------------

Definition at line 116 of file [avr\\_cpu\\_print.c](#).

**4.9.2.3 print\_io\_reg\_with\_name()**

```
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**

<i>u8Addr↔ _</i>	Address of the IO register to display
<i>sz↔ Name_</i>	"Friendly name" of the register.

Definition at line 122 of file [avr\\_cpu\\_print.c](#).

**4.9.2.4 print\_ram()**

```
void print_ram (
    uint16_t u16Start_,
    uint16_t u16Span_ )
```

`print_ram`

Display a block of RAM on the console.

**Parameters**

<i>u16↔ Start_</i>	Start address
<i>u16↔ Span_</i>	Number of bytes to display

Definition at line 128 of file [avr\\_cpu\\_print.c](#).

## 4.9.2.5 print\_rom()

```
void print_rom (
    uint32_t u32Start_,
    uint16_t u16Span_ )
```

print\_rom

Display a block of ROM to the console

## Parameters

<i>u32Start_</i>	Start address
<i>u16Span_</i>	Number of instruction words (16-bit) to display

Definition at line 185 of file [avr\\_cpu\\_print.c](#).

## 4.10 avr\_cpu\_print.h

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( ((/(      \      | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( ) \      | -- [ Little ] -----
00006 *      ( ) _ ( ) ) \ _ ) \ ( ( ( ) _ | -- [ AVR ] -----
00007 *      | _ | | _ ( ) _ \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ / _ \ \ v / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ / _ \ \ \ / | _ \ |
00010 *      | _ | | _ / _ \ \ \ / | _ \ |
00011 *      -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00023 #ifndef __AVR_CPU_PRINT_H__
00024 #define __AVR_CPU_PRINT_H__
00025
00026 #include <stdint.h>
00027 #include "avr_cpu.h"
00028
00029 //-----
00035 void print_core_regs( void );
00036
00037 //-----
00045 void print_io_reg( uint8_t u8Addr_ );
00046
00047 //-----
00057 void print_io_reg_with_name( uint8_t u8Addr_, const char *szName_ );
00058
00059 //-----
00068 void print_ram( uint16_t u16Start_, uint16_t u16Span_ );
00069
00070 //-----
00079 void print_rom( uint32_t u32Start_, uint16_t u16Span_ );
00080
00081 #endif
```

## 4.11 src/avr\_cpu/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

- static int8\_t **Signed\_From\_Unsigned\_6** (uint8\_t u8Signed\_)
- static uint8\_t **Register\_From\_Rd** (void)
- static uint8\_t **Register\_From\_Rr** (void)
- static uint8\_t **Register\_From\_Rd16** (void)
- static 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\_SBRB** (char \*szOutput\_)
- static void **AVR\_Disasm\_SBIC** (char \*szOutput\_)
- static void **AVR\_Disasm\_SBI** (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\_MOVB** (char \*szOutput\_)
- static void **AVR\_Disasm\_MOVBW** (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\_Function()

```
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

$O \leftarrow$	Opcode to disassemble
$P \leftarrow$	
—	

#### 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

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( ()/(      \      ( ( (/(      | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( ) \ ) \ / ( )      | -- [ Little ] -----
00006 *      ( ) _ ( )      ) \ _ \ ( ( ( ( _ ( )      | -- [ AVR ] -----
00007 *      | | _ | |      ( ) _ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _ / _ \ \ V / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _ / / \ \ \ / | _ \      |
00010 *      |                                     | "Yeah, it does Arduino..."
00011 *      |-----|
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #include <stdint.h>
00022 #include <stdio.h>
00023
00024 #include "emu_config.h"
00025
00026 #include "avr_disasm.h"
00027 #include "avr_op_decode.h"
00028 #include "avr_opcodes.h"
```

```

00029 #include "avr_op_size.h"
00030 #include "avr_cpu.h"
00031 #include "avr_cpu_print.h"
00032 #include "avr_loader.h"
00033
00034 //-----
00035 static int8_t Signed_From_Unsigned_6( uint8_t u8Signed_ )
00036 {
00037     int8_t i8Ret = 0;
00038     if( u8Signed_ & 0x20 )
00039     {
00040         //Sign extend...
00041         i8Ret = (int8_t)(u8Signed_ | 0xC0);
00042     }
00043     else
00044     {
00045         i8Ret = (int8_t)u8Signed_;
00046     }
00047     return i8Ret;
00048 }
00049
00050 //-----
00051 static uint8_t Register_From_Rd( void )
00052 {
00053     return stCPU.Rd - &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0);
00054 }
00055 //-----
00056 static uint8_t Register_From_Rr( void )
00057 {
00058     return stCPU.Rr - &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0);
00059 }
00060
00061 //-----
00062 static uint8_t Register_From_Rd16( void )
00063 {
00064     return (uint8_t*)(stCPU.Rd16) - &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0);
00065 }
00066
00067 //-----
00068 static uint8_t Register_From_Rr16( void )
00069 {
00070     return (uint8_t*)(stCPU.Rr16) - &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0);
00071 }
00072
00073 //-----
00074 static void AVR_Disasm_ADD( char *szOutput_ )
00075 {
00076     uint8_t u8Rd = Register_From_Rd();
00077     uint8_t u8Rr = Register_From_Rr();
00078
00079     //ruler: 0----5----10---15---20---25---30---35---40" );
00080     sprintf( szOutput_, "add r%d, r%d\t ; Add: r%d = r%d + r%d\n",
00081             u8Rd, u8Rr,
00082             u8Rd, u8Rd, u8Rr );
00083 }
00084
00085 //-----
00086 static void AVR_Disasm_ADC( char *szOutput_ )
00087 {
00088     uint8_t u8Rd = Register_From_Rd();
00089     uint8_t u8Rr = Register_From_Rr();
00090
00091     //ruler: 0----5----10---15---20---25---30---35---40" );
00092     sprintf( szOutput_, "adc r%d, r%d\t ; Add with carry: r%d = r%d + r%d + C\n",
00093             u8Rd, u8Rr,
00094             u8Rd, u8Rd, u8Rr );
00095 }
00096
00097 //-----
00098 static void AVR_Disasm_ADIW( char *szOutput_ )
00099 {
00100     uint8_t u8Rd = Register_From_Rd16();
00101     uint8_t u8K = stCPU.K;
00102
00103     //ruler: 0----5----10---15---20---25---30---35---40" );
00104     sprintf( szOutput_, "adiw r%d:%d, %d\t ; Add immediate to word: r%d:%d = r%d:%d + %d\n",
00105             u8Rd + 1, u8Rd, u8K,
00106             u8Rd + 1, u8Rd, u8Rd + 1, u8Rd, u8K
00107             );
00108 }
00109
00110 //-----
00111 static void AVR_Disasm_SUB( char *szOutput_ )
00112 {
00113     uint8_t u8Rd = Register_From_Rd();
00114     uint8_t u8Rr = Register_From_Rr();

```

```

00116
00117 //ruler: 0----5----10---15---20---25---30---35---40" );
00118 sprintf( szOutput_, "sub r%d, r%d \t ; Subtract: r%d = r%d - r%d \n",
00119         u8Rd, u8Rr,
00120         u8Rd, u8Rd, u8Rr
00121     );
00122 }
00123
00124 //-----
00125 static void AVR_Disasm_SUBI( char *szOutput_ )
00126 {
00127     uint8_t u8Rd = Register_From_Rd();
00128     uint8_t u8K = stCPU.K;
00129
00130     //ruler: 0----5----10---15---20---25---30---35---40" );
00131     sprintf( szOutput_, "subi r%d, %d \t ; Subtract immediate: r%d = r%d - %d \n",
00132             u8Rd, u8K,
00133             u8Rd, u8Rd, u8K
00134         );
00135 }
00136
00137 //-----
00138 static void AVR_Disasm_SBC( char *szOutput_ )
00139 {
00140     uint8_t u8Rd = Register_From_Rd();
00141     uint8_t u8Rr = Register_From_Rr();
00142
00143     //ruler: 0----5----10---15---20---25---30---35---40" );
00144     sprintf( szOutput_, "sbc r%d, r%d \t ; Subtract with carry: r%d = r%d - r%d - C \n",
00145             u8Rd, u8Rr,
00146             u8Rd, u8Rd, u8Rr
00147         );
00148 }
00149
00150 //-----
00151 static void AVR_Disasm_SBCI( char *szOutput_ )
00152 {
00153     uint8_t u8Rd = Register_From_Rd();
00154     uint8_t u8K = stCPU.K;
00155
00156     //ruler: 0----5----10---15---20---25---30---35---40" );
00157     sprintf( szOutput_, "sbci r%d, %d \t ; Subtract immediate with carry: r%d = r%d - %d - C\n",
00158             u8Rd, u8K,
00159             u8Rd, u8Rd, u8K
00160         );
00161 }
00162
00163 //-----
00164 static void AVR_Disasm_SBIW( char *szOutput_ )
00165 {
00166     uint8_t u8Rd = Register_From_Rd16();
00167     uint8_t u8K = stCPU.K;
00168
00169     //ruler: 0----5----10---15---20---25---30---35---40" );
00170     sprintf( szOutput_, "sbw r%d:%d, %d \t ; Subtract immediate from word: r%d:%d = r%d:%d + %d\n",
00171             u8Rd + 1, u8Rd, u8K,
00172             u8Rd + 1, u8Rd, u8Rd + 1, u8Rd, u8K
00173         );
00174 }
00175
00176 //-----
00177 static void AVR_Disasm_AND( char *szOutput_ )
00178 {
00179     uint8_t u8Rd = Register_From_Rd();
00180     uint8_t u8Rr = Register_From_Rr();
00181
00182     //ruler: 0----5----10---15---20---25---30---35---40" );
00183     sprintf( szOutput_, "and r%d, r%d \t ; Logical AND: r%d = r%d & r%d \n",
00184             u8Rd, u8Rr,
00185             u8Rd, u8Rd, u8Rr
00186         );
00187 }
00188
00189 //-----
00190 static void AVR_Disasm_ANDI( char *szOutput_ )
00191 {
00192     uint8_t u8Rd = Register_From_Rd();
00193     uint8_t u8K = stCPU.K;
00194
00195     //ruler: 0----5----10---15---20---25---30---35---40" );
00196     sprintf( szOutput_, "andi r%d, %d \t ; Logical AND with Immediate: r%d = r%d & %d\n",
00197             u8Rd, u8K,
00198             u8Rd, u8Rd, u8K
00199         );
00200 }

```

```

00201
00202 //-----
00203 static void AVR_Disasm_OR( char *szOutput_ )
00204 {
00205     uint8_t u8Rd = Register_From_Rd();
00206     uint8_t u8Rr = Register_From_Rr();
00207
00208     //ruler: 0----5----10---15---20---25---30---35---40" );
00209     sprintf( szOutput_, "or r%d, r%d          \t ; Logical OR: r%d = r%d | r%d \n",
00210             u8Rd, u8Rr,
00211             u8Rd, u8Rd, u8Rr
00212             );
00213 }
00214
00215 //-----
00216 static void AVR_Disasm_ORI( char *szOutput_ )
00217 {
00218     uint8_t u8Rd = Register_From_Rd();
00219     uint8_t u8K = stCPU.K;
00220
00221     //ruler: 0----5----10---15---20---25---30---35---40" );
00222     sprintf( szOutput_, "ori r%d, %d          \t ; Logical OR with Immediate: r%d = r%d | %d\n",
00223             u8Rd, u8K,
00224             u8Rd, u8Rd, u8K
00225             );
00226 }
00227
00228 //-----
00229 static void AVR_Disasm_EOR( char *szOutput_ )
00230 {
00231     uint8_t u8Rd = Register_From_Rd();
00232     uint8_t u8Rr = Register_From_Rr();
00233
00234     //ruler: 0----5----10---15---20---25---30---35---40" );
00235     sprintf( szOutput_, "eor r%d, r%d          \t ; Exclusive OR: r%d = r%d ^ r%d \n",
00236             u8Rd, u8Rr,
00237             u8Rd, u8Rd, u8Rr
00238             );
00239 }
00240
00241 //-----
00242 static void AVR_Disasm_COM( char *szOutput_ )
00243 {
00244     uint8_t u8Rd = Register_From_Rd();
00245
00246     //ruler: 0----5----10---15---20---25---30---35---40" );
00247     sprintf( szOutput_, "com r%d          \t ; One's complement (bitwise inverse): r%d = 0xFF -
00248 r%d\n",
00249             u8Rd,
00250             u8Rd, u8Rd
00251             );
00252 }
00253 //-----
00254 static void AVR_Disasm_NEG( char *szOutput_ )
00255 {
00256     uint8_t u8Rd = Register_From_Rd();
00257
00258     //ruler: 0----5----10---15---20---25---30---35---40" );
00259     sprintf( szOutput_, "neg r%d          \t ; Two's complement (sign swap): r%d = 0x00 - r%d\n",
00260             u8Rd,
00261             u8Rd, u8Rd
00262             );
00263 }
00264
00265 //-----
00266 static void AVR_Disasm_SBR( char *szOutput_ )
00267 {
00268     uint8_t u8Rd = Register_From_Rd();
00269     uint8_t u8K = stCPU.K;
00270
00271     //ruler: 0----5----10---15---20---25---30---35---40" );
00272     sprintf( szOutput_, "sbr r%d, %d          \t ; Set Bits in Register: r%d = r%d | %d\n",
00273             u8Rd, u8K,
00274             u8Rd, u8Rd, u8K
00275             );
00276 }
00277
00278 //-----
00279 static void AVR_Disasm_CBR( char *szOutput_ )
00280 {
00281     uint8_t u8Rd = Register_From_Rd();
00282     uint8_t u8K = stCPU.K;
00283
00284     //ruler: 0----5----10---15---20---25---30---35---40" );
00285     sprintf( szOutput_, "cbr r%d, %d          \t ; Clear Bits in Register: r%d = r%d & (0xFF - %d)\n",
00286             u8Rd, u8K,

```

```

00287         u8Rd, u8Rd, u8K
00288     );
00289 }
00290
00291 //-----
00292 static void AVR_Disasm_INC( char *szOutput_ )
00293 {
00294     uint8_t u8Rd = Register_From_Rd();
00295
00296     //ruler: 0---5---10---15---20---25---30---35---40" );
00297     sprintf( szOutput_, "inc r%d          \t ; Increment Register: r%d = r%d + 1\n",
00298             u8Rd,
00299             u8Rd, u8Rd
00300     );
00301 }
00302
00303 //-----
00304 static void AVR_Disasm_DEC( char *szOutput_ )
00305 {
00306     uint8_t u8Rd = Register_From_Rd();
00307
00308     //ruler: 0---5---10---15---20---25---30---35---40" );
00309     sprintf( szOutput_, "dec r%d          \t ; Decrement Register: r%d = r%d - 1\n",
00310             u8Rd,
00311             u8Rd, u8Rd
00312     );
00313 }
00314
00315 //-----
00316 static void AVR_Disasm_TST( char *szOutput_ )
00317 {
00318     uint8_t u8Rd = Register_From_Rd();
00319
00320     //ruler: 0---5---10---15---20---25---30---35---40" );
00321     sprintf( szOutput_, "tst r%d          \t ; Test Register for Zero or Negative\n",
00322             u8Rd
00323     );
00324 }
00325
00326 //-----
00327 static void AVR_Disasm_CLR( char *szOutput_ )
00328 {
00329     uint8_t u8Rd = Register_From_Rd();
00330
00331     //ruler: 0---5---10---15---20---25---30---35---40" );
00332     sprintf( szOutput_, "clr r%d          \t ; Clear Register\n",
00333             u8Rd
00334     );
00335 }
00336
00337 //-----
00338 static void AVR_Disasm_SER( char *szOutput_ )
00339 {
00340     uint8_t u8Rd = Register_From_Rd();
00341
00342     //ruler: 0---5---10---15---20---25---30---35---40" );
00343     sprintf( szOutput_, "ser r%d          \t ; Set All Bits in Register\n",
00344             u8Rd
00345     );
00346 }
00347
00348 //-----
00349 static void AVR_Disasm_MUL( char *szOutput_ )
00350 {
00351     uint8_t u8Rd = Register_From_Rd();
00352     uint8_t u8Rr = Register_From_Rr();
00353
00354     //ruler: 0---5---10---15---20---25---30---35---40" );
00355     sprintf( szOutput_, "mul r%d, r%d          \t ; Unsigned Multiply: r1:0 = r%d * r%d\n",
00356             u8Rd, u8Rr,
00357             u8Rd, u8Rr );
00358 }
00359
00360 //-----
00361 static void AVR_Disasm_MULS( char *szOutput_ )
00362 {
00363     uint8_t u8Rd = Register_From_Rd();
00364     uint8_t u8Rr = Register_From_Rr();
00365
00366     //ruler: 0---5---10---15---20---25---30---35---40" );
00367     sprintf( szOutput_, "muls r%d, r%d          \t ; Signed Multiply: r1:0 = r%d * r%d\n",
00368             u8Rd, u8Rr,
00369             u8Rd, u8Rr );
00370 }
00371
00372 //-----
00373 static void AVR_Disasm_MULSU( char *szOutput_ )

```

```

00374 {
00375     uint8_t u8Rd = Register_From_Rd();
00376     uint8_t u8Rr = Register_From_Rr();
00377
00378     //ruler: 0----5----10----15----20----25----30----35----40" );
00379     sprintf( szOutput_, "mulsu r%d, r%d          \t ; Signed * Unsigned Multiply: r1:0 = r%d * r%d\n",
00380             u8Rd, u8Rr,
00381             u8Rd, u8Rr );
00382 }
00383
00384 //-----
00385 static void AVR_Disasm_FMUL( char *szOutput_ )
00386 {
00387     uint8_t u8Rd = Register_From_Rd();
00388     uint8_t u8Rr = Register_From_Rr();
00389
00390     //ruler: 0----5----10----15----20----25----30----35----40" );
00391     sprintf( szOutput_, "fmul r%d, r%d          \t ; Fractional Multiply: r1:0 = r%d * r%d\n",
00392             u8Rd, u8Rr,
00393             u8Rd, u8Rr );
00394 }
00395
00396 //-----
00397 static void AVR_Disasm_FMULS( char *szOutput_ )
00398 {
00399     uint8_t u8Rd = Register_From_Rd();
00400     uint8_t u8Rr = Register_From_Rr();
00401
00402     //ruler: 0----5----10----15----20----25----30----35----40" );
00403     sprintf( szOutput_, "fmuls r%d, r%d          \t ; Signed Fractional Multiply: r1:0 = r%d * r%d\n",
00404             u8Rd, u8Rr,
00405             u8Rd, u8Rr );
00406 }
00407
00408 //-----
00409 static void AVR_Disasm_FMULSU( char *szOutput_ )
00410 {
00411     uint8_t u8Rd = Register_From_Rd();
00412     uint8_t u8Rr = Register_From_Rr();
00413
00414     //ruler: 0----5----10----15----20----25----30----35----40" );
00415     sprintf( szOutput_, "fmulsu r%d, r%d          \t ; Signed * Unsigned Fractional Multiply: r1:0 = r%d *
00416             r%d\n",
00417             u8Rd, u8Rr,
00418             u8Rd, u8Rr );
00419 }
00420
00421 //-----
00422 static void AVR_Disasm_DES( char *szOutput_ )
00423 {
00424     uint8_t u8K = stCPU.K;
00425
00426     //ruler: 0----5----10----15----20----25----30----35----40" );
00427     sprintf( szOutput_, "des %d          \t ; DES Encrypt/Decrypt\n",
00428             u8K );
00429 }
00430
00431 //-----
00432 static void AVR_Disasm_RJMP( char *szOutput_ )
00433 {
00434     int16_t i16k = stCPU.k_s;
00435
00436     //ruler: 0----5----10----15----20----25----30----35----40" );
00437     sprintf( szOutput_, "rjmp %d          \t ; Relative Jump: PC = PC + %d + 1\n",
00438             i16k, i16k );
00439 }
00440
00441 //-----
00442 static void AVR_Disasm_IJMP( char *szOutput_ )
00443 {
00444     //ruler: 0----5----10----15----20----25----30----35----40" );
00445     sprintf( szOutput_, "ijmp          \t ; Indirect Jump: PC = Z\n");
00446 }
00447
00448 //-----
00449 static void AVR_Disasm_EIJMP( char *szOutput_ )
00450 {
00451     //ruler: 0----5----10----15----20----25----30----35----40" );
00452     sprintf( szOutput_, "eijmp          \t ; Extended Indirect Jump: PC(15:0) = Z(15:0),
00453             PC(21:16) = EIND\n" );
00454 }
00455 //-----
00456 static void AVR_Disasm_JMP( char *szOutput_ )
00457 {
00458     uint32_t u32k = stCPU.k;

```

```

00459
00460 //ruler: 0----5----10----15----20----25----30----35----40" );
00461 sprintf( szOutput_, "jmp 0x%X \t ; Jump to 0x%X \n",
00462          u32k, u32k );
00463 }
00464
00465 //-----
00466 static void AVR_Disasm_RCALL( char *szOutput_ )
00467 {
00468     int16_t i16k = stCPU.k_s;
00469
00470     //ruler: 0----5----10----15----20----25----30----35----40" );
00471     sprintf( szOutput_, "rcall %d \t ; Relative call to Subroutine: PC = PC + %d + 1\n",
00472             i16k, i16k );
00473 }
00474 }
00475
00476 //-----
00477 static void AVR_Disasm_ICALL( char *szOutput_ )
00478 {
00479     //ruler: 0----5----10----15----20----25----30----35----40" );
00480     sprintf( szOutput_, "icall \t ; Indirect Jump: PC = Z\n");
00481 }
00482
00483 //-----
00484 static void AVR_Disasm_EICALL( char *szOutput_ )
00485 {
00486     //ruler: 0----5----10----15----20----25----30----35----40" );
00487     sprintf( szOutput_, "eicall \t ; Extended Indirect Jump: PC(15:0) = Z(15:0),
00488             PC(21:16) = EIND\n" );
00489 }
00490 //-----
00491 static void AVR_Disasm_CALL( char *szOutput_ )
00492 {
00493     uint32_t u32k = stCPU.k;
00494
00495     //ruler: 0----5----10----15----20----25----30----35----40" );
00496     sprintf( szOutput_, "call 0x%X \t ; Long Call to Subroutine: PC = 0x%X \n",
00497             u32k, u32k );
00498 }
00499 }
00500 //-----
00501 static void AVR_Disasm_RET( char *szOutput_ )
00502 {
00503     //ruler: 0----5----10----15----20----25----30----35----40" );
00504     sprintf( szOutput_, "ret \t ; Return from subroutine\n" );
00505 }
00506
00507 //-----
00508 static void AVR_Disasm_RETI( char *szOutput_ )
00509 {
00510     //ruler: 0----5----10----15----20----25----30----35----40" );
00511     sprintf( szOutput_, "reti \t ; Return from interrupt\n" );
00512 }
00513
00514 //-----
00515 static void AVR_Disasm_CPSE( char *szOutput_ )
00516 {
00517     uint8_t u8Rd = Register_From_Rd();
00518     uint8_t u8Rr = Register_From_Rr();
00519
00520     //ruler: 0----5----10----15----20----25----30----35----40" );
00521     sprintf( szOutput_, "cpse r%d, r%d \t ; Compare, Skip Next If r%d = r%d\n",
00522             u8Rd, u8Rr,
00523             u8Rd, u8Rr );
00524 }
00525 }
00526
00527 //-----
00528 static void AVR_Disasm_CP( char *szOutput_ )
00529 {
00530     uint8_t u8Rd = Register_From_Rd();
00531     uint8_t u8Rr = Register_From_Rr();
00532
00533     //ruler: 0----5----10----15----20----25----30----35----40" );
00534     sprintf( szOutput_, "cp r%d, r%d \t ; Compare: r%d == r%d\n",
00535             u8Rd, u8Rr,
00536             u8Rd, u8Rr );
00537 }
00538 }
00539
00540 //-----
00541 static void AVR_Disasm_CPC( char *szOutput_ )
00542 {
00543     uint8_t u8Rd = Register_From_Rd();
00544     uint8_t u8Rr = Register_From_Rr();

```

```

00545
00546 //ruler: 0----5----10----15----20----25----30----35----40" );
00547 sprintf( szOutput_, "cpc r%d, r%d          \t ; Compare with carry: r%d == r%d + C\n",
00548          u8Rd, u8Rr,
00549          u8Rd, u8Rr
00550          );
00551 }
00552
00553 //-----
00554 static void AVR_Disasm_CPI( char *szOutput_ )
00555 {
00556     uint8_t u8Rd = Register_From_Rd();
00557     uint8_t u8K = stCPU.K;
00558
00559     //ruler: 0----5----10----15----20----25----30----35----40" );
00560     sprintf( szOutput_, "cpi r%d, %d          \t ; Compare with Immediate: r%d == %d\n",
00561             u8Rd, u8K,
00562             u8Rd, u8K
00563             );
00564 }
00565
00566 //-----
00567 static void AVR_Disasm_SBRc( char *szOutput_ )
00568 {
00569     uint8_t u8Rd = Register_From_Rd();
00570     uint8_t u8b = stCPU.b;
00571
00572     //ruler: 0----5----10----15----20----25----30----35----40" );
00573     sprintf( szOutput_, "sbrc r%d, %d          \t ; Skip if Bit (%d) in Register (r%d) Cleared \n",
00574             u8Rd, u8b,
00575             u8Rd, u8b
00576             );
00577 }
00578
00579 //-----
00580 static void AVR_Disasm_SBRs( char *szOutput_ )
00581 {
00582     uint8_t u8Rd = Register_From_Rd();
00583     uint8_t u8b = stCPU.b;
00584
00585     //ruler: 0----5----10----15----20----25----30----35----40" );
00586     sprintf( szOutput_, "sbrs r%d, %d          \t ; Skip if Bit (%d) in Register (r%d) Set \n",
00587             u8Rd, u8b,
00588             u8Rd, u8b
00589             );
00590 }
00591 }
00592
00593 //-----
00594 static void AVR_Disasm_SBIc( char *szOutput_ )
00595 {
00596     uint8_t u8A = stCPU.A;
00597     uint8_t u8b = stCPU.b;
00598
00599     //ruler: 0----5----10----15----20----25----30----35----40" );
00600     sprintf( szOutput_, "sbic %d, %d          \t ; Skip if Bit (%d) in IO Register (r%d) Cleared \n",
00601             u8A, u8b,
00602             u8A, u8b
00603             );
00604 }
00605
00606 //-----
00607 static void AVR_Disasm_SBIs( char *szOutput_ )
00608 {
00609     uint8_t u8A = stCPU.A;
00610     uint8_t u8b = stCPU.b;
00611
00612     //ruler: 0----5----10----15----20----25----30----35----40" );
00613     sprintf( szOutput_, "sbis %d, %d          \t ; Skip if Bit (%d) in IO Register (r%d) Set \n",
00614             u8A, u8b,
00615             u8A, u8b
00616             );
00617 }
00618
00619 //-----
00620 static void AVR_Disasm_BRBS( char *szOutput_ )
00621 {
00622     uint8_t u8s = stCPU.s;
00623     int8_t s8k = stCPU.k_s;
00624
00625     //ruler: 0----5----10----15----20----25----30----35----40" );
00626     sprintf( szOutput_, "brbs %d, %d          \t ; Branch if Bit (%d) in SR set: PC = PC + %d + 1\n",
00627             u8s, s8k,
00628             u8s, s8k
00629             );
00630 }
00631

```



```

00632 //-----
00633 static void AVR_Disasm_BRBC( char *szOutput_ )
00634 {
00635     uint8_t u8s = stCPU.s;
00636     int8_t s8k = stCPU.k_s;
00637
00638     //ruler: 0---5---10---15---20---25---30---35---40" );
00639     sprintf( szOutput_, "brbc %d, %d          \t ; Branch if Bit (%d) in SR clear: PC = PC + %d + 1\n",
00640
00641             u8s, s8k,
00642             u8s, s8k
00643         );
00644 }
00645 //-----
00646 static void AVR_Disasm_BREQ( char *szOutput_ )
00647 {
00648     int8_t s8k = stCPU.k_s;
00649
00650     //ruler: 0---5---10---15---20---25---30---35---40" );
00651     sprintf( szOutput_, "breq %d          \t ; Branch if zero flag set: PC = PC + %d + 1\n",
00652             s8k,
00653             s8k
00654         );
00655 }
00656 //-----
00657 static void AVR_Disasm_BRNE( char *szOutput_ )
00658 {
00659     int8_t s8k = stCPU.k_s;
00660
00661     //ruler: 0---5---10---15---20---25---30---35---40" );
00662     sprintf( szOutput_, "brne %d          \t ; Branch if zero flag clear: PC = PC + %d + 1\n",
00663             s8k,
00664             s8k
00665         );
00666 }
00667 //-----
00668 static void AVR_Disasm_BRCS( char *szOutput_ )
00669 {
00670     int8_t s8k = stCPU.k_s;
00671
00672     //ruler: 0---5---10---15---20---25---30---35---40" );
00673     sprintf( szOutput_, "brcs %d          \t ; Branch if carry flag set: PC = PC + %d + 1\n",
00674             s8k,
00675             s8k
00676         );
00677 }
00678 //-----
00679 static void AVR_Disasm_BRCC( char *szOutput_ )
00680 {
00681     int8_t s8k = stCPU.k_s;
00682
00683     //ruler: 0---5---10---15---20---25---30---35---40" );
00684     sprintf( szOutput_, "brcc %d          \t ; Branch if carry flag clear: PC = PC + %d + 1\n",
00685             s8k,
00686             s8k
00687         );
00688 }
00689 //-----
00690 static void AVR_Disasm_BRSH( char *szOutput_ )
00691 {
00692     int8_t s8k = stCPU.k_s;
00693
00694     //ruler: 0---5---10---15---20---25---30---35---40" );
00695     sprintf( szOutput_, "brsh %d          \t ; Branch if same or higher: PC = PC + %d + 1\n",
00696             s8k,
00697             s8k
00698         );
00699 }
00700 //-----
00701 static void AVR_Disasm_BRLO( char *szOutput_ )
00702 {
00703     int8_t s8k = stCPU.k_s;
00704
00705     //ruler: 0---5---10---15---20---25---30---35---40" );
00706     sprintf( szOutput_, "brlo %d          \t ; Branch if lower: PC = PC + %d + 1\n",
00707             s8k,
00708             s8k
00709         );
00710 }
00711
00712

```

```

00718 //-----
00719 static void AVR_Disasm_BRMI( char *szOutput_ )
00720 {
00721     int8_t  s8k = stCPU.k_s;
00722
00723     //ruler: 0----5----10----15----20----25----30----35----40" );
00724     sprintf( szOutput_, "brmi %d          \t ; Branch if minus: PC = PC + %d + 1\n",
00725             s8k,
00726             s8k
00727         );
00728 }
00729
00730 //-----
00731 static void AVR_Disasm_BRPL( char *szOutput_ )
00732 {
00733     int8_t  s8k = stCPU.k_s;
00734
00735     //ruler: 0----5----10----15----20----25----30----35----40" );
00736     sprintf( szOutput_, "brpl %d          \t ; Branch if plus: PC = PC + %d + 1\n",
00737             s8k,
00738             s8k
00739         );
00740 }
00741
00742 //-----
00743 static void AVR_Disasm_BRGE( char *szOutput_ )
00744 {
00745     int8_t  s8k = stCPU.k_s;
00746
00747     //ruler: 0----5----10----15----20----25----30----35----40" );
00748     sprintf( szOutput_, "brge %d          \t ; Branch if greater-or-equal (signed): PC = PC + %d +
00749 1\n",
00750             s8k,
00751             s8k
00752         );
00753 }
00754 //-----
00755 static void AVR_Disasm_BRLT( char *szOutput_ )
00756 {
00757     int8_t  s8k = stCPU.k_s;
00758
00759     //ruler: 0----5----10----15----20----25----30----35----40" );
00760     sprintf( szOutput_, "brlt %d          \t ; Branch if less-than (signed): PC = PC + %d + 1\n",
00761             s8k,
00762             s8k
00763         );
00764 }
00765
00766 //-----
00767 static void AVR_Disasm_BRHS( char *szOutput_ )
00768 {
00769     int8_t  s8k = stCPU.k_s;
00770
00771     //ruler: 0----5----10----15----20----25----30----35----40" );
00772     sprintf( szOutput_, "brlt %d          \t ; Branch if half-carry set: PC = PC + %d + 1\n",
00773             s8k,
00774             s8k
00775         );
00776 }
00777
00778 //-----
00779 static void AVR_Disasm_BRHC( char *szOutput_ )
00780 {
00781     int8_t  s8k = stCPU.k_s;
00782
00783     //ruler: 0----5----10----15----20----25----30----35----40" );
00784     sprintf( szOutput_, "brhc %d          \t ; Branch if half-carry clear: PC = PC + %d + 1\n",
00785             s8k,
00786             s8k
00787         );
00788 }
00789
00790 //-----
00791 static void AVR_Disasm_BRTS( char *szOutput_ )
00792 {
00793     int8_t  s8k = stCPU.k_s;
00794
00795     //ruler: 0----5----10----15----20----25----30----35----40" );
00796     sprintf( szOutput_, "brts %d          \t ; Branch if T-flag set: PC = PC + %d + 1\n",
00797             s8k,
00798             s8k
00799         );
00800 }
00801
00802 //-----
00803

```

```

00804 static void AVR_Disasm_BRTC( char *szOutput_ )
00805 {
00806     int8_t  s8k = stCPU.k_s;
00807
00808     //ruler: 0----5----10---15---20---25---30---35---40" );
00809     sprintf( szOutput_, "brtc %d          \t ; Branch if T-flag clear: PC = PC + %d + 1\n",
00810             s8k,
00811             s8k
00812             );
00813 }
00814
00815 //-----
00816 static void AVR_Disasm_BRVS( char *szOutput_ )
00817 {
00818     int8_t  s8k = stCPU.k_s;
00819
00820     //ruler: 0----5----10---15---20---25---30---35---40" );
00821     sprintf( szOutput_, "brvs %d          \t ; Branch if Overflow set: PC = PC + %d + 1\n",
00822             s8k,
00823             s8k
00824             );
00825 }
00826
00827 //-----
00828 static void AVR_Disasm_BRVC( char *szOutput_ )
00829 {
00830     int8_t  s8k = stCPU.k_s;
00831
00832     //ruler: 0----5----10---15---20---25---30---35---40" );
00833     sprintf( szOutput_, "brvc %d          \t ; Branch if Overflow clear: PC = PC + %d + 1\n",
00834             s8k,
00835             s8k
00836             );
00837 }
00838
00839 //-----
00840 static void AVR_Disasm_BRIE( char *szOutput_ )
00841 {
00842     int8_t  s8k = stCPU.k_s;
00843
00844     //ruler: 0----5----10---15---20---25---30---35---40" );
00845     sprintf( szOutput_, "brie %d          \t ; Branch if Interrupt Enabled: PC = PC + %d + 1\n",
00846             s8k,
00847             s8k
00848             );
00849 }
00850
00851 //-----
00852 static void AVR_Disasm_BRID( char *szOutput_ )
00853 {
00854     int8_t  s8k = stCPU.k_s;
00855
00856     //ruler: 0----5----10---15---20---25---30---35---40" );
00857     sprintf( szOutput_, "brid %d          \t ; Branch if Interrupt Disabled: PC = PC + %d + 1\n",
00858             s8k,
00859             s8k
00860             );
00861 }
00862
00863
00864 //-----
00865 static void AVR_Disasm_MOV( char *szOutput_ )
00866 {
00867     uint8_t u8Rd = Register_From_Rd();
00868     uint8_t u8Rr = Register_From_Rr();
00869
00870     //ruler: 0----5----10---15---20---25---30---35---40" );
00871     sprintf( szOutput_, "mov r%d, r%d          \t ; Copy Register: r%d = r%d\n",
00872             u8Rd, u8Rr,
00873             u8Rd, u8Rr
00874             );
00875 }
00876
00877 //-----
00878 static void AVR_Disasm_MOVW( char *szOutput_ )
00879 {
00880     uint16_t u16Rd = Register_From_Rd16();
00881     uint16_t u16Rr = Register_From_Rr16();
00882
00883     //ruler: 0----5----10---15---20---25---30---35---40" );
00884     sprintf( szOutput_, "movw r%d:r%d, r%d:r%d          \t ; Copy Register (Word): r%d:r%d = r%d:r%d\n",
00885             u16Rd+1, u16Rd, u16Rr+1, u16Rr,
00886             u16Rd+1, u16Rd, u16Rr+1, u16Rr
00887             );
00888 }
00889
00890 //-----

```

```

00891 static void AVR_Disasm_LDI( char *szOutput_ )
00892 {
00893     uint8_t u8Rd = Register_From_Rd();
00894     uint8_t u8K = stCPU.K;
00895
00896     //ruler: 0---5---10---15---20---25---30---35---40" );
00897     sprintf( szOutput_, "ldi r%d, %d          \t ; Load Immediate: r%d = %d\n",
00898             u8Rd, u8K,
00899             u8Rd, u8K
00900             );
00901 }
00902
00903 //-----
00904 static void AVR_Disasm_LDS( char *szOutput_ )
00905 {
00906     uint8_t u8Rd = Register_From_Rd();
00907     uint16_t u16k = stCPU.k;
00908
00909     //ruler: 0---5---10---15---20---25---30---35---40" );
00910     sprintf( szOutput_, "lds r%d, %d          \t ; Load Direct from Data Space: r%d = (%d)\n",
00911             u8Rd, u16k,
00912             u8Rd, u16k
00913             );
00914 }
00915
00916 //-----
00917 static void AVR_Disasm_LD_X_Indirect( char *szOutput_ )
00918 {
00919     uint8_t u8Rd = Register_From_Rd();
00920
00921     //ruler: 0---5---10---15---20---25---30---35---40" );
00922     sprintf( szOutput_, "ld r%d, X          \t ; Load Indirect from Data Space\n",
00923             u8Rd
00924             );
00925 }
00926
00927 //-----
00928 static void AVR_Disasm_LD_X_Indirect_Postinc( char *szOutput_ )
00929 {
00930     uint8_t u8Rd = Register_From_Rd();
00931
00932     //ruler: 0---5---10---15---20---25---30---35---40" );
00933     sprintf( szOutput_, "ld r%d, X+        \t ; Load Indirect from Data Space w/Postincrement\n",
00934             u8Rd
00935             );
00936 }
00937
00938 //-----
00939 static void AVR_Disasm_LD_X_Indirect_Predec( char *szOutput_ )
00940 {
00941     uint8_t u8Rd = Register_From_Rd();
00942
00943     //ruler: 0---5---10---15---20---25---30---35---40" );
00944     sprintf( szOutput_, "ld r%d, -X        \t ; Load Indirect from Data Space w/Predecrement\n",
00945             u8Rd
00946             );
00947 }
00948
00949 //-----
00950 static void AVR_Disasm_LD_Y_Indirect( char *szOutput_ )
00951 {
00952     uint8_t u8Rd = Register_From_Rd();
00953
00954     //ruler: 0---5---10---15---20---25---30---35---40" );
00955     sprintf( szOutput_, "ld r%d, Y          \t ; Load Indirect from Data Space\n",
00956             u8Rd
00957             );
00958 }
00959
00960 //-----
00961 static void AVR_Disasm_LD_Y_Indirect_Postinc( char *szOutput_ )
00962 {
00963     uint8_t u8Rd = Register_From_Rd();
00964
00965     //ruler: 0---5---10---15---20---25---30---35---40" );
00966     sprintf( szOutput_, "ld r%d, Y+        \t ; Load Indirect from Data Space w/Postincrement\n",
00967             u8Rd
00968             );
00969 }
00970
00971 //-----
00972 static void AVR_Disasm_LD_Y_Indirect_Predec( char *szOutput_ )
00973 {
00974     uint8_t u8Rd = Register_From_Rd();
00975
00976     //ruler: 0---5---10---15---20---25---30---35---40" );
00977     sprintf( szOutput_, "ld r%d, -Y        \t ; Load Indirect from Data Space w/Predecrement\n",

```

```

00978             u8Rd
00979             );
00980 }
00981
00982 //-----
00983 static void AVR_Disasm_LDD_Y( char *szOutput_ )
00984 {
00985     uint8_t u8Rd = Register_From_Rd();
00986     uint8_t u8q = stCPU.q;
00987
00988     //ruler: 0----5----10---15---20---25---30---35---40" );
00989     sprintf( szOutput_, "ldd r%d, Y+%d          \t ; Load Indirect from Data Space (with Displacement)\n",
00990             u8Rd, u8q
00991             );
00992 }
00993
00994 //-----
00995 static void AVR_Disasm_LD_Z_Indirect( char *szOutput_ )
00996 {
00997     uint8_t u8Rd = Register_From_Rd();
00998
00999     //ruler: 0----5----10---15---20---25---30---35---40" );
01000     sprintf( szOutput_, "ld r%d, Z          \t ; Load Indirect from Data Space\n",
01001             u8Rd
01002             );
01003 }
01004
01005 //-----
01006 static void AVR_Disasm_LD_Z_Indirect_Postinc( char *szOutput_ )
01007 {
01008     uint8_t u8Rd = Register_From_Rd();
01009
01010     //ruler: 0----5----10---15---20---25---30---35---40" );
01011     sprintf( szOutput_, "ld r%d, Z+          \t ; Load Indirect from Data Space w/Postincrement\n",
01012             u8Rd
01013             );
01014 }
01015
01016 //-----
01017 static void AVR_Disasm_LD_Z_Indirect_Preddec( char *szOutput_ )
01018 {
01019     uint8_t u8Rd = Register_From_Rd();
01020
01021     //ruler: 0----5----10---15---20---25---30---35---40" );
01022     sprintf( szOutput_, "ld r%d, -Z          \t ; Load Indirect from Data Space w/Preddecrement\n",
01023             u8Rd
01024             );
01025 }
01026
01027 //-----
01028 static void AVR_Disasm_LDD_Z( char *szOutput_ )
01029 {
01030     uint8_t u8Rd = Register_From_Rd();
01031     uint8_t u8q = stCPU.q;
01032
01033     //ruler: 0----5----10---15---20---25---30---35---40" );
01034     sprintf( szOutput_, "ldd r%d, Z+%d          \t ; Load Indirect from Data Space (with Displacement)\n",
01035             u8Rd, u8q
01036             );
01037 }
01038
01039 //-----
01040 static void AVR_Disasm_STS( char *szOutput_ )
01041 {
01042     uint8_t u8Rd = Register_From_Rd();
01043     uint16_t u16k = stCPU.k;
01044
01045     //ruler: 0----5----10---15---20---25---30---35---40" );
01046     sprintf( szOutput_, "sts %d, r%d          \t ; Store Direct to Data Space: (%d) = r%d\n",
01047             u16k, u8Rd,
01048             u16k, u8Rd
01049             );
01050 }
01051
01052 //-----
01053 static void AVR_Disasm_ST_X_Indirect( char *szOutput_ )
01054 {
01055     uint8_t u8Rd = Register_From_Rd();
01056
01057     //ruler: 0----5----10---15---20---25---30---35---40" );
01058     sprintf( szOutput_, "st X, r%d          \t ; Store Indirect\n",
01059             u8Rd
01060             );
01061 }
01062

```

```

01063 //-----
01064 static void AVR_Disasm_ST_X_Indirect_Postinc( char *szOutput_ )
01065 {
01066     uint8_t u8Rd = Register_From_Rd();
01067
01068     //ruler: 0----5----10---15---20---25---30---35---40" );
01069     sprintf( szOutput_, "st X+, r%d          \t ; Store Indirect w/Postincrement \n",
01070             u8Rd
01071             );
01072 }
01073
01074 //-----
01075 static void AVR_Disasm_ST_X_Indirect_Preddec( char *szOutput_ )
01076 {
01077     uint8_t u8Rd = Register_From_Rd();
01078
01079     //ruler: 0----5----10---15---20---25---30---35---40" );
01080     sprintf( szOutput_, "st -X, r%d          \t ; Store Indirect w/Preddecrement\n",
01081             u8Rd
01082             );
01083 }
01084
01085 //-----
01086 static void AVR_Disasm_ST_Y_Indirect( char *szOutput_ )
01087 {
01088     uint8_t u8Rd = Register_From_Rd();
01089
01090     //ruler: 0----5----10---15---20---25---30---35---40" );
01091     sprintf( szOutput_, "st Y, r%d          \t ; Store Indirect\n",
01092             u8Rd
01093             );
01094 }
01095
01096 //-----
01097 static void AVR_Disasm_ST_Y_Indirect_Postinc( char *szOutput_ )
01098 {
01099     uint8_t u8Rd = Register_From_Rd();
01100
01101     //ruler: 0----5----10---15---20---25---30---35---40" );
01102     sprintf( szOutput_, "st Y+, r%d          \t ; Store Indirect w/Postincrement \n",
01103             u8Rd
01104             );
01105 }
01106
01107 //-----
01108 static void AVR_Disasm_ST_Y_Indirect_Preddec( char *szOutput_ )
01109 {
01110     uint8_t u8Rd = Register_From_Rd();
01111
01112     //ruler: 0----5----10---15---20---25---30---35---40" );
01113     sprintf( szOutput_, "st -Y, r%d          \t ; Store Indirect w/Preddecrement\n",
01114             u8Rd
01115             );
01116 }
01117
01118 //-----
01119 static void AVR_Disasm_STD_Y( char *szOutput_ )
01120 {
01121     uint8_t u8Rd = Register_From_Rd();
01122     uint8_t u8q = stCPU.q;
01123
01124     //ruler: 0----5----10---15---20---25---30---35---40" );
01125     sprintf( szOutput_, "std Y+%d, r%d          \t ; Store Indirect from Data Space (with Displacement)
01126     \n",
01127             u8q, u8Rd
01128             );
01129 }
01130
01131 //-----
01132 static void AVR_Disasm_ST_Z_Indirect( char *szOutput_ )
01133 {
01134     uint8_t u8Rd = Register_From_Rd();
01135
01136     //ruler: 0----5----10---15---20---25---30---35---40" );
01137     sprintf( szOutput_, "st Z, r%d          \t ; Store Indirect\n",
01138             u8Rd
01139             );
01140 }
01141
01142 //-----
01143 static void AVR_Disasm_ST_Z_Indirect_Postinc( char *szOutput_ )
01144 {
01145     uint8_t u8Rd = Register_From_Rd();
01146
01147     //ruler: 0----5----10---15---20---25---30---35---40" );
01148     sprintf( szOutput_, "st Z+, r%d          \t ; Store Indirect w/Postincrement \n",
01149             u8Rd

```

```

01149         );
01150     }
01151
01152     //-----
01153     static void AVR_Disasm_ST_Z_Indirect_Preddec( char *szOutput_ )
01154     {
01155         uint8_t u8Rd = Register_From_Rd();
01156
01157         //ruler: 0---5---10---15---20---25---30---35---40" );
01158         sprintf( szOutput_, "st -Z, r%d          \t ; Store Indirect w/Predecrement\n",
01159                 u8Rd
01160                 );
01161     }
01162
01163     //-----
01164     static void AVR_Disasm_STD_Z( char *szOutput_ )
01165     {
01166         uint8_t u8Rd = Register_From_Rd();
01167         uint8_t u8q = stCPU.q;
01168
01169         //ruler: 0---5---10---15---20---25---30---35---40" );
01170         sprintf( szOutput_, "std Z+%d, r%d          \t ; Store Indirect from Data Space (with Displacement)
01171 \n",
01172                 u8q, u8Rd
01173                 );
01174     }
01175     //-----
01176     static void AVR_Disasm_LPM( char *szOutput_ )
01177     {
01178         //ruler: 0---5---10---15---20---25---30---35---40" );
01179         sprintf( szOutput_, "lpm          \t ; Load Program Memory: r0 = (Z)\n" );
01180     }
01181
01182     //-----
01183     static void AVR_Disasm_LPM_Z( char *szOutput_ )
01184     {
01185         uint8_t u8Rd = Register_From_Rd();
01186
01187         //ruler: 0---5---10---15---20---25---30---35---40" );
01188         sprintf( szOutput_, "lpm r%d, Z          \t ; Load Program Memory: r%d = (Z)\n",
01189                 u8Rd,
01190                 u8Rd
01191                 );
01192     }
01193
01194     //-----
01195     static void AVR_Disasm_LPM_Z_Postinc( char *szOutput_ )
01196     {
01197         uint8_t u8Rd = Register_From_Rd();
01198
01199         //ruler: 0---5---10---15---20---25---30---35---40" );
01200         sprintf( szOutput_, "lpm r%d, Z+          \t ; Load Program Memory with Postincrement: r%d = (Z),
01201 Z = Z + 1\n",
01202                 u8Rd,
01203                 u8Rd
01204                 );
01205     }
01206     //-----
01207     static void AVR_Disasm_ELPM( char *szOutput_ )
01208     {
01209         //ruler: 0---5---10---15---20---25---30---35---40" );
01210         sprintf( szOutput_, "elpm          \t ; (Extended) Load Program Memory: r0 = (Z)\n" );
01211     }
01212
01213     //-----
01214     static void AVR_Disasm_ELPM_Z( char *szOutput_ )
01215     {
01216         uint8_t u8Rd = Register_From_Rd();
01217
01218         //ruler: 0---5---10---15---20---25---30---35---40" );
01219         sprintf( szOutput_, "elpm r%d, Z          \t ; (Extended) Load Program Memory: r%d = (Z)\n",
01220                 u8Rd,
01221                 u8Rd
01222                 );
01223     }
01224
01225     //-----
01226     static void AVR_Disasm_ELPM_Z_Postinc( char *szOutput_ )
01227     {
01228         uint8_t u8Rd = Register_From_Rd();
01229
01230         //ruler: 0---5---10---15---20---25---30---35---40" );
01231         sprintf( szOutput_, "elpm r%d, Z+          \t ; (Extended) Load Program Memory w/Postincrement: r%d
01232 = (Z), Z = Z + 1\n",
01233                 u8Rd,

```

```

01233             u8Rd
01234             );
01235 }
01236
01237 //-----
01238 static void AVR_Disasm_SPM( char *szOutput_ )
01239 {
01240     //ruler: 0----5----10---15---20---25---30---35---40" );
01241     sprintf( szOutput_, "spm                                \t ; Store Program Memory\n" );
01242 }
01243
01244 //-----
01245 static void AVR_Disasm_SPM_Z_Postinc2( char *szOutput_ )
01246 {
01247     //ruler: 0----5----10---15---20---25---30---35---40" );
01248     sprintf( szOutput_, "spm Z+                                \t ; Store Program Memory Z = Z + 2 \n" );
01249 }
01250
01251 //-----
01252 static void AVR_Disasm_IN( char *szOutput_ )
01253 {
01254     uint8_t u8Rd = Register_From_Rd();
01255     uint8_t u8A = stCPU.A;
01256
01257     //ruler: 0----5----10---15---20---25---30---35---40" );
01258     sprintf( szOutput_, "in r%d, %d                                \t ; Load an I/O location to register\n",
01259             u8Rd,
01260             u8A
01261             );
01262 }
01263
01264 //-----
01265 static void AVR_Disasm_OUT( char *szOutput_ )
01266 {
01267     uint8_t u8Rd = Register_From_Rd();
01268     uint8_t u8A = stCPU.A;
01269
01270     //ruler: 0----5----10---15---20---25---30---35---40" );
01271     sprintf( szOutput_, "out %d, r%d                                \t ; Load an I/O location to register\n",
01272             u8A,
01273             u8Rd
01274             );
01275 }
01276
01277 //-----
01278 static void AVR_Disasm_LAC( char *szOutput_ )
01279 {
01280     uint8_t u8Rd = Register_From_Rd();
01281
01282     //ruler: 0----5----10---15---20---25---30---35---40" );
01283     sprintf( szOutput_, "lac Z, r%d                                \t ; Load And Clear\n",
01284             u8Rd
01285             );
01286 }
01287
01288 //-----
01289 static void AVR_Disasm_LAS( char *szOutput_ )
01290 {
01291     uint8_t u8Rd = Register_From_Rd();
01292
01293     //ruler: 0----5----10---15---20---25---30---35---40" );
01294     sprintf( szOutput_, "las Z, r%d                                \t ; Load And Set\n",
01295             u8Rd
01296             );
01297 }
01298
01299 //-----
01300 static void AVR_Disasm_LAT( char *szOutput_ )
01301 {
01302     uint8_t u8Rd = Register_From_Rd();
01303
01304     //ruler: 0----5----10---15---20---25---30---35---40" );
01305     sprintf( szOutput_, "lat Z, r%d                                \t ; Load And Toggle\n",
01306             u8Rd
01307             );
01308 }
01309
01310 //-----
01311 static void AVR_Disasm_LSL( char *szOutput_ )
01312 {
01313     uint8_t u8Rd = Register_From_Rd();
01314
01315     //ruler: 0----5----10---15---20---25---30---35---40" );
01316     sprintf( szOutput_, "lsl r%d                                \t ; Logical shift left r%d by 1 bit\n",
01317             u8Rd,
01318             u8Rd
01319             );

```



```

01320         );
01321     }
01322
01323 //-----
01324 static void AVR_Disasm_LSR( char *szOutput_ )
01325 {
01326     uint8_t u8Rd = Register_From_Rd();
01327
01328     //ruler: 0---5---10---15---20---25---30---35---40" );
01329     sprintf( szOutput_, "lsr r%d          \t ; Logical shift right r%d by 1 bit\n",
01330             u8Rd,
01331             u8Rd
01332         );
01333 }
01334
01335 //-----
01336 static void AVR_Disasm_POP( char *szOutput_ )
01337 {
01338     uint8_t u8Rd = Register_From_Rd();
01339
01340     //ruler: 0---5---10---15---20---25---30---35---40" );
01341     sprintf( szOutput_, "pop r%d          \t ; Pop byte from stack into r%d\n",
01342             u8Rd,
01343             u8Rd
01344         );
01345 }
01346
01347 //-----
01348 static void AVR_Disasm_PUSH( char *szOutput_ )
01349 {
01350     uint8_t u8Rd = Register_From_Rd();
01351
01352     //ruler: 0---5---10---15---20---25---30---35---40" );
01353     sprintf( szOutput_, "push r%d          \t ; Push register r%d to stack\n",
01354             u8Rd,
01355             u8Rd
01356         );
01357 }
01358
01359 //-----
01360 static void AVR_Disasm_ROL( char *szOutput_ )
01361 {
01362     uint8_t u8Rd = Register_From_Rd();
01363
01364     //ruler: 0---5---10---15---20---25---30---35---40" );
01365     sprintf( szOutput_, "rol r%d          \t ; Rotate Left through Carry\n",
01366             u8Rd
01367         );
01368 }
01369
01370 //-----
01371 static void AVR_Disasm_ROR( char *szOutput_ )
01372 {
01373     uint8_t u8Rd = Register_From_Rd();
01374
01375     //ruler: 0---5---10---15---20---25---30---35---40" );
01376     sprintf( szOutput_, "ror r%d          \t ; Rotate Right through Carry\n",
01377             u8Rd
01378         );
01379 }
01380
01381 //-----
01382 static void AVR_Disasm_ASR( char *szOutput_ )
01383 {
01384     uint8_t u8Rd = Register_From_Rd();
01385
01386     //ruler: 0---5---10---15---20---25---30---35---40" );
01387     sprintf( szOutput_, "asr r%d          \t ; Arithmetic Shift Right\n",
01388             u8Rd
01389         );
01390 }
01391
01392 //-----
01393 static void AVR_Disasm_SWAP( char *szOutput_ )
01394 {
01395     uint8_t u8Rd = Register_From_Rd();
01396
01397     //ruler: 0---5---10---15---20---25---30---35---40" );
01398     sprintf( szOutput_, "swap r%d          \t ; Swap high/low Nibbles in Register\n",
01399             u8Rd
01400         );
01401 }
01402
01403 //-----
01404 static void AVR_Disasm_BSET( char *szOutput_ )
01405 {
01406     uint8_t u8s = stCPU.s;

```

```

01407
01408 //ruler: 0----5----10----15----20----25----30----35----40" );
01409 sprintf( szOutput_, "bset %d          \t ; Set bit %d in status register\n",
01410          u8s,
01411          u8s
01412          );
01413 }
01414
01415 //-----
01416 static void AVR_Disasm_BCLR( char *szOutput_ )
01417 {
01418     uint8_t u8s = stCPU.s;
01419
01420     //ruler: 0----5----10----15----20----25----30----35----40" );
01421     sprintf( szOutput_, "bclr %d          \t ; Clear bit %d in status register\n",
01422             u8s,
01423             u8s
01424             );
01425 }
01426
01427 //-----
01428 static void AVR_Disasm_SBI( char *szOutput_ )
01429 {
01430     uint8_t u8b = stCPU.b;
01431     uint8_t u8A = stCPU.A;
01432
01433     //ruler: 0----5----10----15----20----25----30----35----40" );
01434     sprintf( szOutput_, "sbi %d, %d          \t ; Set bit in I/O register\n",
01435             u8A,
01436             u8b
01437             );
01438 }
01439
01440 //-----
01441 static void AVR_Disasm_CBI( char *szOutput_ )
01442 {
01443     uint8_t u8s = stCPU.b;
01444     uint8_t u8A = stCPU.A;
01445
01446     //ruler: 0----5----10----15----20----25----30----35----40" );
01447     sprintf( szOutput_, "cbi %d, %d          \t ; Clear bit in I/O register\n",
01448             u8A,
01449             u8s
01450             );
01451 }
01452
01453 //-----
01454 static void AVR_Disasm_BST( char *szOutput_ )
01455 {
01456     uint8_t u8Rd = Register_From_Rd();
01457     uint8_t u8b = stCPU.b;
01458
01459     //ruler: 0----5----10----15----20----25----30----35----40" );
01460     sprintf( szOutput_, "bst r%d, %d          \t ; Store Bit %d of r%d in the T register\n",
01461             u8Rd, u8b,
01462             u8b, u8Rd
01463             );
01464 }
01465
01466 //-----
01467 static void AVR_Disasm_BLD( char *szOutput_ )
01468 {
01469     uint8_t u8Rd = Register_From_Rd();
01470     uint8_t u8b = stCPU.b;
01471
01472     //ruler: 0----5----10----15----20----25----30----35----40" );
01473     sprintf( szOutput_, "bld r%d, %d          \t ; Load the T register into Bit %d of r%d\n",
01474             u8Rd, u8b,
01475             u8b, u8Rd
01476             );
01477 }
01478
01479 //-----
01480 static void AVR_Disasm_SEC( char *szOutput_ )
01481 {
01482     //ruler: 0----5----10----15----20----25----30----35----40" );
01483     sprintf( szOutput_, "sec          \t ; Set the carry flag in the SR\n" );
01484 }
01485
01486 //-----
01487 static void AVR_Disasm_CLC( char *szOutput_ )
01488 {
01489     //ruler: 0----5----10----15----20----25----30----35----40" );
01490     sprintf( szOutput_, "clc          \t ; Clear the carry flag in the SR\n" );
01491 }
01492
01493 //-----

```

```

01494 static void AVR_Disasm_SEN( char *szOutput_ )
01495 {
01496     //ruler: 0----5----10---15---20---25---30---35---40" );
01497     sprintf( szOutput_, "sen                                \t ; Set the negative flag in the SR\n" );
01498 }
01499
01500 //-----
01501 static void AVR_Disasm_CLN( char *szOutput_ )
01502 {
01503     //ruler: 0----5----10---15---20---25---30---35---40" );
01504     sprintf( szOutput_, "cln                                \t ; Clear the negative flag in the SR\n" );
01505 }
01506
01507 //-----
01508 static void AVR_Disasm_SEZ( char *szOutput_ )
01509 {
01510     //ruler: 0----5----10---15---20---25---30---35---40" );
01511     sprintf( szOutput_, "sez                                \t ; Set the zero flag in the SR\n" );
01512 }
01513
01514 //-----
01515 static void AVR_Disasm_CLZ( char *szOutput_ )
01516 {
01517     //ruler: 0----5----10---15---20---25---30---35---40" );
01518     sprintf( szOutput_, "clz                                \t ; Clear the zero flag in the SR\n" );
01519 }
01520
01521 //-----
01522 static void AVR_Disasm_SEI( char *szOutput_ )
01523 {
01524     //ruler: 0----5----10---15---20---25---30---35---40" );
01525     sprintf( szOutput_, "sei                                \t ; Enable MCU interrupts\n" );
01526 }
01527
01528 //-----
01529 static void AVR_Disasm_CLI( char *szOutput_ )
01530 {
01531     //ruler: 0----5----10---15---20---25---30---35---40" );
01532     sprintf( szOutput_, "cli                                \t ; Disable MCU interrupts\n" );
01533 }
01534
01535 //-----
01536 static void AVR_Disasm_SES( char *szOutput_ )
01537 {
01538     //ruler: 0----5----10---15---20---25---30---35---40" );
01539     sprintf( szOutput_, "ses                                \t ; Set the sign flag in the SR\n" );
01540 }
01541
01542 //-----
01543 static void AVR_Disasm_CLS( char *szOutput_ )
01544 {
01545     //ruler: 0----5----10---15---20---25---30---35---40" );
01546     sprintf( szOutput_, "cls                                \t ; Clear the sign flag in the SR\n" );
01547 }
01548
01549 //-----
01550 static void AVR_Disasm_SEV( char *szOutput_ )
01551 {
01552     //ruler: 0----5----10---15---20---25---30---35---40" );
01553     sprintf( szOutput_, "sev                                \t ; Set the overflow flag in the SR\n" );
01554 }
01555
01556 //-----
01557 static void AVR_Disasm_CLV( char *szOutput_ )
01558 {
01559     //ruler: 0----5----10---15---20---25---30---35---40" );
01560     sprintf( szOutput_, "clv                                \t ; Clear the overflow flag in the SR\n" );
01561 }
01562
01563 //-----
01564 static void AVR_Disasm_SET( char *szOutput_ )
01565 {
01566     //ruler: 0----5----10---15---20---25---30---35---40" );
01567     sprintf( szOutput_, "set                                \t ; Set the T-flag in the SR\n" );
01568 }
01569
01570 //-----
01571 static void AVR_Disasm_CLT( char *szOutput_ )
01572 {
01573     //ruler: 0----5----10---15---20---25---30---35---40" );
01574     sprintf( szOutput_, "clt                                \t ; Clear the T-flag in the SR\n" );
01575 }
01576
01577 //-----
01578 static void AVR_Disasm_SEH( char *szOutput_ )
01579 {
01580     //ruler: 0----5----10---15---20---25---30---35---40" );

```

```

01581     sprintf( szOutput_, "seh                                \t ; Set half-carry flag in SR\n" );
01582 }
01583
01584 //-----
01585 static void AVR_Disasm_CLH( char *szOutput_ )
01586 {
01587     //ruler: 0---5---10---15---20---25---30---35---40" );
01588     sprintf( szOutput_, "clh                                \t ; Clear half-carry flag in SR\n" );
01589 }
01590
01591 //-----
01592 static void AVR_Disasm_BREAK( char *szOutput_ )
01593 {
01594     //ruler: 0---5---10---15---20---25---30---35---40" );
01595     sprintf( szOutput_, "break                                \t ; Halt for debugger\n" );
01596 }
01597
01598 //-----
01599 static void AVR_Disasm_NOP( char *szOutput_ )
01600 {
01601     //ruler: 0---5---10---15---20---25---30---35---40" );
01602     sprintf( szOutput_, "nop                                \t ; Do nothing\n" );
01603 }
01604
01605 //-----
01606 static void AVR_Disasm_SLEEP( char *szOutput_ )
01607 {
01608     //ruler: 0---5---10---15---20---25---30---35---40" );
01609     sprintf( szOutput_, "sleep                                \t ; Put MCU into sleep mode\n" );
01610 }
01611
01612 //-----
01613 static void AVR_Disasm_WDR( char *szOutput_ )
01614 {
01615     //ruler: 0---5---10---15---20---25---30---35---40" );
01616     sprintf( szOutput_, "wdr                                \t ; Reset Watchdog Timer\n" );
01617 }
01618
01619 //-----
01620 static void AVR_Disasm_XCH( char *szOutput_ )
01621 {
01622     uint8_t u8Rd = Register_From_Rd();
01623
01624     //ruler: 0---5---10---15---20---25---30---35---40" );
01625     sprintf( szOutput_, "xch Z, r%d                                \t ; Exchange registers w/memory\n",
01626             u8Rd );
01627 }
01628 }
01629
01630 //-----
01631 static void AVR_Disasm_Unimplemented( char *szOutput_ )
01632 {
01633     sprintf( szOutput_, ".db 0x%04X ; Data (not an opcode)\n", stCPU.pu16ROM[ stCPU.u32PC ] );
01634 }
01635
01636 //-----
01637 AVR_Disasm AVR_Disasm_Function( uint16_t OP_ )
01638 {
01639     // Special instructions - "static" encoding
01640     switch ( OP_ )
01641     {
01642     case 0x0000: return AVR_Disasm_NOP;
01643
01644     case 0x9408: return AVR_Disasm_SEC;
01645     case 0x9409: return AVR_Disasm_IJMP;
01646     case 0x9418: return AVR_Disasm_SEZ;
01647     case 0x9419: return AVR_Disasm_EIJMP;
01648     case 0x9428: return AVR_Disasm_SEN;
01649     case 0x9438: return AVR_Disasm_SEV;
01650     case 0x9448: return AVR_Disasm_SES;
01651     case 0x9458: return AVR_Disasm_SEH;
01652     case 0x9468: return AVR_Disasm_SET;
01653     case 0x9478: return AVR_Disasm_SEI;
01654
01655     case 0x9488: return AVR_Disasm_CLC;
01656     case 0x9498: return AVR_Disasm_CLZ;
01657     case 0x94A8: return AVR_Disasm_CLN;
01658     case 0x94B8: return AVR_Disasm_CLV;
01659     case 0x94C8: return AVR_Disasm_CLS;
01660     case 0x94D8: return AVR_Disasm_CLH;
01661     case 0x94E8: return AVR_Disasm_CLT;
01662     case 0x94F8: return AVR_Disasm_CLI;
01663
01664     case 0x9508: return AVR_Disasm_RET;
01665     case 0x9509: return AVR_Disasm_ICALL;
01666     case 0x9518: return AVR_Disasm_RETI;
01667     case 0x9519: return AVR_Disasm_EICALL;

```

```

01668     case 0x9588: return AVR_Disasm_SLEEP;
01669     case 0x9598: return AVR_Disasm_BREAK;
01670     case 0x95A8: return AVR_Disasm_WDR;
01671     case 0x95C8: return AVR_Disasm_LPM;
01672     case 0x95D8: return AVR_Disasm_ELPM;
01673     case 0x95E8: return AVR_Disasm_SPM;
01674     case 0x95F8: return AVR_Disasm_SPM_Z_Postinc2;
01675     }
01676
01677 #if 0
01678 // Note: These disasm handlers are generalized versions of specific mnemonics in the above list.
01679 // For disassembly, it's probably easier to read the output from the more "specific" mnemonics, so
01680 // those are used. For emulation, using the generalized functions may be more desirable.
01681 switch( OP_ & 0xFF8F)
01682 {
01683     case 0x9408: return AVR_Disasm_BSET;
01684     case 0x9488: return AVR_Disasm_BCLR;
01685 }
01686 #endif
01687
01688 switch (OP_ & 0xFF88)
01689 {
01690     case 0x0300: return AVR_Disasm_MULSU;
01691     case 0x0308: return AVR_Disasm_FML;
01692     case 0x0380: return AVR_Disasm_FMULS;
01693     case 0x0388: return AVR_Disasm_FMULSU;
01694 }
01695
01696 switch (OP_ & 0xFF0F)
01697 {
01698     case 0x940B: return AVR_Disasm_DES;
01699     case 0xEF0F: return AVR_Disasm_SER;
01700 }
01701
01702 switch (OP_ & 0xFF00)
01703 {
01704     case 0x0100: return AVR_Disasm_MOVW;
01705     case 0x9600: return AVR_Disasm_ADIW;
01706     case 0x9700: return AVR_Disasm_SBIW;
01707
01708     case 0x9800: return AVR_Disasm_CBI;
01709     case 0x9900: return AVR_Disasm_SBIC;
01710     case 0x9A00: return AVR_Disasm_SBI;
01711     case 0x9B00: return AVR_Disasm_SBIS;
01712 }
01713
01714 switch (OP_ & 0xFE0F)
01715 {
01716     case 0x8008: return AVR_Disasm_LD_Y_Indirect;
01717     case 0x8000: return AVR_Disasm_LD_Z_Indirect;
01718     case 0x8200: return AVR_Disasm_ST_Z_Indirect;
01719     case 0x8208: return AVR_Disasm_ST_Y_Indirect;
01720
01721 // -- Single 5-bit register...
01722     case 0x9000: return AVR_Disasm_LDS;
01723     case 0x9001: return AVR_Disasm_LD_Z_Indirect_Postinc;
01724     case 0x9002: return AVR_Disasm_LD_Z_Indirect_Predec;
01725     case 0x9004: return AVR_Disasm_LPM_Z;
01726     case 0x9005: return AVR_Disasm_LPM_Z_Postinc;
01727     case 0x9006: return AVR_Disasm_ELPM_Z;
01728     case 0x9007: return AVR_Disasm_ELPM_Z_Postinc;
01729     case 0x9009: return AVR_Disasm_LD_Y_Indirect_Postinc;
01730     case 0x900A: return AVR_Disasm_LD_Y_Indirect_Predec;
01731     case 0x900C: return AVR_Disasm_LD_X_Indirect;
01732     case 0x900D: return AVR_Disasm_LD_X_Indirect_Postinc;
01733     case 0x900E: return AVR_Disasm_LD_X_Indirect_Predec;
01734     case 0x900F: return AVR_Disasm_POP;
01735
01736     case 0x9200: return AVR_Disasm_STS;
01737     case 0x9201: return AVR_Disasm_ST_Z_Indirect_Postinc;
01738     case 0x9202: return AVR_Disasm_ST_Z_Indirect_Predec;
01739     case 0x9204: return AVR_Disasm_XCH;
01740     case 0x9205: return AVR_Disasm_LAS;
01741     case 0x9206: return AVR_Disasm_LAC;
01742     case 0x9207: return AVR_Disasm_LAT;
01743     case 0x9209: return AVR_Disasm_ST_Y_Indirect_Postinc;
01744     case 0x920A: return AVR_Disasm_ST_Y_Indirect_Predec;
01745     case 0x920C: return AVR_Disasm_ST_X_Indirect;
01746     case 0x920D: return AVR_Disasm_ST_X_Indirect_Postinc;
01747     case 0x920E: return AVR_Disasm_ST_X_Indirect_Predec;
01748     case 0x920F: return AVR_Disasm_PUSH;
01749
01750 // -- One-operand instructions
01751     case 0x9400: return AVR_Disasm_COM;
01752     case 0x9401: return AVR_Disasm_NEG;
01753     case 0x9402: return AVR_Disasm_SWAP;
01754     case 0x9403: return AVR_Disasm_INC;

```

```

01755     case 0x9405: return AVR_Disasm_ASR;
01756     case 0x9406: return AVR_Disasm_LSR;
01757     case 0x9407: return AVR_Disasm_ROR;
01758     case 0x940A: return AVR_Disasm_DEC;
01759
01760 }
01761 switch (OP_ & 0xFE0E)
01762 {
01763     case 0x940C: return AVR_Disasm_JMP;
01764     case 0x940E: return AVR_Disasm_CALL;
01765 }
01766
01767 switch (OP_ & 0xFE08)
01768 {
01769
01770     // -- BLD/BST Encoding
01771     case 0xF800: return AVR_Disasm_BLD;
01772     case 0xFA00: return AVR_Disasm_BST;
01773     // -- SBRC/SBRS Encoding
01774     case 0xFC00: return AVR_Disasm_SBRC;
01775     case 0xFE00: return AVR_Disasm_SBRS;
01776 }
01777
01778 switch (OP_ & 0xFC07)
01779 {
01780     // -- Conditional branches
01781     case 0xF000: return AVR_Disasm_BRCS;
01782     // case 0xF000: return AVR_Disasm_BRLO;           // AKA AVR_Disasm_BRCS;
01783     case 0xF001: return AVR_Disasm_BREQ;
01784     case 0xF002: return AVR_Disasm_BRMI;
01785     case 0xF003: return AVR_Disasm_BRVS;
01786     case 0xF004: return AVR_Disasm_BRLT;
01787     case 0xF006: return AVR_Disasm_BRTS;
01788     case 0xF007: return AVR_Disasm_BRIE;
01789     case 0xF400: return AVR_Disasm_BRCC;
01790     // case 0xF400: return AVR_Disasm_BRSH;           // AKA AVR_Disasm_BRCC;
01791     case 0xF401: return AVR_Disasm_BRNE;
01792     case 0xF402: return AVR_Disasm_BRPL;
01793     case 0xF403: return AVR_Disasm_BRVC;
01794     case 0xF404: return AVR_Disasm_BRGE;
01795     case 0xF405: return AVR_Disasm_BRHC;
01796     case 0xF406: return AVR_Disasm_BRTC;
01797     case 0xF407: return AVR_Disasm_BRID;
01798 }
01799
01800 switch (OP_ & 0xFC00)
01801 {
01802     // -- 4-bit register pair
01803     case 0x0200: return AVR_Disasm_MULS;
01804
01805     // -- 5-bit register pairs --
01806     case 0x0400: return AVR_Disasm_CPC;
01807     case 0x0800: return AVR_Disasm_SBC;
01808     case 0x0C00: return AVR_Disasm_ADD;
01809     // case 0x0C00: return AVR_Disasm_LSL; (!! Implemented with: " add rd, rd"
01810     case 0x1000: return AVR_Disasm_CPSE;
01811     case 0x1300: return AVR_Disasm_ROL;
01812     case 0x1400: return AVR_Disasm_CP;
01813     case 0x1C00: return AVR_Disasm_ADC;
01814     case 0x1800: return AVR_Disasm_SUB;
01815     case 0x2000: return AVR_Disasm_AND;
01816     // case 0x2000: return AVR_Disasm_TST; (!! Implemented with: " and rd, rd"
01817     case 0x2400: return AVR_Disasm_EOR;
01818     case 0x2C00: return AVR_Disasm_MOV;
01819     case 0x2800: return AVR_Disasm_OR;
01820
01821     // -- 5-bit register pairs -- Destination = R1:R0
01822     case 0x9C00: return AVR_Disasm_MUL;
01823 }
01824
01825 switch (OP_ & 0xF800)
01826 {
01827     case 0xB800: return AVR_Disasm_OUT;
01828     case 0xB000: return AVR_Disasm_IN;
01829 }
01830
01831 switch (OP_ & 0xF000)
01832 {
01833     // -- Register immediate --
01834     case 0x3000: return AVR_Disasm_CPI;
01835     case 0x4000: return AVR_Disasm_SBCI;
01836     case 0x5000: return AVR_Disasm_SUBI;
01837     case 0x6000: return AVR_Disasm_ORI; // return AVR_Disasm_SBR;
01838     case 0x7000: return AVR_Disasm_ANDI;
01839
01840     //-- 12-bit immediate
01841     case 0xC000: return AVR_Disasm_RJMP;

```

```

01842     case 0xD000: return AVR_Disasm_RCALL;
01843
01844     // -- Register immediate
01845     case 0xE000: return AVR_Disasm_LDI;
01846     }
01847
01848     switch (OP_ & 0xD208)
01849     {
01850     // -- 7-bit signed offset
01851     case 0x8000: return AVR_Disasm_LDD_Z;
01852     case 0x8008: return AVR_Disasm_LDD_Y;
01853     case 0x8200: return AVR_Disasm_STD_Z;
01854     case 0x8208: return AVR_Disasm_STD_Y;
01855     }
01856
01857     return AVR_Disasm_Unimplemented;
01858 }
01859

```

## 4.13 src/avr\_cpu/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\_Function()

```
AVR_Disasm AVR_Disasm_Function (
    uint16_t OP_ )
```

AVR\_Disasm\_Function.

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





### 4.15.1 Detailed Description

CPU Interrupt management.

Definition in file [avr\\_interrupt.c](#).

### 4.15.2 Function Documentation

#### 4.15.2.1 AVR\_ClearCandidate()

```
void AVR_ClearCandidate (
    uint8_t u8Vector_ )
```

AVR\_ClearCandidate.

Parameters

<i>u8↔ Vector_</i>	Vector to clear pending interrupt for.
------------------------	--

Definition at line 63 of file [avr\\_interrupt.c](#).

#### 4.15.2.2 AVR\_Interrupt()

```
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 75 of file [avr\\_interrupt.c](#).

#### 4.15.2.3 AVR\_InterruptCandidate()

```
void AVR_InterruptCandidate (
    uint8_t u8Vector_ )
```

AVR\_InterruptCandidate.

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



```

00075 void AVR_Interrupt( void )
00076 {
00077     // First - check to see if there's an interrupt pending.
00078     if (stCPU.u8IntPriority == 255 || stCPU.pstRAM->stRegisters.SREG.I == 0)
00079     {
00080         return; // no interrupt pending
00081     }
00082
00083     // Push the current PC to stack.
00084     uint16_t u16SP = (((uint16_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
00085                     (((uint16_t)stCPU.pstRAM->stRegisters.SPL.r));
00086
00087     uint16_t u16StoredPC = stCPU.u32PC;
00088
00089     stCPU.pstRAM->au8RAM[ u16SP ] = (uint8_t)(u16StoredPC & 0x00FF);
00090     stCPU.pstRAM->au8RAM[ u16SP - 1 ] = (uint8_t)(u16StoredPC >> 8);
00091
00092     // Stack is post-decremented
00093     u16SP -= 2;
00094
00095     // Store the new SP.
00096     stCPU.pstRAM->stRegisters.SPH.r = (u16SP >> 8);
00097     stCPU.pstRAM->stRegisters.SPL.r = (u16SP & 0x00FF);
00098
00099     // Read the new PC from the vector table
00100     uint16_t u16NewPC = (uint16_t)(stCPU.u8IntPriority * 2);
00101
00102     // Set the new PC
00103     stCPU.u32PC = u16NewPC;
00104     stCPU.u16ExtraPC = 0;
00105
00106     // Clear the "I" (global interrupt enabled) register in the SR
00107     stCPU.pstRAM->stRegisters.SREG.I = 0;
00108
00109     // Run the interrupt-acknowledge callback associated with this vector
00110     uint8_t u8Pri = stCPU.u8IntPriority;
00111     if (u8Pri < 32 && stCPU.apfInterruptCallbacks[ u8Pri ])
00112     {
00113         stCPU.apfInterruptCallbacks[ u8Pri ]( u8Pri );
00114     }
00115
00116     // Reset the CPU interrupt priority
00117     stCPU.u32IntFlags &= ~(1 << u8Pri);
00118     AVR_NextInterrupt();
00119
00120     // Run the generic interrupt callout routine
00121     InterruptCallout_Run( true, u8Pri );
00122
00123     // Clear any sleep-mode flags currently set
00124     stCPU.bAsleep = false;
00125 }

```

## 4.17 src/avr\_cpu/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 AVR\_ClearCandidate()

```
void AVR_ClearCandidate (
    uint8_t u8Vector_ )
```

AVR\_ClearCandidate.

Parameters

<i>u8↔ Vector_</i>	Vector to clear pending interrupt for.
------------------------	--

Definition at line 63 of file [avr\\_interrupt.c](#).

#### 4.17.2.2 AVR\_Interrupt()

```
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 75 of file [avr\\_interrupt.c](#).

#### 4.17.2.3 AVR\_InterruptCandidate()

```
void AVR_InterruptCandidate (
    uint8_t u8Vector_ )
```

AVR\_InterruptCandidate.

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

## Parameters

<i>u8Vector_</i>	- Candidate interrupt vector.
------------------	-------------------------------

Definition at line 47 of file [avr\\_interrupt.c](#).

## 4.18 avr\_interrupt.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ )  )\ )      (      )\ )  |
00004 *      ((/( ((/(      \      ( ((/(      | -- [ Funkenstein ] -----
00005 *      /( )  /( )  ((( ( ) \ \  /( )  | -- [ Little ] -----
00006 *      ( ) _ ( ) )  ) \ _ \ ( ) ( ( ) _ | -- [ AVR ] -----
00007 *      | | _ | | ( ) \ ( ) \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ / _ \ \ \ v / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ / _ \ \ \ / | _ \ |
00010 *                                          | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00023 #ifndef __AVR_INTERRUPT_H__
00024 #define __AVR_INTERRUPT_H__
00025
00026 #include <stdint.h>
00027 #include "emu_config.h"
00028 #include "avr_cpu.h"
00029
00030 //-----
00039 void AVR_InterruptCandidate( uint8_t u8Vector_ );
00040
00041 //-----
00047 void AVR_ClearCandidate( uint8_t u8Vector_ );
00048
00049 //-----
00058 void AVR_Interrupt( void );
00059
00060 #endif //__AVR_INTERRUPT_H__

```

## 4.19 src/avr\_cpu/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 IO\_AddClocker()

```
void IO_AddClocker (
    AVRPeripheral * pstPeriph_ )
```

*IO\_AddClocker.*

Parameters

<i>pst↔ Periph_</i>	
-------------------------	--

Definition at line 69 of file [avr\\_io.c](#).

##### 4.19.2.2 IO\_AddReader()

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

*IO\_AddReader.*

Parameters

<i>pst↔ Periph_</i>	
<i>addr_</i>	

Definition at line 33 of file [avr\\_io.c](#).

#### 4.19.2.3 IO\_AddWriter()

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

IO\_AddWriter.

##### Parameters

<i>pstPeriph_</i>	
<i>addr_</i>	

Definition at line 51 of file [avr\\_io.c](#).

#### 4.19.2.4 IO\_Clock()

```
void IO_Clock (
    void )
```

IO\_Clock.

Definition at line 115 of file [avr\\_io.c](#).

#### 4.19.2.5 IO\_Read()

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

IO\_Read.

##### Parameters

<i>addr_</i>	
<i>value_</i>	

Definition at line 101 of file [avr\\_io.c](#).





```

00055     node = (IOWriterList*)malloc(sizeof(*node));
00056     if (!node)
00057     {
00058         return;
00059     }
00060
00061     node->next = stCPU.apstPeriphWriteTable[addr_];
00062     node->pfWriter = pstPeriph_>pfWrite;
00063     node->pvContext = pstPeriph_>pvContext;
00064
00065     stCPU.apstPeriphWriteTable[addr_] = node;
00066 }
00067
00068 //-----
00069 void IO_AddClocker( AVRPeripheral *pstPeriph_ )
00070 {
00071     IOClockList *node = NULL;
00072
00073     node = (IOClockList*)malloc(sizeof(*node));
00074     if (!node)
00075     {
00076         return;
00077     }
00078
00079     node->next = stCPU.pstClockList;
00080     node->pfClock = pstPeriph_>pfClock;
00081     node->pvContext = pstPeriph_>pvContext;
00082
00083     stCPU.pstClockList = node;
00084 }
00085
00086 //-----
00087 void IO_Write( uint8_t addr_, uint8_t value_ )
00088 {
00089     IOWriterList *node = stCPU.apstPeriphWriteTable[addr_];
00090     while (node)
00091     {
00092         if (node->pfWriter)
00093         {
00094             node->pfWriter( node->pvContext, addr_, value_ );
00095         }
00096         node = node->next;
00097     }
00098 }
00099
00100 //-----
00101 void IO_Read( uint8_t addr_, uint8_t *value_ )
00102 {
00103     IOReaderList *node = stCPU.apstPeriphReadTable[addr_];
00104     while (node)
00105     {
00106         if (node->pfReader)
00107         {
00108             node->pfReader( node->pvContext, addr_, value_ );
00109         }
00110         node = node->next;
00111     }
00112 }
00113
00114 //-----
00115 void IO_Clock( void )
00116 {
00117     IOClockList *node = stCPU.pstClockList;
00118     while (node)
00119     {
00120         if (node->pfClock)
00121         {
00122             node->pfClock( node->pvContext );
00123         }
00124         node = node->next;
00125     }
00126 }

```

## 4.21 src/avr\_cpu/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 IO\_AddClocker()

```
void IO_AddClocker (
    AVRPeripheral * pstPeriph_ )
```

*IO\_AddClocker.*

## Parameters

<i>pst</i> ↔ <i>Periph_</i>	
--------------------------------	--

Definition at line 69 of file [avr\\_io.c](#).

## 4.21.2.2 IO\_AddReader()

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

IO\_AddReader.

## Parameters

<i>pst</i> ↔ <i>Periph_</i>	
<i>addr_</i>	

Definition at line 33 of file [avr\\_io.c](#).

## 4.21.2.3 IO\_AddWriter()

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

IO\_AddWriter.

## Parameters

<i>pst</i> ↔ <i>Periph_</i>	
<i>addr_</i>	

Definition at line 51 of file [avr\\_io.c](#).

## 4.21.2.4 IO\_Clock()

```
void IO_Clock (
    void )
```

IO\_Clock.

Definition at line 115 of file [avr\\_io.c](#).

#### 4.21.2.5 IO\_Read()

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

IO\_Read.

##### Parameters

<i>addr</i> ↔	
—	
<i>value</i> ↔	
—	

Definition at line 101 of file [avr\\_io.c](#).

#### 4.21.2.6 IO\_Write()

```
void IO_Write (
    uint8_t addr_,
    uint8_t value_ )
```

IO\_Write.

##### Parameters

<i>addr</i> ↔	
—	
<i>value</i> ↔	
—	

Definition at line 87 of file [avr\\_io.c](#).

## 4.22 avr\_io.h

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ )  )\ )      (      |
00004 *      ((/( ((/(      \      | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( ) \  )\ / ( )      | -- [ Little ] -----
00006 *      ( ) _ | ( )      ) \ _ ) \ ( ( ) ( ( )      | -- [ AVR ] -----
00007 *      | | _ | |      ( ) _ \ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
```

```

00008 * | _ | | _ | / _ \ \ V / | / | -- [ Runtime ] -----
00009 * | _ | | _ | / _ \ \ V / | _ | |
00010 * | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00022 #ifndef __AVR_IO_H__
00023 #define __AVR_IO_H__
00024
00025 #include "avr_peripheral.h"
00026
00027 //-----
00028 typedef struct _IOReaderList
00029 {
00030     struct _IOReaderList *next;
00031     void *pvContext;
00032     PeriphRead pfReader;
00033 } IOReaderList;
00034
00035 //-----
00036 typedef struct _IOWriterList
00037 {
00038     struct _IOWriterList *next;
00039     void *pvContext;
00040     PeriphWrite pfWriter;
00041 } IOWriterList;
00042
00043 //-----
00044 typedef struct _IOClockList
00045 {
00046     struct _IOClockList *next;
00047     void *pvContext;
00048     PeriphClock pfClock;
00049 } IOClockList;
00050
00051 //-----
00058 void IO_AddReader( AVRPeripheral *pstPeriph_, uint8_t addr_);
00059
00060 //-----
00067 void IO_AddWriter( AVRPeripheral *pstPeriph_, uint8_t addr_);
00068
00069 //-----
00075 void IO_AddClocker( AVRPeripheral *pstPeriph_ );
00076
00077 //-----
00084 void IO_Write( uint8_t addr_, uint8_t value_ );
00085
00086 //-----
00093 void IO_Read( uint8_t addr_, uint8_t *value_ );
00094
00095 //-----
00100 void IO_Clock( void );
00101
00102 #endif

```

## 4.23 src/avr\_cpu/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\_SBRB ()
- static uint8\_t AVR\_Opcode\_Cycles\_SBIC ()
- static uint8\_t AVR\_Opcode\_Cycles\_SBIW ()
- 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\_Opcode\_Cycles.*

### 4.23.1 Detailed Description

Opcode cycle counting functions.

Definition in file [avr\\_op\\_cycles.c](#).

### 4.23.2 Function Documentation

#### 4.23.2.1 AVR\_Opcode\_Cycles()

```
uint8_t AVR_Opcode_Cycles (
    uint16_t OP_ )
```

*AVR\_Opcode\_Cycles.*



## Parameters

$O \leftarrow$	Opcode to compute the minimum cycles to execute for
$P \leftarrow$	
—	

## Returns

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

Definition at line 892 of file [avr\\_op\\_cycles.c](#).

## 4.23.2.2 AVR\_Opcode\_Cycles\_CALL()

```
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](#).

## 4.23.2.3 AVR\_Opcode\_Cycles\_CBI()

```
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](#).

## 4.23.2.4 AVR\_Opcode\_Cycles\_ICALL()

```
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](#).

## 4.23.2.5 AVR\_Opcode\_Cycles\_LD\_Z\_Indirect\_Postinc()

```
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](#).

#### 4.23.2.6 AVR\_Opcode\_Cycles\_LD\_Z\_Indirect\_Predc()

```
static uint8_t AVR_Opcode_Cycles_LD_Z_Indirect_Predc ( ) [static]
```

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 514 of file [avr\\_op\\_cycles.c](#).

#### 4.23.2.7 AVR\_Opcode\_Cycles\_RCALL()

```
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](#).

#### 4.23.2.8 AVR\_Opcode\_Cycles\_RET()

```
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](#).

#### 4.23.2.9 AVR\_Opcode\_Cycles\_RETI()

```
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](#).

#### 4.23.2.10 AVR\_Opcode\_Cycles\_SBI()

```
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](#).

#### 4.23.2.11 AVR\_Opcode\_Cycles\_SPM()

```
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](#).

#### 4.23.2.12 AVR\_Opcode\_Cycles\_SPM\_Z\_Postinc2()

```
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](#).

#### 4.23.2.13 AVR\_Opcode\_Cycles\_ST\_X\_Indirect()

```
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](#).

#### 4.23.2.14 AVR\_Opcode\_Cycles\_ST\_X\_Indirect\_Postinc()

```
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](#).

#### 4.23.2.15 AVR\_Opcode\_Cycles\_ST\_X\_Indirect\_Predec()

```
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](#).

#### 4.23.2.16 AVR\_Opcode\_Cycles\_ST\_Y\_Indirect()

```
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](#).

#### 4.23.2.17 AVR\_Opcode\_Cycles\_ST\_Y\_Indirect\_Postinc()

```
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](#).

#### 4.23.2.18 AVR\_Opcode\_Cycles\_ST\_Y\_Indirect\_Predec()

```
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](#).

#### 4.23.2.19 AVR\_Opcode\_Cycles\_ST\_Z\_Indirect()

```
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](#).

#### 4.23.2.20 AVR\_Opcode\_Cycles\_ST\_Z\_Indirect\_Postinc()

```
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](#).

## 4.23.2.21 AVR\_Opcode\_Cycles\_ST\_Z\_Indirect\_Predc()

```
static uint8_t AVR_Opcode_Cycles_ST_Z_Indirect_Predc ( ) [static]
```

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 586 of file [avr\\_op\\_cycles.c](#).

## 4.23.2.22 AVR\_Opcode\_Cycles\_STD\_Y()

```
static uint8_t AVR_Opcode_Cycles_STD_Y ( ) [static]
```

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 568 of file [avr\\_op\\_cycles.c](#).

## 4.23.2.23 AVR\_Opcode\_Cycles\_STD\_Z()

```
static uint8_t AVR_Opcode_Cycles_STD_Z ( ) [static]
```

! ToDo - Cycles on XMEGA/tinyAVR

Definition at line 592 of file [avr\\_op\\_cycles.c](#).

## 4.24 avr\_op\_cycles.c

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      )\ ) |
00004 *      ((/( ((/(      )\      ( ((/( | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( ) )\ )\ / ( ) | -- [ Little ] -----
00006 *      ( ) _ ( ) )\ _ )\ ( ( ) ( ) ( ) | -- [ AVR ] -----
00007 *      | _ | | ( ) _ ( ) \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ / _ \ \ v / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ / _ \ \ v / | _ \ |
00010 *      | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #include <stdint.h>
00022 #include <stdio.h>
00023
00024 #include "emu_config.h"
00025
00026 #include "avr_op_decode.h"
00027 #include "avr_opcodes.h"
00028 #include "avr_op_size.h"
00029 #include "avr_cpu.h"
00030 #include "avr_cpu_print.h"
00031 #include "avr_loader.h"
00032
00033 //-----
00034 static uint8_t AVR_Opcode_Cycles_ADD()
00035 {
00036     return 1;
00037 }
00038
00039 //-----
```

```
00040 static uint8_t AVR_Opcode_Cycles_ADC()
00041 {
00042     return 1;
00043 }
00044
00045 //-----
00046 static uint8_t AVR_Opcode_Cycles_ADDIW()
00047 {
00048     return 2;
00049 }
00050
00051 //-----
00052 static uint8_t AVR_Opcode_Cycles_SUB()
00053 {
00054     return 1;
00055 }
00056
00057 //-----
00058 static uint8_t AVR_Opcode_Cycles_SUBI()
00059 {
00060     return 1;
00061 }
00062
00063 //-----
00064 static uint8_t AVR_Opcode_Cycles_SBC()
00065 {
00066     return 1;
00067 }
00068
00069 //-----
00070 static uint8_t AVR_Opcode_Cycles_SBCI()
00071 {
00072     return 1;
00073 }
00074
00075 //-----
00076 static uint8_t AVR_Opcode_Cycles_SBIW()
00077 {
00078     return 2;
00079 }
00080
00081 //-----
00082 static uint8_t AVR_Opcode_Cycles_AND()
00083 {
00084     return 1;
00085 }
00086
00087 //-----
00088 static uint8_t AVR_Opcode_Cycles_ANDI()
00089 {
00090     return 1;
00091 }
00092
00093 //-----
00094 static uint8_t AVR_Opcode_Cycles_OR()
00095 {
00096     return 1;
00097 }
00098
00099 //-----
00100 static uint8_t AVR_Opcode_Cycles_ORI()
00101 {
00102     return 1;
00103 }
00104
00105 //-----
00106 static uint8_t AVR_Opcode_Cycles_EOR()
00107 {
00108     return 1;
00109 }
00110
00111 //-----
00112 static uint8_t AVR_Opcode_Cycles_COM()
00113 {
00114     return 1;
00115 }
00116
00117 //-----
00118 static uint8_t AVR_Opcode_Cycles_NEG()
00119 {
00120     return 1;
00121 }
00122
00123 //-----
00124 static uint8_t AVR_Opcode_Cycles_SBR()
00125 {
00126     return 1;
```

```
00127 }
00128
00129 //-----
00130 static uint8_t AVR_Opcode_Cycles_CBR()
00131 {
00132     return 1;
00133 }
00134
00135 //-----
00136 static uint8_t AVR_Opcode_Cycles_INC()
00137 {
00138     return 1;
00139 }
00140
00141 //-----
00142 static uint8_t AVR_Opcode_Cycles_DEC()
00143 {
00144     return 1;
00145 }
00146
00147 //-----
00148 static uint8_t AVR_Opcode_Cycles_TST()
00149 {
00150     return 1;
00151 }
00152
00153 //-----
00154 static uint8_t AVR_Opcode_Cycles_CLR()
00155 {
00156     return 1;
00157 }
00158
00159 //-----
00160 static uint8_t AVR_Opcode_Cycles_SER()
00161 {
00162     return 1;
00163 }
00164
00165 //-----
00166 static uint8_t AVR_Opcode_Cycles_MUL()
00167 {
00168     return 2;
00169 }
00170
00171 //-----
00172 static uint8_t AVR_Opcode_Cycles_MULS()
00173 {
00174     return 2;
00175 }
00176
00177 //-----
00178 static uint8_t AVR_Opcode_Cycles_MULSU()
00179 {
00180     return 2;
00181 }
00182
00183 //-----
00184 static uint8_t AVR_Opcode_Cycles_FMUL()
00185 {
00186     return 2;
00187 }
00188
00189 //-----
00190 static uint8_t AVR_Opcode_Cycles_FMULS()
00191 {
00192     return 2;
00193 }
00194
00195 //-----
00196 static uint8_t AVR_Opcode_Cycles_FMULSU()
00197 {
00198     return 2;
00199 }
00200
00201 //-----
00202 static uint8_t AVR_Opcode_Cycles_DES()
00203 {
00204     return 1;
00205 }
00206
00207 //-----
00208 static uint8_t AVR_Opcode_Cycles_RJMP()
00209 {
00210     return 2;
00211 }
00212
00213 //-----
```

```
00214 static uint8_t AVR_Opcode_Cycles_IJMP()
00215 {
00216     return 2;
00217 }
00218
00219 //-----
00220 static uint8_t AVR_Opcode_Cycles_EIJMP()
00221 {
00222     return 2;
00223 }
00224
00225 //-----
00226 static uint8_t AVR_Opcode_Cycles_JMP()
00227 {
00228     return 2;
00229 }
00230
00231 //-----
00232 static uint8_t AVR_Opcode_Cycles_RCALL()
00233 {
00234     return 3;
00235 }
00236
00237 //-----
00238 static uint8_t AVR_Opcode_Cycles_ICALL()
00239 {
00240     return 3;
00241 }
00242
00243 //-----
00244 static uint8_t AVR_Opcode_Cycles_EICALL()
00245 {
00246     return 4;
00247 }
00248
00249 //-----
00250 static uint8_t AVR_Opcode_Cycles_CALL()
00251 {
00252     return 4;
00253 }
00254
00255 //-----
00256 static uint8_t AVR_Opcode_Cycles_RET()
00257 {
00258     return 4;
00259 }
00260
00261 //-----
00262 static uint8_t AVR_Opcode_Cycles_RETI()
00263 {
00264     return 4;
00265 }
00266
00267 //-----
00268 static uint8_t AVR_Opcode_Cycles_CPSE()
00269 {
00270     return 1;
00271 }
00272
00273 //-----
00274 static uint8_t AVR_Opcode_Cycles_CP()
00275 {
00276     return 1;
00277 }
00278
00279 //-----
00280 static uint8_t AVR_Opcode_Cycles_CPC()
00281 {
00282     return 1;
00283 }
00284
00285 //-----
00286 static uint8_t AVR_Opcode_Cycles_CPI()
00287 {
00288     return 1;
00289 }
00290
00291 //-----
00292 static uint8_t AVR_Opcode_Cycles_SBRCL()
00293 {
00294     return 1;
00295 }
00296
00297 //-----
00298 static uint8_t AVR_Opcode_Cycles_SBRSL()
00299 {
00300     return 1;
```



```
00301 }
00302
00303 //-----
00304 static uint8_t AVR_Opcode_Cycles_SBIC()
00305 {
00306     return 1;
00307 }
00308
00309 //-----
00310 static uint8_t AVR_Opcode_Cycles_SBIS()
00311 {
00312     return 1;
00313 }
00314
00315 //-----
00316 static uint8_t AVR_Opcode_Cycles_BRBS()
00317 {
00318     return 1;
00319 }
00320
00321 //-----
00322 static uint8_t AVR_Opcode_Cycles_BRBC()
00323 {
00324     return 1;
00325 }
00326
00327 //-----
00328 static uint8_t AVR_Opcode_Cycles_BREQ()
00329 {
00330     return 1;
00331 }
00332
00333 //-----
00334 static uint8_t AVR_Opcode_Cycles_BRNE()
00335 {
00336     return 1;
00337 }
00338
00339 //-----
00340 static uint8_t AVR_Opcode_Cycles_BRCS()
00341 {
00342     return 1;
00343 }
00344
00345 //-----
00346 static uint8_t AVR_Opcode_Cycles_BRCC()
00347 {
00348     return 1;
00349 }
00350
00351 //-----
00352 static uint8_t AVR_Opcode_Cycles_BRSH()
00353 {
00354     return 1;
00355 }
00356
00357 //-----
00358 static uint8_t AVR_Opcode_Cycles_BRL0()
00359 {
00360     return 1;
00361 }
00362
00363 //-----
00364 static uint8_t AVR_Opcode_Cycles_BRMI()
00365 {
00366     return 1;
00367 }
00368
00369 //-----
00370 static uint8_t AVR_Opcode_Cycles_BRPL()
00371 {
00372     return 1;
00373 }
00374
00375 //-----
00376 static uint8_t AVR_Opcode_Cycles_BRGE()
00377 {
00378     return 1;
00379 }
00380
00381 //-----
00382 static uint8_t AVR_Opcode_Cycles_BRLT()
00383 {
00384     return 1;
00385 }
00386
00387 //-----
```

```
00388 static uint8_t AVR_Opcode_Cycles_BRHS()
00389 {
00390     return 1;
00391 }
00392
00393 //-----
00394 static uint8_t AVR_Opcode_Cycles_BRHC()
00395 {
00396     return 1;
00397 }
00398
00399 //-----
00400 static uint8_t AVR_Opcode_Cycles_BRTS()
00401 {
00402     return 1;
00403 }
00404
00405 //-----
00406 static uint8_t AVR_Opcode_Cycles_BRTC()
00407 {
00408     return 1;
00409 }
00410
00411 //-----
00412 static uint8_t AVR_Opcode_Cycles_BRVS()
00413 {
00414     return 1;
00415 }
00416
00417 //-----
00418 static uint8_t AVR_Opcode_Cycles_BRVC()
00419 {
00420     return 1;
00421 }
00422
00423 //-----
00424 static uint8_t AVR_Opcode_Cycles_BRIE()
00425 {
00426     return 1;
00427 }
00428
00429 //-----
00430 static uint8_t AVR_Opcode_Cycles_BRID()
00431 {
00432     return 1;
00433 }
00434
00435 //-----
00436 static uint8_t AVR_Opcode_Cycles_MOV()
00437 {
00438     return 1;
00439 }
00440
00441 //-----
00442 static uint8_t AVR_Opcode_Cycles_MOVW()
00443 {
00444     return 1;
00445 }
00446
00447 //-----
00448 static uint8_t AVR_Opcode_Cycles_LDI()
00449 {
00450     return 1;
00451 }
00452
00453 //-----
00454 static uint8_t AVR_Opcode_Cycles_LDS()
00455 {
00456     return 2;
00457 }
00458
00459 //-----
00460 static uint8_t AVR_Opcode_Cycles_LD_X_Indirect()
00461 {
00462     return 1;
00463 }
00464
00465 //-----
00466 static uint8_t AVR_Opcode_Cycles_LD_X_Indirect_Postinc()
00467 {
00468     return 2;
00469 }
00470
00471 //-----
00472 static uint8_t AVR_Opcode_Cycles_LD_X_Indirect_Predec()
00473 {
00474     return 3;
```

```
00475 }
00476
00477 //-----
00478 static uint8_t AVR_Opcode_Cycles_LD_Y_Indirect()
00479 {
00480     return 1;
00481 }
00482
00483 //-----
00484 static uint8_t AVR_Opcode_Cycles_LD_Y_Indirect_Postinc()
00485 {
00486     return 2;
00487 }
00488
00489 //-----
00490 static uint8_t AVR_Opcode_Cycles_LD_Y_Indirect_Predec()
00491 {
00492     return 3;
00493 }
00494
00495 //-----
00496 static uint8_t AVR_Opcode_Cycles_LDD_Y()
00497 {
00498     return 2;
00499 }
00500
00501 //-----
00502 static uint8_t AVR_Opcode_Cycles_LD_Z_Indirect()
00503 {
00504     return 1;
00505 }
00506
00507 //-----
00508 static uint8_t AVR_Opcode_Cycles_LD_Z_Indirect_Postinc()
00509 {
00510     return 2;
00511 }
00512
00513 //-----
00514 static uint8_t AVR_Opcode_Cycles_LD_Z_Indirect_Predec()
00515 {
00516     return 3;
00517 }
00518
00519 //-----
00520 static uint8_t AVR_Opcode_Cycles_LDD_Z()
00521 {
00522     return 2;
00523 }
00524
00525 //-----
00526 static uint8_t AVR_Opcode_Cycles_STS()
00527 {
00528     return 2;
00529 }
00530
00531 //-----
00532 static uint8_t AVR_Opcode_Cycles_ST_X_Indirect()
00533 {
00534     return 2;
00535 }
00536
00537 //-----
00538 static uint8_t AVR_Opcode_Cycles_ST_X_Indirect_Postinc()
00539 {
00540     return 2;
00541 }
00542
00543 //-----
00544 static uint8_t AVR_Opcode_Cycles_ST_X_Indirect_Predec()
00545 {
00546     return 2;
00547 }
00548
00549 //-----
00550 static uint8_t AVR_Opcode_Cycles_ST_Y_Indirect()
00551 {
00552     return 2;
00553 }
00554
00555 //-----
00556 static uint8_t AVR_Opcode_Cycles_ST_Y_Indirect_Postinc()
00557 {
00558     return 2;
00559 }
00560
00561 //-----
```

```

00562 static uint8_t AVR_Opcode_Cycles_ST_Y_Indirect_Predc()
00563 {
00564     return 2;
00565 }
00566
00567 //-----
00568 static uint8_t AVR_Opcode_Cycles_STD_Y()
00569 {
00570     return 2;
00571 }
00572
00573 //-----
00574 static uint8_t AVR_Opcode_Cycles_ST_Z_Indirect()
00575 {
00576     return 2;
00577 }
00578
00579 //-----
00580 static uint8_t AVR_Opcode_Cycles_ST_Z_Indirect_Postinc()
00581 {
00582     return 2;
00583 }
00584
00585 //-----
00586 static uint8_t AVR_Opcode_Cycles_ST_Z_Indirect_Predc()
00587 {
00588     return 2;
00589 }
00590
00591 //-----
00592 static uint8_t AVR_Opcode_Cycles_STD_Z()
00593 {
00594     return 2;
00595 }
00596
00597 //-----
00598 static uint8_t AVR_Opcode_Cycles_LPM()
00599 {
00600     return 3;
00601 }
00602
00603 //-----
00604 static uint8_t AVR_Opcode_Cycles_LPM_Z()
00605 {
00606     return 3;
00607 }
00608
00609 //-----
00610 static uint8_t AVR_Opcode_Cycles_LPM_Z_Postinc()
00611 {
00612     return 3;
00613 }
00614
00615 //-----
00616 static uint8_t AVR_Opcode_Cycles_ELPM()
00617 {
00618     return 3;
00619 }
00620
00621 //-----
00622 static uint8_t AVR_Opcode_Cycles_ELPM_Z()
00623 {
00624     return 3;
00625 }
00626
00627 //-----
00628 static uint8_t AVR_Opcode_Cycles_ELPM_Z_Postinc()
00629 {
00630     return 3;
00631 }
00632
00633 //-----
00634 static uint8_t AVR_Opcode_Cycles_SPM()
00635 {
00636     return 2;
00637 }
00638
00639 //-----
00640 static uint8_t AVR_Opcode_Cycles_SPM_Z_Postinc2()
00641 {
00642     return 2;
00643 }
00644
00645 //-----
00646 static uint8_t AVR_Opcode_Cycles_IN()
00647 {
00648     return 1;

```

```
00649 }
00650
00651 //-----
00652 static uint8_t AVR_Opcode_Cycles_OUT()
00653 {
00654     return 1;
00655 }
00656
00657 //-----
00658 static uint8_t AVR_Opcode_Cycles_LAC()
00659 {
00660     return 1;
00661 }
00662
00663 //-----
00664 static uint8_t AVR_Opcode_Cycles_LAS()
00665 {
00666     return 1;
00667 }
00668
00669 //-----
00670 static uint8_t AVR_Opcode_Cycles_LAT()
00671 {
00672     return 1;
00673 }
00674
00675 //-----
00676 static uint8_t AVR_Opcode_Cycles_LSL()
00677 {
00678     return 1;
00679 }
00680
00681 //-----
00682 static uint8_t AVR_Opcode_Cycles_LSR()
00683 {
00684     return 1;
00685 }
00686
00687 //-----
00688 static uint8_t AVR_Opcode_Cycles_POP()
00689 {
00690     return 2;
00691 }
00692
00693 //-----
00694 static uint8_t AVR_Opcode_Cycles_PUSH()
00695 {
00696     return 2;
00697 }
00698
00699 //-----
00700 static uint8_t AVR_Opcode_Cycles_ROL()
00701 {
00702     return 1;
00703 }
00704
00705 //-----
00706 static uint8_t AVR_Opcode_Cycles_ROR()
00707 {
00708     return 1;
00709 }
00710
00711 //-----
00712 static uint8_t AVR_Opcode_Cycles_ASR()
00713 {
00714     return 1;
00715 }
00716
00717 //-----
00718 static uint8_t AVR_Opcode_Cycles_SWAP()
00719 {
00720     return 1;
00721 }
00722
00723 //-----
00724 static uint8_t AVR_Opcode_Cycles_BSET()
00725 {
00726     return 1;
00727 }
00728
00729 //-----
00730 static uint8_t AVR_Opcode_Cycles_BCLR()
00731 {
00732     return 1;
00733 }
00734
00735 //-----
```

```
00736 static uint8_t AVR_Opcode_Cycles_SBI()
00737 {
00738     return 2;
00739 }
00740
00741 //-----
00742 static uint8_t AVR_Opcode_Cycles_CBI()
00743 {
00744     return 2;
00745 }
00746
00747 //-----
00748 static uint8_t AVR_Opcode_Cycles_BST()
00749 {
00750     return 1;
00751 }
00752
00753 //-----
00754 static uint8_t AVR_Opcode_Cycles_BLD()
00755 {
00756     return 1;
00757 }
00758
00759 //-----
00760 static uint8_t AVR_Opcode_Cycles_SEC()
00761 {
00762     return 1;
00763 }
00764
00765 //-----
00766 static uint8_t AVR_Opcode_Cycles_CLC()
00767 {
00768     return 1;
00769 }
00770
00771 //-----
00772 static uint8_t AVR_Opcode_Cycles_SEN()
00773 {
00774     return 1;
00775 }
00776
00777 //-----
00778 static uint8_t AVR_Opcode_Cycles_CLN()
00779 {
00780     return 1;
00781 }
00782
00783 //-----
00784 static uint8_t AVR_Opcode_Cycles_SEZ()
00785 {
00786     return 1;
00787 }
00788
00789 //-----
00790 static uint8_t AVR_Opcode_Cycles_CLZ()
00791 {
00792     return 1;
00793 }
00794
00795 //-----
00796 static uint8_t AVR_Opcode_Cycles_SEI()
00797 {
00798     return 1;
00799 }
00800
00801 //-----
00802 static uint8_t AVR_Opcode_Cycles_CLI()
00803 {
00804     return 1;
00805 }
00806
00807 //-----
00808 static uint8_t AVR_Opcode_Cycles_SES()
00809 {
00810     return 1;
00811 }
00812
00813 //-----
00814 static uint8_t AVR_Opcode_Cycles_CLS()
00815 {
00816     return 1;
00817 }
00818
00819 //-----
00820 static uint8_t AVR_Opcode_Cycles_SEV()
00821 {
00822     return 1;
```

```

00823 }
00824
00825 //-----
00826 static uint8_t AVR_Opcode_Cycles_CLV()
00827 {
00828     return 1;
00829 }
00830
00831 //-----
00832 static uint8_t AVR_Opcode_Cycles_SET()
00833 {
00834     return 1;
00835 }
00836
00837 //-----
00838 static uint8_t AVR_Opcode_Cycles_CLT()
00839 {
00840     return 1;
00841 }
00842
00843 //-----
00844 static uint8_t AVR_Opcode_Cycles_SEH()
00845 {
00846     return 1;
00847 }
00848
00849 //-----
00850 static uint8_t AVR_Opcode_Cycles_CLH()
00851 {
00852     return 1;
00853 }
00854
00855 //-----
00856 static uint8_t AVR_Opcode_Cycles_BREAK()
00857 {
00858     return 1;
00859 }
00860
00861 //-----
00862 static uint8_t AVR_Opcode_Cycles_NOP()
00863 {
00864     return 1;
00865 }
00866
00867 //-----
00868 static uint8_t AVR_Opcode_Cycles_SLEEP()
00869 {
00870     return 1;
00871 }
00872
00873 //-----
00874 static uint8_t AVR_Opcode_Cycles_WDR()
00875 {
00876     return 1;
00877 }
00878
00879 //-----
00880 static uint8_t AVR_Opcode_Cycles_XCH()
00881 {
00882     return 1;
00883 }
00884
00885 //-----
00886 static uint8_t AVR_Opcode_Cycles_Unimplemented()
00887 {
00888     return 1;
00889 }
00890
00891 //-----
00892 uint8_t AVR_Opcode_Cycles( uint16_t OP_ )
00893 {
00894     // Special instructions - "static" encoding
00895     switch (OP_)
00896     {
00897         case 0x0000: return AVR_Opcode_Cycles_NOP();
00898
00899         case 0x9408: return AVR_Opcode_Cycles_SEC();
00900         case 0x9409: return AVR_Opcode_Cycles_IJMP();
00901         case 0x9418: return AVR_Opcode_Cycles_SEZ();
00902         case 0x9419: return AVR_Opcode_Cycles_EIJMP();
00903         case 0x9428: return AVR_Opcode_Cycles_SEN();
00904         case 0x9438: return AVR_Opcode_Cycles_SEV();
00905         case 0x9448: return AVR_Opcode_Cycles_SES();
00906         case 0x9458: return AVR_Opcode_Cycles_SEH();
00907         case 0x9468: return AVR_Opcode_Cycles_SET();
00908         case 0x9478: return AVR_Opcode_Cycles_SEI();
00909

```

```

00910     case 0x9488: return AVR_Opcode_Cycles_CLC();
00911     case 0x9498: return AVR_Opcode_Cycles_CLZ();
00912     case 0x94A8: return AVR_Opcode_Cycles_CLN();
00913     case 0x94B8: return AVR_Opcode_Cycles_CLV();
00914     case 0x94C8: return AVR_Opcode_Cycles_CLS();
00915     case 0x94D8: return AVR_Opcode_Cycles_CLH();
00916     case 0x94E8: return AVR_Opcode_Cycles_CLT();
00917     case 0x94F8: return AVR_Opcode_Cycles_CLI();
00918
00919     case 0x9508: return AVR_Opcode_Cycles_RET();
00920     case 0x9509: return AVR_Opcode_Cycles_ICALL();
00921     case 0x9518: return AVR_Opcode_Cycles_RETI();
00922     case 0x9519: return AVR_Opcode_Cycles_EICALL();
00923     case 0x9588: return AVR_Opcode_Cycles_SLEEP();
00924     case 0x9598: return AVR_Opcode_Cycles_BREAK();
00925     case 0x95A8: return AVR_Opcode_Cycles_WDR();
00926     case 0x95C8: return AVR_Opcode_Cycles_LPM();
00927     case 0x95D8: return AVR_Opcode_Cycles_ELPM();
00928     case 0x95E8: return AVR_Opcode_Cycles_SPM();
00929     case 0x95F8: return AVR_Opcode_Cycles_SPM_Z_Postinc2();
00930 }
00931
00932 #if 0
00933 // Note: These disasm handlers are generalized versions of specific mnemonics in the above list.
00934 // For disassembly, it's probably easier to read the output from the more "specific" mnemonics, so
00935 // those are used. For emulation, using the generalized functions may be more desirable.
00936 switch( OP_ & 0xFF8F)
00937 {
00938     case 0x9408: return AVR_Opcode_Cycles_BSET();
00939     case 0x9488: return AVR_Opcode_Cycles_BCLR();
00940 }
00941 #endif
00942
00943 switch( OP_ & 0xFF88)
00944 {
00945     case 0x0300: return AVR_Opcode_Cycles_MULSU();
00946     case 0x0308: return AVR_Opcode_Cycles_FMUL();
00947     case 0x0380: return AVR_Opcode_Cycles_FMULS();
00948     case 0x0388: return AVR_Opcode_Cycles_FMULSU();
00949 }
00950
00951 switch( OP_ & 0xFF0F)
00952 {
00953     case 0x940B: return AVR_Opcode_Cycles_DES();
00954     case 0xEF0F: return AVR_Opcode_Cycles_SER();
00955 }
00956
00957 switch( OP_ & 0xFF00)
00958 {
00959     case 0x0100: return AVR_Opcode_Cycles_MOVW();
00960     case 0x9600: return AVR_Opcode_Cycles_ADIW();
00961     case 0x9700: return AVR_Opcode_Cycles_SBIW();
00962
00963     case 0x9800: return AVR_Opcode_Cycles_CBI();
00964     case 0x9900: return AVR_Opcode_Cycles_SBIC();
00965     case 0x9A00: return AVR_Opcode_Cycles_SBI();
00966     case 0x9B00: return AVR_Opcode_Cycles_SBIS();
00967 }
00968
00969 switch( OP_ & 0xFE0F)
00970 {
00971     case 0x8008: return AVR_Opcode_Cycles_LD_Y_Indirect();
00972     case 0x8000: return AVR_Opcode_Cycles_LD_Z_Indirect();
00973     case 0x8200: return AVR_Opcode_Cycles_ST_Z_Indirect();
00974     case 0x8208: return AVR_Opcode_Cycles_ST_Y_Indirect();
00975
00976     // -- Single 5-bit register...
00977     case 0x9000: return AVR_Opcode_Cycles_LDS();
00978     case 0x9001: return AVR_Opcode_Cycles_LD_Z_Indirect_Postinc();
00979     case 0x9002: return AVR_Opcode_Cycles_LD_Z_Indirect_Predec();
00980     case 0x9004: return AVR_Opcode_Cycles_LPM_Z();
00981     case 0x9005: return AVR_Opcode_Cycles_LPM_Z_Postinc();
00982     case 0x9006: return AVR_Opcode_Cycles_ELPM_Z();
00983     case 0x9007: return AVR_Opcode_Cycles_ELPM_Z_Postinc();
00984     case 0x9009: return AVR_Opcode_Cycles_LD_Y_Indirect_Postinc();
00985     case 0x900A: return AVR_Opcode_Cycles_LD_Y_Indirect_Predec();
00986     case 0x900C: return AVR_Opcode_Cycles_LD_X_Indirect();
00987     case 0x900D: return AVR_Opcode_Cycles_LD_X_Indirect_Postinc();
00988     case 0x900E: return AVR_Opcode_Cycles_LD_X_Indirect_Predec();
00989     case 0x900F: return AVR_Opcode_Cycles_POP();
00990
00991     case 0x9200: return AVR_Opcode_Cycles_STS();
00992     case 0x9201: return AVR_Opcode_Cycles_ST_Z_Indirect_Postinc();
00993     case 0x9202: return AVR_Opcode_Cycles_ST_Z_Indirect_Predec();
00994     case 0x9204: return AVR_Opcode_Cycles_XCH();
00995     case 0x9205: return AVR_Opcode_Cycles_LAS();
00996     case 0x9206: return AVR_Opcode_Cycles_LAC();

```



```

00997     case 0x9207: return AVR_Opcode_Cycles_LAT();
00998     case 0x9209: return AVR_Opcode_Cycles_ST_Y_Indirect_Postinc();
00999     case 0x920A: return AVR_Opcode_Cycles_ST_Y_Indirect_Preddec();
01000     case 0x920C: return AVR_Opcode_Cycles_ST_X_Indirect();
01001     case 0x920D: return AVR_Opcode_Cycles_ST_X_Indirect_Postinc();
01002     case 0x920E: return AVR_Opcode_Cycles_ST_X_Indirect_Preddec();
01003     case 0x920F: return AVR_Opcode_Cycles_PUSH();
01004
01005     // -- One-operand instructions
01006     case 0x9400: return AVR_Opcode_Cycles_COM();
01007     case 0x9401: return AVR_Opcode_Cycles_NEG();
01008     case 0x9402: return AVR_Opcode_Cycles_SWAP();
01009     case 0x9403: return AVR_Opcode_Cycles_INC();
01010     case 0x9405: return AVR_Opcode_Cycles_ASR();
01011     case 0x9406: return AVR_Opcode_Cycles_LSR();
01012     case 0x9407: return AVR_Opcode_Cycles_ROR();
01013     case 0x940A: return AVR_Opcode_Cycles_DEC();
01014
01015     }
01016     switch (OP_ & 0xFE0E)
01017     {
01018     case 0x940C: return AVR_Opcode_Cycles_JMP();
01019     case 0x940E: return AVR_Opcode_Cycles_CALL();
01020     }
01021
01022     switch (OP_ & 0xFE08)
01023     {
01024
01025     // -- BLD/BST Encoding
01026     case 0xF800: return AVR_Opcode_Cycles_BLD();
01027     case 0xFA00: return AVR_Opcode_Cycles_BST();
01028     // -- SBRC/SBRS Encoding
01029     case 0xFC00: return AVR_Opcode_Cycles_SBRC();
01030     case 0xFE00: return AVR_Opcode_Cycles_SBRS();
01031     }
01032
01033     switch (OP_ & 0xFC07)
01034     {
01035     // -- Conditional branches
01036     case 0xF000: return AVR_Opcode_Cycles_BRCS();
01037     // case 0xF000: return AVR_Opcode_Cycles_BRLO(); // AKA AVR_Opcode_Cycles_BRCS();
01038     case 0xF001: return AVR_Opcode_Cycles_BREQ();
01039     case 0xF002: return AVR_Opcode_Cycles_BRMI();
01040     case 0xF003: return AVR_Opcode_Cycles_BRVS();
01041     case 0xF004: return AVR_Opcode_Cycles_BRLT();
01042     case 0xF006: return AVR_Opcode_Cycles_BRTS();
01043     case 0xF007: return AVR_Opcode_Cycles_BRIE();
01044     case 0xF400: return AVR_Opcode_Cycles_BRCC();
01045     // case 0xF400: return AVR_Opcode_Cycles_BRSH(); // AKA AVR_Opcode_Cycles_BRCC();
01046     case 0xF401: return AVR_Opcode_Cycles_BRNE();
01047     case 0xF402: return AVR_Opcode_Cycles_BRPL();
01048     case 0xF403: return AVR_Opcode_Cycles_BRVC();
01049     case 0xF404: return AVR_Opcode_Cycles_BRGE();
01050     case 0xF405: return AVR_Opcode_Cycles_BRHC();
01051     case 0xF406: return AVR_Opcode_Cycles_BRTC();
01052     case 0xF407: return AVR_Opcode_Cycles_BRID();
01053     }
01054
01055     switch (OP_ & 0xFC00)
01056     {
01057     // -- 4-bit register pair
01058     case 0x0200: return AVR_Opcode_Cycles_MULS();
01059
01060     // -- 5-bit register pairs --
01061     case 0x0400: return AVR_Opcode_Cycles_CPC();
01062     case 0x0800: return AVR_Opcode_Cycles_SBC();
01063     case 0x0C00: return AVR_Opcode_Cycles_ADD();
01064     // case 0x0C00: return AVR_Opcode_Cycles_LSL(); (!! Implemented with: " add rd, rd"
01065     case 0x1000: return AVR_Opcode_Cycles_CPSE();
01066     case 0x1300: return AVR_Opcode_Cycles_ROL();
01067     case 0x1400: return AVR_Opcode_Cycles_CP();
01068     case 0x1C00: return AVR_Opcode_Cycles_ADC();
01069     case 0x1800: return AVR_Opcode_Cycles_SUB();
01070     case 0x2000: return AVR_Opcode_Cycles_AND();
01071     // case 0x2000: return AVR_Opcode_Cycles_TST(); (!! Implemented with: " and rd, rd"
01072     case 0x2400: return AVR_Opcode_Cycles_EOR();
01073     case 0x2C00: return AVR_Opcode_Cycles_MOV();
01074     case 0x2800: return AVR_Opcode_Cycles_OR();
01075
01076     // -- 5-bit register pairs -- Destination = R1:R0
01077     case 0x9C00: return AVR_Opcode_Cycles_MUL();
01078     }
01079
01080     switch (OP_ & 0xF800)
01081     {
01082     case 0xB800: return AVR_Opcode_Cycles_OUT();
01083     case 0xB000: return AVR_Opcode_Cycles_IN();

```

```

01084     }
01085
01086     switch (OP_ & 0xF000)
01087     {
01088         // -- Register immediate --
01089         case 0x3000: return AVR_Opcode_Cycles_CPI();
01090         case 0x4000: return AVR_Opcode_Cycles_SBCI();
01091         case 0x5000: return AVR_Opcode_Cycles_SUBI();
01092         case 0x6000: return AVR_Opcode_Cycles_ORI();
01093         case 0x7000: return AVR_Opcode_Cycles_ANDI();
01094
01095         //-- 12-bit immediate
01096         case 0xC000: return AVR_Opcode_Cycles_RJMP();
01097         case 0xD000: return AVR_Opcode_Cycles_RCALL();
01098
01099         // -- Register immediate
01100         case 0xE000: return AVR_Opcode_Cycles_LDI();
01101     }
01102
01103     switch (OP_ & 0xD208)
01104     {
01105         // -- 7-bit signed offset
01106         case 0x8000: return AVR_Opcode_Cycles_LDD_Z();
01107         case 0x8008: return AVR_Opcode_Cycles_LDD_Y();
01108         case 0x8200: return AVR_Opcode_Cycles_STD_Z();
01109         case 0x8208: return AVR_Opcode_Cycles_STD_Y();
01110     }
01111
01112     return AVR_Opcode_Cycles_Unimplemented();
01113 }
01114

```

## 4.25 src/avr\_cpu/avr\_op\_cycles.h File Reference

Opcode cycle counting functions.

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

### Functions

- `uint8_t AVR_Opcode_Cycles (uint16_t OP_)`  
*AVR\_Opcode\_Cycles.*

#### 4.25.1 Detailed Description

Opcode cycle counting functions.

Definition in file [avr\\_op\\_cycles.h](#).

#### 4.25.2 Function Documentation

##### 4.25.2.1 AVR\_Opcode\_Cycles()

```
uint8_t AVR_Opcode_Cycles (
    uint16_t OP_ )
```

*AVR\_Opcode\_Cycles.*

## Parameters

$O \leftarrow$	Opcode to compute the minimum cycles to execute for
$P \leftarrow$	
$-$	

## Returns

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

Definition at line 892 of file [avr\\_op\\_cycles.c](#).

## 4.26 avr\_op\_cycles.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ )  )\ )      (      |
00004 *      ((/( ((/(      \      ( ((/(      -- [ Funkenstein ] -----
00005 *      /( )  /( ) ((( ( ) \      / ( )      -- [ Little ] -----
00006 *      ( ) _ ( )      ) \ _ ) \ ( ( ) ( ( )      -- [ AVR ] -----
00007 *      | _ | | |      ( ) _ ( ) \ \ / / | _ \      -- [ Virtual ] -----
00008 *      | _ | | |      / _ \ \ \ / / | _ \      -- [ Runtime ] -----
00009 *      | _ | | |      / _ \ \ \ / / | _ \      |
00010 *      | _ | | |      / _ \ \ \ / / | _ \      | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __AVR_OP_CYCLES_H__
00022 #define __AVR_OP_CYCLES_H__
00023
00024 #include <stdint.h>
00025
00026 //-----
00032 uint8_t AVR_Opcode_Cycles( uint16_t OP_ );
00033
00034 #endif

```

## 4.27 src/avr\_cpu/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\_SBRB** (uint16\_t OP\_)
  - AVR\_Decoder [AVR\\_Decoder\\_Function](#) (uint16\_t OP\_)
- AVR\_Decoder\_Function.*
- void [AVR\\_Decode](#) (uint16\_t OP\_)
- AVR\_Decode.*

#### 4.27.1 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](#).

#### 4.27.2 Function Documentation

##### 4.27.2.1 AVR\_Decode()

```
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

$O \leftarrow$	Opcode to decode
$P \leftarrow$	
—	

Definition at line 400 of file [avr\\_op\\_decode.c](#).

#### 4.27.2.2 AVR\_Decoder\_Function()

```
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

$O \leftrightarrow$ $P \leftrightarrow$ —	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 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

!MOS Verified

Definition at line 251 of file [avr\\_op\\_decode.c](#).

## 4.28 avr\_op\_decode.c

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( ((/(      (      | -- [ Funkenstein ] -----
00005 *      /(_ ) /(_ ) ((((_ )\ )\ /(_ ) | -- [ Little ] -----
00006 *      (_ )| (_ )      )\ _ )\ (( _ )| | -- [ AVR ] -----
00007 *      | | _ | |      (_ )\(_ )\ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ | / _ \ \ \ V / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ | / _ \ \ \ \ / | _ \ |
00010 *      | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00024 #include <stdint.h>
00025
00026 #include "emu_config.h"
00027
00028 #include "avr_op_decode.h"
00029
00030 //-----
00031 static void AVR_Decoder_NOP( uint16_t OP_ )
00032 {
00033     // Nothing to do here...
00034 }
00035 //-----
00036 static void AVR_Decoder_Register_Pair_4bit( uint16_t OP_ )
00037 {
00038     uint8_t Rr = (OP_ & 0x000F);
00039     uint8_t Rd = ((OP_ & 0x00F0) >> 4);
00040
00041     stCPU.Rr16 = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r_word[Rr]);
00042     stCPU.Rd16 = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r_word[Rd]);
00043 }
00044 //-----
00045 static void AVR_Decoder_Register_Pair_3bit( uint16_t OP_ )
00046 {
00047     uint8_t Rr = (OP_ & 0x0007) + 16;
00048     uint8_t Rd = ((OP_ & 0x0070) >> 4) + 16;
00049
00050     stCPU.Rr = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rr]);
00051     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00052 }
00053 //-----
00054 static void AVR_Decoder_Register_Pair_5bit( uint16_t OP_ )
00055 {
00056     uint8_t Rr = (OP_ & 0x000F) | ((OP_ & 0x0200) >> 5);
00057     uint8_t Rd = (OP_ & 0x01F0) >> 4;
00058
00059     stCPU.Rr = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rr]);
00060     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00061 }
00062 //-----
00063 static void AVR_Decoder_Register_Immediate( uint16_t OP_ )
00064 {
00065     uint8_t K = (OP_ & 0x000F) | ((OP_ & 0x0F00) >> 4);
00066     uint8_t Rd = ((OP_ & 0x00F0) >> 4) + 16;
00067
00068     stCPU.K = K;
00069     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00070 }
00071 //-----
00072 static void AVR_Decoder_LDST_YZ_k( uint16_t OP_ )
00073 {
00074     uint8_t q = (OP_ & 0x0007) | ((OP_ & 0x0C00) >> (7)) | // Awkward encoding... see manual for details.
00075                ((OP_ & 0x2000) >> (8));
00076
00077     uint8_t Rd = (OP_ & 0x01F0) >> 4;
00078
00079     stCPU.q = q;
00080     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00081 }
00082 //-----
00083 static void AVR_Decoder_LDST( uint16_t OP_ )
00084 {
00085     uint8_t Rd = (OP_ & 0x01F0) >> 4;
00086
00087     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00088 }
00089 //-----
00090 static void AVR_Decoder_LDS_STS( uint16_t OP_ )
00091 {
00092     uint8_t Rd = (OP_ & 0x01F0) >> 4;

```

```

00094
00095     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00096     stCPU.K = stCPU.pu16ROM[ stCPU.u32PC + 1 ];
00097 }
00098 //-----
00099 static void AVR_Decoder_Register_Single( uint16_t OP_)
00100 {
00101     uint8_t Rd = (OP_ & 0x01F0) >> 4;
00102
00103     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00104 }
00105 //-----
00106 static void AVR_Decoder_Register_SC( uint16_t OP_)
00107 {
00108     uint8_t b = (OP_ & 0x0070) >> 4;
00109
00110     stCPU.b = b;
00111 }
00112 //-----
00113 static void AVR_Decoder_Misc( uint16_t OP_)
00114 {
00115     // Nothing to do here.
00116 }
00117 //-----
00118 static void AVR_Decoder_Indirect_Jump( uint16_t OP_)
00119 {
00120     // Nothing to do here.
00121 }
00122 //-----
00123 static void AVR_Decoder_DEC_Rd( uint16_t OP_)
00124 {
00125     uint8_t Rd = (OP_ & 0x01F0) >> 4;
00126
00127     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00128 }
00129 //-----
00130 static void AVR_Decoder_DES_round_4( uint16_t OP_)
00131 {
00132     uint8_t K = (OP_ & 0x00F0) >> 4;
00133     stCPU.K = K;
00134 }
00135 //-----
00136 static void AVR_Decoder_JMP_CALL_22( uint16_t OP_)
00137 {
00138     uint16_t op = stCPU.pu16ROM[ stCPU.u32PC + 1 ];
00139     uint32_t k = op;
00140     k |= (((OP_ & 0x0001) | (OP_ & 0x01F0) >> 3) << 16);
00141
00142     stCPU.k = k;
00143
00144     // These are 2-cycle instructions.  Clock the CPU here, since we're fetching
00145     // the second word of data for this opcode here.
00146     IO_Clock();
00147 }
00148 //-----
00149 static void AVR_Decoder_ADIW_SBIW_6( uint16_t OP_)
00150 {
00151     uint8_t K = (OP_ & 0x000F) | ((OP_ & 0x00C0) >> 2);
00152     uint8_t Rd16 = (((OP_ & 0x0030) >> 4) * 2) + 24;
00153
00154     stCPU.K = K;
00155     stCPU.Rd16 = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r_word[Rd16 >> 1]);
00156 }
00157 //-----
00158 static void AVR_Decoder_IO_Bit( uint16_t OP_)
00159 {
00160     uint8_t b = (OP_ & 0x0007);
00161     uint8_t A = (OP_ & 0x00F8) >> 3;
00162
00163     stCPU.b = b;
00164     stCPU.A = A;
00165 }
00166 //-----
00167 static void AVR_Decoder_MUL( uint16_t OP_)
00168 {
00169     uint8_t Rr = (OP_ & 0x000F) | ((OP_ & 0x0200) >> 5);
00170     uint8_t Rd = (OP_ & 0x01F0) >> 4;
00171
00172     stCPU.Rr = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rr]);
00173     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00174 }
00175 //-----
00176 static void AVR_Decoder_IO_In_Out( uint16_t OP_)
00177 {
00178     uint8_t A = (OP_ & 0x000F) | ((OP_ & 0x0600) >> 5);
00179     uint8_t Rd = (OP_ & 0x01F0) >> 4;
00180

```

```

00181     stCPU.A = A;
00182     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00183 }
00184 //-----
00185 static void AVR_Decoder_Relative_Jump( uint16_t OP_ )
00186 {
00187     // NB: -2K <= k <= 2K
00188     uint16_t k = (OP_ & 0xFFFF);
00189
00190     // Check for sign bit in 12-bit value...
00191     if (k & 0x0800)
00192     {
00193         stCPU.k_s = (int32_t)((~k & 0x07FF) + 1) * -1;
00194     }
00195     else
00196     {
00197         stCPU.k_s = (int32_t)k;
00198     }
00199 }
00200 //-----
00201 static void AVR_Decoder_LDI( uint16_t OP_ )
00202 {
00203     uint8_t K = (OP_ & 0x000F) | ((OP_ & 0x0F00) >> 4);
00204     uint8_t Rd = ((OP_ & 0x00F0) >> 4) + 16;
00205
00206     stCPU.K = K;
00207     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00208 }
00209 //-----
00210 static void AVR_Decoder_Conditional_Branch( uint16_t OP_ )
00211 {
00212     // NB: -64 <= k <= 63
00213     uint8_t b = (OP_ & 0x0007);
00214     uint8_t k = ((OP_ & 0x03F8) >> 3);
00215
00216     stCPU.b = b;
00217
00218     // Check for sign bit in 7-bit value...
00219     if (k & 0x40)
00220     {
00221         // Convert to signed 32-bit integer... probably a cleaner way
00222         // of doing this, but I'm tired.
00223         stCPU.k_s = (int32_t)((~k & 0x3F) + 1) * -1;
00224     }
00225     else
00226     {
00227         stCPU.k_s = (int32_t)k;
00228     }
00229 }
00230 //-----
00231 static void AVR_Decoder_BLD_BST( uint16_t OP_ )
00232 {
00233     uint8_t b = (OP_ & 0x0007);
00234     uint8_t Rd = ((OP_ & 0x01F0) >> 4);
00235
00236     stCPU.b = b;
00237     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00238 }
00239 //-----
00240 static void AVR_Decoder_SBRC_SBRs( uint16_t OP_ )
00241 {
00242     uint8_t b = (OP_ & 0x0007);
00243     uint8_t Rd = ((OP_ & 0x01F0) >> 4);
00244
00245     stCPU.b = b;
00246     stCPU.Rd = &(stCPU.pstRAM->stRegisters.CORE_REGISTERS.r[Rd]);
00247 }
00248 //-----
00251 AVR_Decoder AVR_Decoder_Function( uint16_t OP_ )
00252 {
00253     if ((OP_ & 0xFF0F) == 0x9408 )
00254     {
00255         // SEx/CLx status register clear/set bit.
00256         return AVR_Decoder_Register_SC;
00257     }
00258     else if ((OP_ & 0xFF0F) == 0x9508 )
00259     {
00260         // Miscellaneous instruction
00261         return AVR_Decoder_Misc;
00262     }
00263     else if ((OP_ & 0xFF0F) == 0x940B )
00264     {
00265         // Des round k
00266         return AVR_Decoder_DES_round_4;
00267     }
00270 }

```



```

00271     else if ( (( OP_ & 0xFF00 ) == 0x0100 ) ||
00272               (( OP_ & 0xFF00 ) == 0x0200 ) )
00273     {
00275         // Register pair 4bit (MOVW, MULS)
00276         return AVR_Decoder_Register_Pair_4bit;
00277     }
00278     else if ( ( OP_ & 0xFF00 ) == 0x0300 )
00279     {
00281         // 3-bit register pair (R16->R23) - (FMUL, FMULS, FMULSU, MULSU)
00282         return AVR_Decoder_Register_Pair_3bit;
00283     }
00284     else if ( ( OP_ & 0xFF00 ) <= 0x2F00 )
00285     {
00286         // Register pair 5bit
00287         return AVR_Decoder_Register_Pair_5bit;
00288     }
00289     else if ( ( OP_ & 0xFF00 ) <= 0x7F00 )
00290     {
00292         // Register immediate
00293         return AVR_Decoder_Register_Immediate;
00294     }
00295     else if ( ( OP_ & 0xFEEF ) == 0x9409 )
00296     {
00298         // Indirect Jump/call
00299         return AVR_Decoder_Indirect_Jump;
00300     }
00301     else if ( ( OP_ & 0xFE08 ) == 0x9400 )
00302     {
00304         // 1-operand instructions.
00305         return AVR_Decoder_Register_Single;
00306     }
00307     else if ( ( OP_ & 0xFE0F ) == 0x940A )
00308     {
00310         // Dec Rd
00311         return AVR_Decoder_DEC_Rd;
00312     }
00313     else if ( ( OP_ & 0xFE0C ) == 0x940C )
00314     {
00316         // Jmp/call abs22
00317         return AVR_Decoder_JMP_CALL_22;
00318     }
00319     else if ( ( OP_ & 0xFE00 ) == 0x9600 )
00320     {
00322         // ADIW/SBIW Rp
00323         return AVR_Decoder_ADIW_SBIW_6;
00324     }
00325     else if ( ( OP_ & 0xFC0F ) == 0x9000 )
00326     {
00328         // LDS/STS
00329         return AVR_Decoder_LDS_STS;
00330     }
00331     else if ( ( OP_ & 0xFC00 ) == 0x9000 )
00332     {
00334         // LD/ST other
00335         return AVR_Decoder_LDST;
00336     }
00337     else if ( ( OP_ & 0xFC00 ) == 0x9800 )
00338     {
00340         // IO Space bit operations
00341         return AVR_Decoder_IO_Bit;
00342     }
00343     else if ( ( OP_ & 0xFC00 ) == 0x9C00 )
00344     {
00346         // MUL unsigned R1:R0 = Rr x Rd
00347         return AVR_Decoder_MUL;
00348     }
00349     else if ( ( OP_ & 0xFC00 ) == 0xF800 )
00350     {
00352         // BLD/BST register bit to STATUS.T
00353         return AVR_Decoder_BLD_BST;
00354     }
00355     else if ( ( OP_ & 0xFC00 ) == 0xFC00 )
00356     {
00358         // SBRC/SBRS
00359         return AVR_Decoder_SBRC_SBRS;
00360     }
00361     else if ( ( OP_ & 0xF800 ) == 0xF000 )
00362     {
00364         // Conditional branch
00365         return AVR_Decoder_Conditional_Branch;
00366     }
00367     else if ( ( OP_ & 0xF000 ) == 0xE000 )
00368     {
00370         // LDI Rh, K
00371         return AVR_Decoder_LDI;
00372     }
00373     else if ( ( OP_ & 0xF000 ) == 0xB000 )

```

```

00374     {
00375         // IO space IN/OUT operations
00376         return AVR_Decoder_IO_In_Out;
00377     }
00378 }
00379 else if (( OP_ & 0xE000) == 0xC000 )
00380 {
00381     // Relative Jump/Call
00382     return AVR_Decoder_Relative_Jump;
00383 }
00384 }
00385 else if (( OP_ & 0xD000) == 0x8000 )
00386 {
00387     // LDD/STD to Z+kY+k
00388     return AVR_Decoder_LDST_YZ_k;
00389 }
00390 }
00391 else if ( OP_ == 0 )
00392 {
00393     return AVR_Decoder_NOP;
00394 }
00395 }
00396 return AVR_Decoder_NOP;
00397 }
00398
00399 //-----
00400 void AVR_Decode( uint16_t OP_ )
00401 {
00402     AVR_Decoder myDecoder;
00403     myDecoder = AVR_Decoder_Function(OP_);
00404     myDecoder( OP_ );
00405 }

```

## 4.29 src/avr\_cpu/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.*

#### 4.29.1 Detailed Description

Module providing logic to decode AVR CPU Opcodes.

Definition in file [avr\\_op\\_decode.h](#).

#### 4.29.2 Function Documentation

## 4.29.2.1 AVR\_Decode()

```
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

$O \leftarrow$	Opcode to decode
$P \leftarrow$	
—	

Definition at line 400 of file [avr\\_op\\_decode.c](#).

## 4.29.2.2 AVR\_Decoder\_Function()

```
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

$O \leftarrow$	Opcode to return the instruction decode function for
$P \leftarrow$	
—	

## Returns

Pointer to an opcode/instruction decoder routine

! 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

! 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](#).

## 4.30 avr\_op\_decode.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      )\ ) |
00004 *      ((/( ((/(      \      ( ((/( | -- [ Funkenstein ] -----
00005 *      /( ) / ) )((( ) )\ ) / ) | -- [ Little ] -----
00006 *      ( ) _ ( )      )\ _ )\ ( ( ) ( ) | -- [ AVR ] -----
00007 *      | _ | |      ( ) _ ( ) \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ / _ \ \ v / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ | / _ \ \ \ / | _ | \ |
00010 *      | _ | | _ | / _ \ \ \ / | _ | \ |
00011 *      | _ | | _ | / _ \ \ \ / | _ | \ | "Yeah, it does Arduino..."
00012 *      * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 *      See license.txt for details
00014 *****/
00021 #ifndef __AVR_OP_DECODE_H__
00022 #define __AVR_OP_DECODE_H__
00023
00024 #include <stdint.h>
00025 #include "avr_cpu.h"
00026
00027 //-----
00028 // Format decoder function jump table
00029 typedef void (*AVR_Decoder)( uint16_t OP_ );
00030
00031 //-----
00041 AVR_Decoder AVR_Decoder_Function( uint16_t OP_ );
00042
00043 //-----
00052 void AVR_Decode( uint16_t OP_ );
00053
00054 #endif
00055
```

## 4.31 src/avr\_cpu/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\_SBRB** (uint16\_t OP\_)
- uint8\_t **AVR\_Opcode\_Size** (uint16\_t OP\_)

*AVR\_Opcode\_Size.*

#### 4.31.1 Detailed Description

Module providing opcode sizes.

Definition in file [avr\\_op\\_size.c](#).

#### 4.31.2 Function Documentation

##### 4.31.2.1 AVR\_Opcode\_Size()

```
uint8_t AVR_Opcode_Size (
    uint16_t OP_ )
```

*AVR\_Opcode\_Size.*

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

## Parameters

$O \leftarrow$	Opcode word to determine instruction size for
$P \leftarrow$	
$-$	

## Returns

The number of words in an instruction

Definition at line 150 of file [avr\\_op\\_size.c](#).

## 4.32 avr\_op\_size.c

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( ((/(      \      ( ((/(      | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( ) \      / ( )      | -- [ Little ] -----
00006 *      ( ) _ ( )      ) \ _ ) \ ( ( ( ( )      | -- [ AVR ] -----
00007 *      | _ | |      ( ) _ \ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \      \ V / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _ _      / _ \      \ /      | _ | _      |
00010 *      | _ | | _ _ _      / _ \      \ /      | _ | _      | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #include <stdint.h>
00022
00023 #include "emu_config.h"
00024
00025 #include "avr_op_size.h"
00026
00027 //-----
00028 static uint8_t AVR_Opcode_Size_NOP( uint16_t OP_ )
00029 {
00030     return 1;
00031 }
00032 //-----
00033 static uint8_t AVR_Opcode_Size_Register_Pair_4bit( uint16_t OP_ )
00034 {
00035     return 1;
00036 }
00037 //-----
00038 static uint8_t AVR_Opcode_Size_Register_Pair_3bit( uint16_t OP_ )
00039 {
00040     return 1;
00041 }
00042 //-----
00043 static uint8_t AVR_Opcode_Size_Register_Pair_5bit( uint16_t OP_ )
00044 {
00045     return 1;
00046 }
00047 //-----
00048 static uint8_t AVR_Opcode_Size_Register_Immediate( uint16_t OP_ )
00049 {
00050     return 1;
00051 }
00052 //-----
00053 static uint8_t AVR_Opcode_Size_LDST_YZ_k( uint16_t OP_ )
00054 {
00055     return 1;
00056 }
00057 //-----
00058 static uint8_t AVR_Opcode_Size_LDST( uint16_t OP_ )
00059 {
00060     return 1;
00061 }
00062 //-----
00063 static uint8_t AVR_Opcode_Size_LDS_STS( uint16_t OP_ )
00064 {
00065     return 2;
00066 }

```

```

00067 //-----
00068 static uint8_t AVR_Opcode_Size_Register_Single( uint16_t OP_)
00069 {
00070     return 1;
00071 }
00072 //-----
00073 static uint8_t AVR_Opcode_Size_Register_SC( uint16_t OP_)
00074 {
00075     return 1;
00076 }
00077 //-----
00078 static uint8_t AVR_Opcode_Size_Misc( uint16_t OP_)
00079 {
00080     return 1;
00081 }
00082 //-----
00083 static uint8_t AVR_Opcode_Size_Indirect_Jump( uint16_t OP_)
00084 {
00085     return 1;
00086 }
00087 //-----
00088 static uint8_t AVR_Opcode_Size_DEC_Rd( uint16_t OP_)
00089 {
00090     return 1;
00091 }
00092 //-----
00093 static uint8_t AVR_Opcode_Size_DES_round_4( uint16_t OP_)
00094 {
00095     return 1;
00096 }
00097 //-----
00098 static uint8_t AVR_Opcode_Size_JMP_CALL_22( uint16_t OP_)
00099 {
00100     return 2;
00101 }
00102 //-----
00103 static uint8_t AVR_Opcode_Size_ADJW_SBIW_6( uint16_t OP_)
00104 {
00105     return 1;
00106 }
00107 //-----
00108 static uint8_t AVR_Opcode_Size_IO_Bit( uint16_t OP_)
00109 {
00110     return 1;
00111 }
00112 //-----
00113 static uint8_t AVR_Opcode_Size_MUL( uint16_t OP_)
00114 {
00115     return 1;
00116 }
00117 //-----
00118 static uint8_t AVR_Opcode_Size_IO_In_Out( uint16_t OP_)
00119 {
00120     return 1;
00121 }
00122 //-----
00123 static uint8_t AVR_Opcode_Size_Relative_Jump( uint16_t OP_)
00124 {
00125     return 1;
00126 }
00127 //-----
00128 static uint8_t AVR_Opcode_Size_LDI( uint16_t OP_)
00129 {
00130     return 1;
00131 }
00132 //-----
00133 static uint8_t AVR_Opcode_Size_Conditional_Branch( uint16_t OP_)
00134 {
00135     return 1;
00136 }
00137 //-----
00138 static uint8_t AVR_Opcode_Size_BLD_BST( uint16_t OP_)
00139 {
00140     return 1;
00141 }
00142 //-----
00143 //-----
00144 static uint8_t AVR_Opcode_Size_SBRB_SBRB( uint16_t OP_)
00145 {
00146     return 1;
00147 }
00148 //-----
00149 //-----
00150 uint8_t AVR_Opcode_Size( uint16_t OP_ )
00151 {
00152     if (( OP_ & 0xFF0F) == 0x9408 )
00153     {

```

```

00154         // SEx/CLx status register clear/set bit.
00155         return AVR_Opcode_Size_Register_SC( OP_ );
00156     }
00157     else if ( ( OP_ & 0xFF0F ) == 0x9508 )
00158     {
00159         // Miscellaneous instruction
00160         return AVR_Opcode_Size_Misc( OP_ );
00161     }
00162     else if ( ( OP_ & 0xFF0F ) == 0x940B )
00163     {
00164         // Des round k
00165         return AVR_Opcode_Size_DES_round_4( OP_ );
00166     }
00167     else if ( ( ( OP_ & 0xFF00 ) == 0x0100 ) ||
00168              ( ( OP_ & 0xFF00 ) == 0x0200 ) )
00169     {
00170         // Register pair 4bit (MOVW, MULS)
00171         return AVR_Opcode_Size_Register_Pair_4bit( OP_ );
00172     }
00173     else if ( ( OP_ & 0xFF00 ) == 0x0300 )
00174     {
00175         // 3-bit register pair (R16->R23) - (FMUL, FMULS, FMULSU, MULSU)
00176         return AVR_Opcode_Size_Register_Pair_3bit( OP_ );
00177     }
00178     else if ( ( OP_ & 0xFF00 ) <= 0x4F00 )
00179     {
00180         // Register pair 5bit
00181         return AVR_Opcode_Size_Register_Pair_5bit( OP_ );
00182     }
00183     else if ( ( OP_ & 0xFF00 ) <= 0x7F00 )
00184     {
00185         // Register immediate
00186         return AVR_Opcode_Size_Register_Immediate( OP_ );
00187     }
00188     else if ( ( OP_ & 0xFEEF ) == 0x9409 )
00189     {
00190         // Indirect Jump/call
00191         return AVR_Opcode_Size_Indirect_Jump( OP_ );
00192     }
00193     else if ( ( OP_ & 0xFE08 ) == 0x9400 )
00194     {
00195         // 1-operand instructions.
00196         return AVR_Opcode_Size_Register_Single( OP_ );
00197     }
00198     else if ( ( OP_ & 0xFE0F ) == 0x940A )
00199     {
00200         // Dec Rd
00201         return AVR_Opcode_Size_DEC_Rd( OP_ );
00202     }
00203     else if ( ( OP_ & 0xFE0C ) == 0x940C )
00204     {
00205         // Jmp/call abs22
00206         return AVR_Opcode_Size_JMP_CALL_22( OP_ );
00207     }
00208     else if ( ( OP_ & 0xFE00 ) == 0x9600 )
00209     {
00210         // ADIW/SBIW Rp
00211         return AVR_Opcode_Size_ADIW_SBIW_6( OP_ );
00212     }
00213     else if ( ( OP_ & 0xFC0F ) == 0x9000 )
00214     {
00215         // LDS/STS
00216         return AVR_Opcode_Size_LDS_STS( OP_ );
00217     }
00218     else if ( ( OP_ & 0xFC00 ) == 0x9000 )
00219     {
00220         // LD/ST other
00221         return AVR_Opcode_Size_LDST( OP_ );
00222     }
00223     else if ( ( OP_ & 0xFC00 ) == 0x9800 )
00224     {
00225         // IO Space bit operations
00226         return AVR_Opcode_Size_IO_Bit( OP_ );
00227     }
00228     else if ( ( OP_ & 0xFC00 ) == 0x9C00 )
00229     {
00230         // MUL unsigned R1:R0 = Rr x Rd
00231         return AVR_Opcode_Size_MUL( OP_ );
00232     }
00233     else if ( ( OP_ & 0xFC00 ) == 0xF800 )
00234     {
00235         // BLD/BST register bit to STATUS.T
00236         return AVR_Opcode_Size_BLD_BST( OP_ );
00237     }
00238     else if ( ( OP_ & 0xFC00 ) == 0xFC00 )
00239     {
00240         // SBRC/SBRS

```



```

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

```

## 4.33 src/avr\_cpu/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\_Opcode\_Size.*

#### 4.33.1 Detailed Description

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

Definition in file [avr\\_op\\_size.h](#).

#### 4.33.2 Function Documentation

##### 4.33.2.1 AVR\_Opcode\_Size()

```
uint8_t AVR_Opcode_Size (
    uint16_t OP_ )
```

*AVR\_Opcode\_Size.*

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



## Functions

- static void **AVR\_Abort** (void)
- static uint32\_t **Get\_ZAddress** (void)
- static uint32\_t **Get\_ZAddressPostInc** (void)
- static uint32\_t **Get\_ZAddressPreDec** (void)
- static uint32\_t **Get\_ZAddressWithRAMP** (void)
- static uint32\_t **Get\_ZAddressPostIncWithRAMP** (void)
- static uint32\_t **Get\_ZAddressPreDecWithRAMP** (void)
- static uint32\_t **Get\_YAddress** (void)
- static uint32\_t **Get\_XAddress** (void)
- static void **Data\_Write** (uint32\_t u32Addr\_, uint8\_t u8Val\_)
- static uint8\_t **Data\_Read** (uint32\_t u32Addr\_)
- static void **AVR\_Opcode\_NOP** (void)
- static void **ADD\_Half\_Carry** (uint8\_t Rd\_, uint8\_t Rr\_, uint8\_t Result\_)
- static void **ADD\_Full\_Carry** (uint8\_t Rd\_, uint8\_t Rr\_, uint8\_t Result\_)
- static void **ADD\_Overflow\_Flag** (uint8\_t Rd\_, uint8\_t Rr\_, uint8\_t Result\_)
- static void **Signed\_Flag** (void)
- static void **R8\_Zero\_Flag** (uint8\_t R\_)
- static void **R8\_CPC\_Zero\_Flag** (uint8\_t R\_)
- static void **R8\_Negative\_Flag** (uint8\_t R\_)
- static void **AVR\_Opcode\_ADD** (void)
- static void **AVR\_Opcode\_ADC** (void)
- static void **R16\_Negative\_Flag** (uint16\_t Result\_)
- static void **R16\_Zero\_Flag** (uint16\_t Result\_)
- static void **ADIW\_Overflow\_Flag** (uint16\_t Rd\_, uint16\_t Result\_)
- static void **ADIW\_Carry\_Flag** (uint16\_t Rd\_, uint16\_t Result\_)
- static void **AVR\_Opcode\_ADIW** (void)
- static void **SUB\_Overflow\_Flag** (uint8\_t Rd\_, uint8\_t Rr\_, uint8\_t Result\_)
- static void **SUB\_Half\_Carry** (uint8\_t Rd\_, uint8\_t Rr\_, uint8\_t Result\_)
- static 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)
- static void **SBIW\_Overflow\_Flag** (uint16\_t Rd\_, uint16\_t Result\_)
- static 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)
- static void **NEG\_Overflow\_Flag** (uint8\_t u8Result\_)
- static 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)
- static void **INC\_Overflow\_Flag** (uint8\_t u8Result\_)
- static void **AVR\_Opcode\_INC** (void)
- static void **DEC\_Overflow\_Flag** (uint8\_t u8Result\_)
- static void **AVR\_Opcode\_DEC** (void)
- static void **AVR\_Opcode\_SER** (void)
- static void **Mul\_Carry\_Flag** (uint16\_t R\_)

- static 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 void **Unconditional\_Jump** (uint16\_t u16Addr\_)
- static void **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)
- static void **CP\_Half\_Carry** (uint8\_t Rd\_, uint8\_t Rr\_, uint8\_t Result\_)
- static void **CP\_Full\_Carry** (uint8\_t Rd\_, uint8\_t Rr\_, uint8\_t Result\_)
- static 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 void **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)
- static void **LSL\_HalfCarry\_Flag** (uint8\_t R\_)
- static void **Left\_Carry\_Flag** (uint8\_t R\_)
- static void **Rotate\_Overflow\_Flag** ()
- static void **AVR\_Opcode\_LSL** (void)
- static 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.*

#### 4.35.1 Detailed Description

AVR CPU - Opcode implementation.

Definition in file [avr\\_opcodes.c](#).

#### 4.35.2 Function Documentation

##### 4.35.2.1 AVR\_Opcode\_DES()

```
static void AVR_Opcode_DES (
    void ) [static]
```

ToDo - Implement DES

Definition at line 811 of file [avr\\_opcodes.c](#).

##### 4.35.2.2 AVR\_Opcode\_EICALL()

```
static void AVR_Opcode_EICALL (
    void ) [static]
```

! ToDo - Implement EIND calling!

Definition at line 920 of file [avr\\_opcodes.c](#).

##### 4.35.2.3 AVR\_Opcode\_EI JMP()

```
static void AVR_Opcode_EI JMP (
    void ) [static]
```

ToDo - implement EIND instructions

Definition at line 855 of file [avr\\_opcodes.c](#).

##### 4.35.2.4 AVR\_Opcode\_Function()

```
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

$O \leftarrow$	Opcode to return an "opcode execution" function pointer for
$P \leftarrow$	
—	

## Returns

Opcode execution function pointer corresponding to the given opcode.

Definition at line 1915 of file [avr\\_opcodes.c](#).

## 4.35.2.5 AVR Opcode\_SPM()

```
static void AVR_Opcode_SPM (
    void ) [static]
```

! Implement later...

Definition at line 1594 of file [avr\\_opcodes.c](#).

## 4.35.2.6 AVR Opcode\_SPM\_Z\_Postinc2()

```
static void AVR_Opcode_SPM_Z_Postinc2 (
    void ) [static]
```

! Implement later...

Definition at line 1600 of file [avr\\_opcodes.c](#).

## 4.35.2.7 AVR\_RunOpcode()

```
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

$O \leftarrow$	Opcode to execute
$P \leftarrow$	
$-$	

Definition at line 2116 of file [avr\\_opcodes.c](#).

## 4.36 avr\_opcodes.c

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      )\ )      |
00004 *      ((/( ((/(      \      ( ((/(      | -- [ Funkenstein ] -----
00005 *      /( ) )/( ) )((( ( \      \      / ( )      | -- [ Little ] -----
00006 *      ( ) _ ( )      )\ _ )\ ( ( ( ( ) _      | -- [ AVR ] -----
00007 *      | _ | |      ( ) _ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ V / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _      / _ \ \ \ / | _ \      |
00010 *      |      |      |      |      |      | "Yeah, it does Arduino..."
00011 *      -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00022 #include <stdint.h>
00023 #include <stdio.h>
00024 #include <stdlib.h>
00025
00026 #include "avr_cpu_print.h"
00027 #include "emu_config.h"
00028 #include "avr_opcodes.h"
00029 #include "interactive.h"
00030 #include "write_callout.h"
00031 #include "interrupt_callout.h"
00032
00033 //-----
00034 #define DEBUG_PRINT(...)
00035
00036 //-----
00037 static void AVR_Abort(void)
00038 {
00039     print_core_regs();
00040     exit(-1);
00041 }
00042
00043 //-----
00044 static uint32_t Get_ZAddress(void)
00045 {
00046     return (uint32_t)stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z;
00047 }
00048
00049 //-----
00050 static uint32_t Get_ZAddressPostInc(void)
00051 {
00052     uint32_t u32RC = (uint32_t)stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z;
00053     stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z++;
00054
00055     return u32RC;
00056 }
00057
00058 //-----
00059 static uint32_t Get_ZAddressPreDec(void)
00060 {
00061     stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z--;
00062     return stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z;
00063 }
00064
00065 //-----
00066 static uint32_t Get_ZAddressWithRAMP(void)
00067 {
00068     return ((uint32_t)stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z) |
00069         ((uint32_t)stCPU.pstRAM->stRegisters.RAMPZ) << 16);
00070 }
00071
00072 //-----
00073 static uint32_t Get_ZAddressPostIncWithRAMP(void)

```



```

00075 {
00076     uint32_t u32RC = (((uint32_t)stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z) |
00077                      (((uint32_t)stCPU.pstRAM->stRegisters.RAMPZ) << 16));
00078
00079     stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z++;
00080
00081     return u32RC;
00082 }
00083
00084 //-----
00085 static uint32_t Get_ZAddressPreDecWithRAMP(void)
00086 {
00087     stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z--;
00088     return (((uint32_t)stCPU.pstRAM->stRegisters.CORE_REGISTERS.Z) |
00089            (((uint32_t)stCPU.pstRAM->stRegisters.RAMPZ) << 16));
00090 }
00091
00092
00093 //-----
00094 static uint32_t Get_YAddress(void)
00095 {
00096     return ((uint32_t)stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y);
00097 }
00098
00099 //-----
00100 static uint32_t Get_XAddress(void)
00101 {
00102     return ((uint32_t)stCPU.pstRAM->stRegisters.CORE_REGISTERS.X);
00103 }
00104
00105 //-----
00106 static void Data_Write( uint32_t u32Addr_, uint8_t u8Val_ )
00107 {
00108     // Writing to RAM can be a tricky deal, because the address space is shared
00109     // between RAM, the core registers, and a bunch of peripheral I/O registers.
00110     DEBUG_PRINT("Write: 0x%08X=%02X\n", u32Addr_, u8Val_ );
00111     if (!WriteCallout_Run( u32Addr_, u8Val_ ))
00112     {
00113         return;
00114     }
00115
00116     // Check to see if the write operation falls within the peripheral I/O range
00117     if (u32Addr_ >= 32 && u32Addr_ <= 255)
00118     {
00119         // I/O range - check to see if there's a peripheral installed at this address
00120         IOWriterList *pstIOWrite = stCPU.apstPeriphWriteTable[ u32Addr_ ];
00121
00122         // If there is a peripheral or peripherals
00123         if (pstIOWrite)
00124         {
00125             // Iterate through the list of installed peripherals at this address, and
00126             // call their write handler
00127             while (pstIOWrite)
00128             {
00129                 pstIOWrite->pfWriter( pstIOWrite->pvContext, (uint8_t)u32Addr_, u8Val_ );
00130                 pstIOWrite = pstIOWrite->next;
00131             }
00132         }
00133         // Otherwise, there is no peripheral -- just assume we can treat this as normal RAM.
00134         else
00135         {
00136             stCPU.pstRAM->au8RAM[ u32Addr_ ] = u8Val_;
00137         }
00138     }
00139     else if (u32Addr_ >= (stCPU.u32RAMSize + 256))
00140     {
00141         fprintf( stderr, "[Write Abort] RAM Address 0x%08X is out of range!\n", u32Addr_ );
00142         AVR_Abort();
00143     }
00144     // RAM address range - direct write-through.
00145     else
00146     {
00147         stCPU.pstRAM->au8RAM[ u32Addr_ ] = u8Val_;
00148     }
00149 }
00150 }
00151
00152 //-----
00153 static uint8_t Data_Read( uint32_t u32Addr_ )
00154 {
00155     // Writing to RAM can be a tricky deal, because the address space is shared
00156     // between RAM, the core registers, and a bunch of peripheral I/O registers.
00157
00158     // Check to see if the write operation falls within the peripheral I/O range
00159     DEBUG_PRINT( "Data Read: %08X\n", u32Addr_ );
00160     if (u32Addr_ >= 32 && u32Addr_ <= 255)
00161     {

```

```

00162         // I/O range - check to see if there's a peripheral installed at this address
00163         IOReaderList *pstIORead = stCPU.apstPeriphReadTable[ u32Addr_ ];
00164         DEBUG_PRINT( "Peripheral Read: 0x%08X\n", u32Addr_ );
00165         // If there is a peripheral or peripherals
00166         if (pstIORead)
00167         {
00168             DEBUG_PRINT(" Found peripheral\n");
00169             // Iterate through the list of installed peripherals at this address, and
00170             // call their read handler
00171             uint8_t u8Val;
00172             while (pstIORead)
00173             {
00174                 pstIORead->pfReader( pstIORead->pvContext, (uint8_t)u32Addr_, &u8Val);
00175                 pstIORead = pstIORead->next;
00176             }
00177             return u8Val;
00178         }
00179         // Otherwise, there is no peripheral -- just assume we can treat this as normal RAM.
00180         else
00181         {
00182             DEBUG_PRINT(" No peripheral\n");
00183             return stCPU.pstRAM->au8RAM[ u32Addr_ ];
00184         }
00185     }
00186     else if (u32Addr_ >= (stCPU.u32RAMSize + 256))
00187     {
00188         fprintf( stderr, "[Read Abort] RAM Address 0x%04X is out of range!\n", u32Addr_ );
00189         AVR_Abort();
00190     }
00191     // RAM address range - direct read
00192     else
00193     {
00194         return stCPU.pstRAM->au8RAM[ u32Addr_ ];
00195     }
00196 }
00197
00198 //-----
00199 static void AVR_Opcode_NOP( void )
00200 {
00201     // Nop - do nothing.
00202 }
00203
00204 //-----
00205 static void ADD_Half_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_ )
00206 {
00207     stCPU.pstRAM->stRegisters.SREG.H =
00208         ( ((Rd_ & Rr_) & 0x08 )
00209         | ((Rr_ & (~Result_)) & 0x08 )
00210         | (((~Result_) & Rd_) & 0x08) ) != false;
00211 }
00212
00213 //-----
00214 static void ADD_Full_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_ )
00215 {
00216     stCPU.pstRAM->stRegisters.SREG.C =
00217         ( ((Rd_ & Rr_) & 0x80 )
00218         | ((Rr_ & (~Result_)) & 0x80 )
00219         | (((~Result_) & Rd_) & 0x80) ) != false;
00220 }
00221
00222 //-----
00223 static void ADD_Overflow_Flag( uint8_t Rd_, uint8_t Rr_, uint8_t Result_ )
00224 {
00225     stCPU.pstRAM->stRegisters.SREG.V =
00226         ( ((Rd_ & Rr_ & ~Result_) & 0x80 )
00227         | ((~Rd_ & ~Rr_ & Result_) & 0x80 ) ) != 0;
00228 }
00229
00230 //-----
00231 static void Signed_Flag( void )
00232 {
00233     unsigned int N = stCPU.pstRAM->stRegisters.SREG.N;
00234     unsigned int V = stCPU.pstRAM->stRegisters.SREG.V;
00235
00236     stCPU.pstRAM->stRegisters.SREG.S = N ^ V;
00237 }
00238
00239 //-----
00240 static void R8_Zero_Flag( uint8_t R_ )
00241 {
00242     stCPU.pstRAM->stRegisters.SREG.Z = (R_ == 0);
00243 }
00244
00245 //-----
00246 static void R8_CPC_Zero_Flag( uint8_t R_ )
00247 {
00248     stCPU.pstRAM->stRegisters.SREG.Z = (stCPU.pstRAM->stRegisters.SREG.Z && (R_ == 0));

```

```

00249 }
00250
00251 //-----
00252 static void R8_Negative_Flag( uint8_t R_ )
00253 {
00254     stCPU.pstRAM->stRegisters.SREG.N = ((R_ & 0x80) == 0x80);
00255 }
00256
00257 //-----
00258 static void AVR_Opcode_ADD( void )
00259 {
00260     uint8_t u8Result;
00261     uint8_t u8Rd = *(stCPU.Rd);
00262     uint8_t u8Rr = *(stCPU.Rr);
00263
00264     u8Result = u8Rd + u8Rr;
00265     *(stCPU.Rd) = u8Result;
00266
00267 // ---- Update flags ----
00268     ADD_Half_Carry( u8Rd, u8Rr, u8Result );
00269     ADD_Full_Carry( u8Rd, u8Rr, u8Result );
00270     ADD_Overflow_Flag( u8Rd, u8Rr, u8Result );
00271     R8_Negative_Flag( u8Result );
00272     R8_Zero_Flag( u8Result );
00273     Signed_Flag();
00274 }
00275
00276 //-----
00277 static void AVR_Opcode_ADC( void )
00278 {
00279     uint8_t u8Result;
00280     uint8_t u8Rd = *(stCPU.Rd);
00281     uint8_t u8Rr = *(stCPU.Rr);
00282     uint8_t u8Carry = (stCPU.pstRAM->stRegisters.SREG.C);
00283
00284     u8Result = u8Rd + u8Rr + u8Carry;
00285     *(stCPU.Rd) = u8Result;
00286
00287 // ---- Update flags ----
00288     ADD_Half_Carry( u8Rd, u8Rr, u8Result );
00289     ADD_Full_Carry( u8Rd, u8Rr, u8Result );
00290     ADD_Overflow_Flag( u8Rd, u8Rr, u8Result );
00291     R8_Negative_Flag( u8Result );
00292     R8_Zero_Flag( u8Result );
00293     Signed_Flag();
00294 }
00295
00296 //-----
00297 static void R16_Negative_Flag( uint16_t Result_ )
00298 {
00299     stCPU.pstRAM->stRegisters.SREG.N =
00300         ((Result_ & 0x8000) != 0);
00301 }
00302
00303 //-----
00304 static void R16_Zero_Flag( uint16_t Result_ )
00305 {
00306     stCPU.pstRAM->stRegisters.SREG.Z =
00307         (Result_ == 0);
00308 }
00309
00310 //-----
00311 static void ADIW_Overflow_Flag( uint16_t Rd_, uint16_t Result_ )
00312 {
00313     stCPU.pstRAM->stRegisters.SREG.V =
00314         (((Rd_ & 0x8000) == 0) && ((Result_ & 0x8000) == 0x8000));
00315 }
00316
00317 //-----
00318 static void ADIW_Carry_Flag( uint16_t Rd_, uint16_t Result_ )
00319 {
00320     stCPU.pstRAM->stRegisters.SREG.C =
00321         (((Rd_ & 0x8000) == 0x8000) && ((Result_ & 0x8000) == 0));
00322 }
00323
00324 //-----
00325 static void AVR_Opcode_ADIW( void )
00326 {
00327     uint16_t u16K = (stCPU.K);
00328     uint16_t u16Rd = *(stCPU.Rd16);
00329     uint16_t u16Result;
00330
00331     u16Result = u16Rd + u16K;
00332     *(stCPU.Rd16) = u16Result;
00333
00334 // ---- Update Flags ----
00335     ADIW_Carry_Flag( u16Rd, u16Result);

```

```

00336     ADIW_Overflow_Flag( u16Rd, u16Result );
00337     R16_Negative_Flag( u16Result );
00338     R16_Zero_Flag( u16Result );
00339     Signed_Flag();
00340 }
00341
00342 //-----
00343 static void SUB_Overflow_Flag( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
00344 {
00345     stCPU.pstRAM->stRegisters.SREG.V =
00346         ( ((Rd_ & ~Rr_ & ~Result_) & 0x80 )
00347           | ((~Rd_ & Rr_ & Result_) & 0x80 ) ) != 0;
00348 }
00349 //-----
00350 static void SUB_Half_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
00351 {
00352     stCPU.pstRAM->stRegisters.SREG.H =
00353         ( ((~Rd_ & Rr_) & 0x08 )
00354           | ((Rr_ & Result_) & 0x08 )
00355           | ((Result_ & ~Rd_) & 0x08 ) ) == 0x08;
00356 }
00357 //-----
00358 static void SUB_Full_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_)
00359 {
00360     stCPU.pstRAM->stRegisters.SREG.C =
00361         ( ((~Rd_ & Rr_) & 0x80 )
00362           | ((Rr_ & Result_) & 0x80 )
00363           | ((Result_ & ~Rd_) & 0x80 ) ) == 0x80;
00364 }
00365
00366 //-----
00367 static void AVR_Opcode_SUB( void )
00368 {
00369     uint8_t u8Rd = *stCPU.Rd;
00370     uint8_t u8Rr = *stCPU.Rr;
00371     uint8_t u8Result = u8Rd - u8Rr;
00372
00373     *stCPU.Rd = u8Result;
00374
00375     //--Flags
00376     SUB_Half_Carry( u8Rd, u8Rr, u8Result);
00377     SUB_Full_Carry( u8Rd, u8Rr, u8Result);
00378     SUB_Overflow_Flag( u8Rd, u8Rr, u8Result);
00379     R8_Negative_Flag( u8Result);
00380     R8_Zero_Flag( u8Result);
00381     Signed_Flag();
00382 }
00383
00384 //-----
00385 static void AVR_Opcode_SUBI( void )
00386 {
00387     uint8_t u8Rd = *stCPU.Rd;
00388     uint8_t u8K = (uint8_t)stCPU.K;
00389     uint8_t u8Result = u8Rd - u8K;
00390
00391     *stCPU.Rd = u8Result;
00392
00393     //--Flags
00394     SUB_Half_Carry( u8Rd, u8K, u8Result);
00395     SUB_Full_Carry( u8Rd, u8K, u8Result);
00396     SUB_Overflow_Flag( u8Rd, u8K, u8Result);
00397     R8_Negative_Flag( u8Result);
00398     R8_Zero_Flag( u8Result);
00399     Signed_Flag();
00400 }
00401
00402 //-----
00403 static void AVR_Opcode_SBC( void )
00404 {
00405     uint8_t u8Rd = *stCPU.Rd;
00406     uint8_t u8Rr = *stCPU.Rr;
00407     uint8_t u8C = stCPU.pstRAM->stRegisters.SREG.C;
00408     uint8_t u8Result = u8Rd - u8Rr - u8C;
00409
00410     *stCPU.Rd = u8Result;
00411
00412     //--Flags
00413     SUB_Half_Carry( u8Rd, u8Rr, u8Result);
00414     SUB_Full_Carry( u8Rd, u8Rr, u8Result);
00415     SUB_Overflow_Flag( u8Rd, u8Rr, u8Result);
00416     R8_Negative_Flag( u8Result);
00417     if (u8Result)
00418     {
00419         stCPU.pstRAM->stRegisters.SREG.Z = 0;
00420     }
00421     Signed_Flag();
00422 }

```

```

00423
00424 //-----
00425 static void AVR_Opcode_SBCI( void )
00426 {
00427     uint8_t u8Rd = *stCPU.Rd;
00428     uint8_t u8K = (uint8_t)stCPU.K;
00429     uint8_t u8C = stCPU.pstRAM->stRegisters.SREG.C;
00430     uint8_t u8Result = u8Rd - u8K - u8C;
00431
00432     *stCPU.Rd = u8Result;
00433
00434     //--Flags
00435     SUB_Half_Carry( u8Rd, u8K, u8Result);
00436     SUB_Full_Carry( u8Rd, u8K, u8Result);
00437     SUB_Overflow_Flag( u8Rd, u8K, u8Result);
00438     R8_Negative_Flag( u8Result);
00439     if (u8Result)
00440     {
00441         stCPU.pstRAM->stRegisters.SREG.Z = 0;
00442     }
00443     Signed_Flag();
00444 }
00445
00446
00447 //-----
00448 static void SBIW_Overflow_Flag( uint16_t Rd_, uint16_t Result_)
00449 {
00450     stCPU.pstRAM->stRegisters.SREG.V =
00451         ((Rd_ & 0x8000) == 0x8000) && ((Result_ & 0x8000) == 0);
00452 }
00453
00454
00455 //-----
00456 static void SBIW_Full_Carry( uint16_t Rd_, uint16_t Result_)
00457 {
00458     stCPU.pstRAM->stRegisters.SREG.C =
00459         ((Rd_ & 0x8000) == 0) && ((Result_ & 0x8000) == 0x8000);
00460 }
00461
00462 //-----
00463 static void AVR_Opcode_SBIW( void )
00464 {
00465     uint16_t u16Rd = *stCPU.Rd16;
00466     uint16_t u16Result;
00467
00468     //fprintf( stderr, "SBIW: RD=[%4X], K=[%2X]\n", u16Rd, stCPU.K );
00469     u16Result = u16Rd - stCPU.K;
00470
00471     *stCPU.Rd16 = u16Result;
00472     //fprintf( stderr, "    Result=[%4X]\n", u16Result );
00473
00474     SBIW_Full_Carry( u16Rd, u16Result);
00475     SBIW_Overflow_Flag( u16Rd, u16Result);
00476     R16_Negative_Flag( u16Result);
00477     R16_Zero_Flag( u16Result);
00478     Signed_Flag();
00479
00480 }
00481
00482 //-----
00483 static void AVR_Opcode_AND( void )
00484 {
00485     uint8_t u8Rd = *stCPU.Rd;
00486     uint8_t u8Rr = *stCPU.Rr;
00487     uint8_t u8Result = u8Rd & u8Rr;
00488
00489     *stCPU.Rd = u8Result;
00490
00491     //--Update Status registers;
00492     stCPU.pstRAM->stRegisters.SREG.V = 0;
00493     R8_Negative_Flag( u8Result );
00494     R8_Zero_Flag( u8Result );
00495     Signed_Flag();
00496 }
00497
00498 //-----
00499 static void AVR_Opcode_ANDI( void )
00500 {
00501     uint8_t u8Rd = *stCPU.Rd;
00502     uint8_t u8Result = u8Rd & (uint8_t)stCPU.K;
00503
00504     *stCPU.Rd = u8Result;
00505
00506     //--Update Status registers;
00507     stCPU.pstRAM->stRegisters.SREG.V = 0;
00508     R8_Negative_Flag( u8Result );
00509     R8_Zero_Flag( u8Result );

```

```

00510     Signed_Flag();
00511 }
00512
00513 //-----
00514 static void AVR_Opcode_OR( void )
00515 {
00516     uint8_t u8Rd = *stCPU.Rd;
00517     uint8_t u8Rr = *stCPU.Rr;
00518     uint8_t u8Result = u8Rd | u8Rr;
00519
00520     *stCPU.Rd = u8Result;
00521
00522     //--Update Status registers;
00523     stCPU.pstRAM->stRegisters.SREG.V = 0;
00524     R8_Negative_Flag( u8Result );
00525     R8_Zero_Flag( u8Result );
00526     Signed_Flag();
00527 }
00528
00529 //-----
00530 static void AVR_Opcode_ORI( void )
00531 {
00532     uint8_t u8Rd = *stCPU.Rd;
00533     uint8_t u8Result = u8Rd | (uint8_t)stCPU.K;
00534
00535     *stCPU.Rd = u8Result;
00536
00537     //--Update Status registers;
00538     stCPU.pstRAM->stRegisters.SREG.V = 0;
00539     R8_Negative_Flag( u8Result );
00540     R8_Zero_Flag( u8Result );
00541     Signed_Flag();
00542 }
00543
00544 //-----
00545 static void AVR_Opcode_EOR( void )
00546 {
00547     uint8_t u8Rd = *stCPU.Rd;
00548     uint8_t u8Rr = *stCPU.Rr;
00549     uint8_t u8Result = u8Rd ^ u8Rr;
00550
00551     *stCPU.Rd = u8Result;
00552
00553     //--Update Status registers;
00554     stCPU.pstRAM->stRegisters.SREG.V = 0;
00555     R8_Negative_Flag( u8Result );
00556     R8_Zero_Flag( u8Result );
00557     Signed_Flag();
00558 }
00559
00560 //-----
00561 static void AVR_Opcode_COM( void )
00562 {
00563     // 1's complement.
00564     uint8_t u8Result = *stCPU.Rd;
00565     u8Result = (0xFF - u8Result);
00566
00567     *stCPU.Rd = u8Result;
00568
00569     //--Update Status registers;
00570     stCPU.pstRAM->stRegisters.SREG.V = 0;
00571     stCPU.pstRAM->stRegisters.SREG.C = 1;
00572     R8_Negative_Flag( u8Result );
00573     R8_Zero_Flag( u8Result );
00574     Signed_Flag();
00575 }
00576
00577 //-----
00578 static void NEG_Overflow_Flag( uint8_t u8Result_ )
00579 {
00580     stCPU.pstRAM->stRegisters.SREG.V = (u8Result_ == 0x80);
00581 }
00582
00583 //-----
00584 static void NEG_Carry_Flag( uint8_t u8Result_ )
00585 {
00586     stCPU.pstRAM->stRegisters.SREG.C = (u8Result_ != 0x00);
00587 }
00588
00589 //-----
00590 static void AVR_Opcode_NEG( void )
00591 {
00592     // 2's complement.
00593     uint8_t u8Result = *stCPU.Rd;
00594     u8Result = (0 - u8Result);
00595
00596     *stCPU.Rd = u8Result;

```

```

00597
00598     //--Update Status registers;
00599     NEG_Overflow_Flag( u8Result );
00600     NEG_Carry_Flag( u8Result );
00601     R8_Negative_Flag( u8Result );
00602     R8_Zero_Flag( u8Result );
00603     Signed_Flag();
00604 }
00605
00606 //-----
00607 static void AVR_Opcode_SBR( void )
00608 {
00609     // Set Bits in Register
00610     uint8_t u8Result = *stCPU.Rd;
00611     u8Result |= ((uint8_t)stCPU.K);
00612
00613     *stCPU.Rd = u8Result;
00614
00615     //--Update Status registers;
00616     stCPU.pstRAM->stRegisters.SREG.V = 0;
00617     R8_Negative_Flag( u8Result );
00618     R8_Zero_Flag( u8Result );
00619     Signed_Flag();
00620 }
00621
00622 //-----
00623 static void AVR_Opcode_CBR( void )
00624 {
00625     // Clear Bits in Register
00626     uint8_t u8Result = *stCPU.Rd;
00627     u8Result &= ~((uint8_t)stCPU.K);
00628
00629     *stCPU.Rd = u8Result;
00630
00631     //--Update Status registers;
00632     stCPU.pstRAM->stRegisters.SREG.V = 0;
00633     R8_Negative_Flag( u8Result );
00634     R8_Zero_Flag( u8Result );
00635     Signed_Flag();
00636 }
00637
00638 //-----
00639 static void INC_Overflow_Flag( uint8_t u8Result_ )
00640 {
00641     stCPU.pstRAM->stRegisters.SREG.V = (u8Result_ == 0x80);
00642 }
00643
00644 //-----
00645 static void AVR_Opcode_INC( void )
00646 {
00647     uint8_t u8Result;
00648     u8Result = *stCPU.Rd + 1;
00649
00650     *stCPU.Rd = u8Result;
00651
00652     //--Update Status registers;
00653     INC_Overflow_Flag( u8Result );
00654     R8_Negative_Flag( u8Result );
00655     R8_Zero_Flag( u8Result );
00656     Signed_Flag();
00657 }
00658 //-----
00659 static void DEC_Overflow_Flag( uint8_t u8Result_ )
00660 {
00661     stCPU.pstRAM->stRegisters.SREG.V = (u8Result_ == 0x7F);
00662 }
00663 //-----
00664 static void AVR_Opcode_DEC( void )
00665 {
00666     uint8_t u8Result;
00667     u8Result = *stCPU.Rd - 1;
00668
00669     *stCPU.Rd = u8Result;
00670
00671     //--Update Status registers;
00672     DEC_Overflow_Flag( u8Result );
00673     R8_Negative_Flag( u8Result );
00674     R8_Zero_Flag( u8Result );
00675     Signed_Flag();
00676 }
00677
00678 //-----
00679 static void AVR_Opcode_SER( void )
00680 {
00681     *stCPU.Rd = 0xFF;
00682 }
00683

```

```

00684 //-----
00685 static void Mul_Carry_Flag( uint16_t R_ )
00686 {
00687     stCPU.pstRAM->stRegisters.SREG.C = ((R_ & 0x8000) == 0x8000);
00688 }
00689
00690 //-----
00691 static void Mul_Zero_Flag( uint16_t R_ )
00692 {
00693     stCPU.pstRAM->stRegisters.SREG.Z = (R_ == 0);
00694 }
00695
00696 //-----
00697 static void AVR_Opcode_MUL( void )
00698 {
00699     uint16_t ul6Product;
00700     uint16_t ul6R1;
00701     uint16_t ul6R2;
00702
00703     ul6R1 = *stCPU.Rd;
00704     ul6R2 = *stCPU.Rr;
00705
00706     ul6Product = ul6R1 * ul6R2;
00707
00708     stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = ul6Product;
00709
00710     //-- Update Flags --
00711     Mul_Zero_Flag( ul6Product);
00712     Mul_Carry_Flag( ul6Product);
00713 }
00714
00715 //-----
00716 static void AVR_Opcode_MULS( void )
00717 {
00718     int16_t s16Product;
00719     int16_t s16R1;
00720     int16_t s16R2;
00721
00722     s16R1 = (int8_t)*stCPU.Rd;
00723     s16R2 = (int8_t)*stCPU.Rr;
00724
00725     s16Product = s16R1 * s16R2;
00726
00727     stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = (uint16_t)s16Product;
00728
00729     //-- Update Flags --
00730     Mul_Zero_Flag( (uint16_t)s16Product);
00731     Mul_Carry_Flag( (uint16_t)s16Product);
00732 }
00733
00734 //-----
00735 static void AVR_Opcode_MULSU( void )
00736 {
00737     int16_t s16Product;
00738     int16_t s16R1;
00739     uint16_t ul6R2;
00740
00741     s16R1 = (int8_t)*stCPU.Rd;
00742     ul6R2 = *stCPU.Rr;
00743
00744     s16Product = s16R1 * ul6R2;
00745
00746     stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = (uint16_t)s16Product;
00747
00748     //-- Update Flags --
00749     Mul_Zero_Flag( (uint16_t)s16Product);
00750     Mul_Carry_Flag( (uint16_t)s16Product);
00751 }
00752
00753 //-----
00754 static void AVR_Opcode_FMUL( void )
00755 {
00756     uint16_t ul6Product;
00757     uint16_t ul6R1;
00758     uint16_t ul6R2;
00759
00760     ul6R1 = *stCPU.Rd;
00761     ul6R2 = *stCPU.Rr;
00762
00763     ul6Product = ul6R1 * ul6R2;
00764
00765     stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = ul6Product << 1;
00766
00767     //-- Update Flags --
00768     Mul_Zero_Flag( ul6Product);
00769     Mul_Carry_Flag( ul6Product);
00770 }

```



```

00771
00772 //-----
00773 static void AVR_Opcode_FMULS( void )
00774 {
00775     int16_t s16Product;
00776     int16_t s16R1;
00777     int16_t s16R2;
00778
00779     s16R1 = (int8_t)*stCPU.Rd;
00780     s16R2 = (int8_t)*stCPU.Rr;
00781
00782     s16Product = s16R1 * s16R2;
00783
00784     stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = ((uint16_t)s16Product) << 1;
00785
00786     //-- Update Flags --
00787     Mul_Zero_Flag( (uint16_t)s16Product);
00788     Mul_Carry_Flag( (uint16_t)s16Product);
00789 }
00790
00791 //-----
00792 static void AVR_Opcode_FMULSU( void )
00793 {
00794     int16_t s16Product;
00795     int16_t s16R1;
00796     uint16_t u16R2;
00797
00798     s16R1 = (int8_t)*stCPU.Rd;
00799     u16R2 = *stCPU.Rr;
00800
00801     s16Product = s16R1 * u16R2;
00802
00803     stCPU.pstRAM->stRegisters.CORE_REGISTERS.r1_0 = ((uint16_t)s16Product) << 1;
00804
00805     //-- Update Flags --
00806     Mul_Zero_Flag( (uint16_t)s16Product);
00807     Mul_Carry_Flag( (uint16_t)s16Product);
00808 }
00809
00810 //-----
00811 static void AVR_Opcode_DES( void )
00812 {
00813 }
00814 }
00815
00816 //-----
00817 static void Unconditional_Jump( uint16_t u16Addr_ )
00818 {
00819     stCPU.u32PC = u16Addr_;
00820     stCPU.u16ExtraPC = 0;
00821
00822     // Feature -- Terminate emulator if jump-to-zero encountered at runtime.
00823     if (stCPU.u32PC == 0 && stCPU.bExitOnReset)
00824     {
00825         exit(0);
00826     }
00827 }
00828
00829 //-----
00830 static void Relative_Jump( uint16_t u16Offset_ )
00831 {
00832     // u16Offset_ Will always be 1 or 2, based on the size of the next opcode
00833     // in a program
00834
00835     stCPU.u32PC += u16Offset_;
00836     stCPU.u16ExtraPC = 0;
00837     stCPU.u16ExtraCycles += u16Offset_;
00838 }
00839
00840 //-----
00841 static void AVR_Opcode_RJMP( void )
00842 {
00843     int32_t s32NewPC = (int32_t)stCPU.u32PC + (int32_t)stCPU.k_s + 1;
00844
00845     Unconditional_Jump( (uint16_t)s32NewPC );
00846 }
00847
00848 //-----
00849 static void AVR_Opcode_IJMP( void )
00850 {
00851     Unconditional_Jump( Get_ZAddress() );
00852 }
00853
00854 //-----
00855 static void AVR_Opcode_EIJMP( void )
00856 {
00857 }
00858 }
00859

```

```

00860 //-----
00861 static void AVR_Opcode_JMP( void )
00862 {
00863     Unconditional_Jump( (uint16_t)stCPU.k );
00864 }
00865
00866 //-----
00867 static void AVR_Opcode_RCALL( void )
00868 {
00869     // Push the next instruction address onto the stack
00870     uint32_t u32PC = (((uint16_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
00871                     (((uint16_t)stCPU.pstRAM->stRegisters.SPL.r));
00872
00873     uint32_t u32StoredPC = stCPU.u32PC + 1;
00874
00875     Data_Write( u32PC, (uint8_t)(u32StoredPC & 0x00FF));
00876     Data_Write( u32PC - 1, (uint8_t)(u32StoredPC >> 8));
00877
00878     // Stack is post-decremented
00879     u32PC -= 2;
00880
00881     // Set the new PC (relative call)
00882     int32_t s32NewPC = (int32_t)stCPU.u32PC + (int32_t)stCPU.k_s + 1;
00883     uint32_t u32NewPC = (uint32_t)s32NewPC;
00884
00885     // Store the new SP.
00886     stCPU.pstRAM->stRegisters.SPH.r = (u32PC >> 8);
00887     stCPU.pstRAM->stRegisters.SPL.r = (u32PC & 0x00FF);
00888
00889     // Set the new PC
00890     Unconditional_Jump( u32NewPC );
00891 }
00892
00893 //-----
00894 static void AVR_Opcode_ICALL( void )
00895 {
00896     // Push the next instruction address onto the stack
00897     uint32_t u32SP = (((uint32_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
00898                     (((uint32_t)stCPU.pstRAM->stRegisters.SPL.r));
00899
00900     uint32_t u32StoredPC = stCPU.u32PC + 1;
00901
00902     Data_Write( u32SP, (uint8_t)(u32StoredPC & 0x00FF));
00903     Data_Write( u32SP - 1, (uint8_t)(u32StoredPC >> 8));
00904
00905     // Stack is post-decremented
00906     u32SP -= 2;
00907
00908     // Set the new PC
00909     uint32_t u32NewPC = Get_ZAddress();
00910
00911     // Store the new SP.
00912     stCPU.pstRAM->stRegisters.SPH.r = (u32SP >> 8);
00913     stCPU.pstRAM->stRegisters.SPL.r = (u32SP & 0x00FF);
00914
00915     // Set the new PC
00916     Unconditional_Jump( u32NewPC );
00917 }
00918
00919 //-----
00920 static void AVR_Opcode_EICALL( void )
00921 {
00922 }
00923
00924
00925 //-----
00926 static void AVR_Opcode_CALL( void )
00927 {
00928     // See ICALL for documentation
00929     uint32_t u32SP = (((uint32_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
00930                     (((uint32_t)stCPU.pstRAM->stRegisters.SPL.r));
00931
00932     uint32_t u32StoredPC = stCPU.u32PC + 2;
00933
00934     Data_Write( u32SP, (uint8_t)(u32StoredPC & 0x00FF));
00935     Data_Write( u32SP - 1, (uint8_t)(u32StoredPC >> 8));
00936
00937     u32SP -= 2;
00938
00939     uint32_t u32NewPC = stCPU.k;
00940
00941     stCPU.pstRAM->stRegisters.SPH.r = (u32SP >> 8);
00942     stCPU.pstRAM->stRegisters.SPL.r = (u32SP & 0x00FF);
00943
00944     Unconditional_Jump( u32NewPC );
00945 }
00946
00947 //-----

```

```

00948 static void AVR_Opcode_RET( void )
00949 {
00950     // Pop the next instruction off of the stack, pre-incrementing
00951     uint32_t u32SP = (((uint32_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
00952                     (((uint32_t)stCPU.pstRAM->stRegisters.SPL.r));
00953     u32SP += 2;
00954
00955     uint32_t u32High = Data_Read( u32SP - 1 );
00956     uint32_t u32Low = Data_Read( u32SP );
00957     uint32_t u32NewPC = (u32High << 8) | u32Low;
00958
00959     stCPU.pstRAM->stRegisters.SPH.r = (u32SP >> 8);
00960     stCPU.pstRAM->stRegisters.SPL.r = (u32SP & 0x00FF);
00961
00962     // Set new PC based on address read from stack
00963     Unconditional_Jump( u32NewPC );
00964 }
00965
00966 //-----
00967 static void AVR_Opcode_RETI( void )
00968 {
00969     uint32_t u32SP = (((uint32_t)stCPU.pstRAM->stRegisters.SPH.r) << 8) |
00970                     (((uint32_t)stCPU.pstRAM->stRegisters.SPL.r));
00971     u32SP += 2;
00972
00973     uint32_t u32High = Data_Read( u32SP - 1 );
00974     uint32_t u32Low = Data_Read( u32SP );
00975     uint32_t u32NewPC = (u32High << 8) | u32Low;
00976
00977     stCPU.pstRAM->stRegisters.SPH.r = (u32SP >> 8);
00978     stCPU.pstRAM->stRegisters.SPL.r = (u32SP & 0x00FF);
00979
00980     //-- Enable interrupts
00981     stCPU.pstRAM->stRegisters.SREG.I = 1;
00982     Unconditional_Jump( u32NewPC );
00983
00984     //-- Run callout functions registered when we return from interrupt.
00985     InterruptCallout_Run( false, 0 );
00986 }
00987
00988 //-----
00989 static void AVR_Opcode_CPSE( void )
00990 {
00991     if (*stCPU.Rr == *stCPU.Rd)
00992     {
00993         uint8_t u8NextOpSize = AVR_Opcode_Size( stCPU.pu16ROM[ stCPU.u32PC + 1 ] );
00994         Relative_Jump( u8NextOpSize + 1 );
00995     }
00996 }
00997
00998 //-----
00999 static void CP_Half_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_ )
01000 {
01001     stCPU.pstRAM->stRegisters.SREG.H =
01002         ( ((~Rd_ & Rr_) & 0x08 )
01003         | ((Rr_ & (Result_)) & 0x08 )
01004         | (((Result_) & ~Rd_) & 0x08) ) != false;
01005 }
01006
01007 //-----
01008 static void CP_Full_Carry( uint8_t Rd_, uint8_t Rr_, uint8_t Result_ )
01009 {
01010     stCPU.pstRAM->stRegisters.SREG.C =
01011         ( ((~Rd_ & Rr_) & 0x80 )
01012         | ((Rr_ & (Result_)) & 0x80 )
01013         | (((Result_) & ~Rd_) & 0x80) ) != false;
01014 }
01015
01016 //-----
01017 static void CP_Overflow_Flag( uint8_t Rd_, uint8_t Rr_, uint8_t Result_ )
01018 {
01019     stCPU.pstRAM->stRegisters.SREG.V =
01020         ( ((Rd_ & ~Rr_ & ~Result_) & 0x80 )
01021         | ((~Rd_ & Rr_ & Result_) & 0x80) ) != 0;
01022 }
01023
01024 //-----
01025 static void AVR_Opcode_CP( void )
01026 {
01027     // Compare
01028     uint8_t u8Result;
01029     uint8_t u8Rd = *stCPU.Rd;
01030     uint8_t u8Rr = *stCPU.Rr;
01031
01032     u8Result = u8Rd - u8Rr;
01033
01034     //--

```

```

01035     CP_Half_Carry( u8Rd, u8Rr, u8Result );
01036     CP_Overflow_Flag( u8Rd, u8Rr, u8Result );
01037     CP_Full_Carry( u8Rd, u8Rr, u8Result );
01038
01039     R8_Zero_Flag( u8Result );
01040     R8_Negative_Flag( u8Result );
01041
01042     Signed_Flag();
01043 }
01044
01045 //-----
01046 static void AVR_Opcode_CPC( void )
01047 {
01048     // Compare with carry
01049     uint8_t u8Result;
01050     uint8_t u8Rd = *stCPU.Rd;
01051     uint8_t u8Rr = *stCPU.Rr;
01052     uint8_t u8C = (stCPU.pstRAM->stRegisters.SREG.C == 1);
01053
01054     u8Result = u8Rd - u8Rr - u8C;
01055
01056     //---
01057     CP_Half_Carry( u8Rd, u8Rr, u8Result );
01058     CP_Overflow_Flag( u8Rd, u8Rr, u8Result );
01059     CP_Full_Carry( u8Rd, u8Rr, u8Result );
01060
01061     R8_CPC_Zero_Flag( u8Result );
01062     R8_Negative_Flag( u8Result );
01063
01064     Signed_Flag();
01065 }
01066
01067 //-----
01068 static void AVR_Opcode_CPI( void )
01069 {
01070     // Compare with immediate
01071     uint8_t u8Result;
01072     uint8_t u8Rd = *stCPU.Rd;
01073     uint8_t u8K = stCPU.K;
01074
01075     u8Result = u8Rd - u8K;
01076
01077     //---
01078     CP_Half_Carry( u8Rd, u8K, u8Result );
01079     CP_Overflow_Flag( u8Rd, u8K, u8Result );
01080     CP_Full_Carry( u8Rd, u8K, u8Result );
01081
01082     R8_Zero_Flag( u8Result );
01083     R8_Negative_Flag( u8Result );
01084
01085     Signed_Flag();
01086 }
01087
01088 //-----
01089 static void AVR_Opcode_SBRC( void )
01090 {
01091     // Skip if Bit in IO register clear
01092     if ((*stCPU.Rd & (1 << stCPU.b)) == 0)
01093     {
01094         uint8_t u8NextOpSize = AVR_Opcode_Size( stCPU.pu16ROM[ stCPU.u32PC + 1 ] );
01095         Relative_Jump( u8NextOpSize + 1 );
01096     }
01097 }
01098
01099 //-----
01100 static void AVR_Opcode_SBRs( void )
01101 {
01102     // Skip if Bit in IO register set
01103     if ((*stCPU.Rd & (1 << stCPU.b)) != 0)
01104     {
01105         uint8_t u8NextOpSize = AVR_Opcode_Size( stCPU.pu16ROM[ stCPU.u32PC + 1 ] );
01106         Relative_Jump( u8NextOpSize + 1 );
01107     }
01108 }
01109
01110 //-----
01111 static void AVR_Opcode_SBIC( void )
01112 {
01113     // Skip if Bit in IO register clear
01114     uint8_t u8IOVal = Data_Read( 32 + stCPU.A );
01115     if ((u8IOVal & (1 << stCPU.b)) == 0)
01116     {
01117         uint8_t u8NextOpSize = AVR_Opcode_Size( stCPU.pu16ROM[ stCPU.u32PC + 1 ] );
01118         Relative_Jump( u8NextOpSize + 1 );
01119     }
01120 }
01121

```

```

01122 //-----
01123 static void AVR_Opcode_SBIS( void )
01124 {
01125     // Skip if Bit in IO register set
01126     uint8_t u8IOVal = Data_Read( 32 + stCPU.A );
01127     if ((u8IOVal & (1 << stCPU.b)) != 0)
01128     {
01129         uint8_t u8NextOpSize = AVR_Opcode_Size( stCPU.pu16ROM[ stCPU.u32PC + 1 ] );
01130         Relative_Jump( u8NextOpSize + 1 );
01131     }
01132 }
01133
01134 //-----
01135 static void Conditional_Branch( void )
01136 {
01137     stCPU.u32PC = (uint16_t)((int16_t)stCPU.u32PC + stCPU.k_s + 1);
01138     stCPU.ul6ExtraPC = 0;
01139     stCPU.ul6ExtraCycles++;
01140 }
01141
01142 //-----
01143 static void AVR_Opcode_BRBS( void )
01144 {
01145     if (0 != (stCPU.pstRAM->stRegisters.SREG.r & (1 << stCPU.b)))
01146     {
01147         Conditional_Branch();
01148     }
01149 }
01150
01151 //-----
01152 static void AVR_Opcode_BRBC( void )
01153 {
01154     if (0 == (stCPU.pstRAM->stRegisters.SREG.r & (1 << stCPU.b)))
01155     {
01156         Conditional_Branch();
01157     }
01158 }
01159
01160 //-----
01161 static void AVR_Opcode_BREQ( void )
01162 {
01163     if (1 == stCPU.pstRAM->stRegisters.SREG.Z)
01164     {
01165         Conditional_Branch();
01166     }
01167 }
01168
01169 //-----
01170 static void AVR_Opcode_BRNE( void )
01171 {
01172     if (0 == stCPU.pstRAM->stRegisters.SREG.Z)
01173     {
01174         Conditional_Branch();
01175     }
01176 }
01177
01178 //-----
01179 static void AVR_Opcode_BRCS( void )
01180 {
01181     if (1 == stCPU.pstRAM->stRegisters.SREG.C)
01182     {
01183         Conditional_Branch();
01184     }
01185 }
01186
01187 //-----
01188 static void AVR_Opcode_BRCC( void )
01189 {
01190     if (0 == stCPU.pstRAM->stRegisters.SREG.C)
01191     {
01192         Conditional_Branch();
01193     }
01194 }
01195
01196 //-----
01197 static void AVR_Opcode_BRSH( void )
01198 {
01199     if (0 == stCPU.pstRAM->stRegisters.SREG.C)
01200     {
01201         Conditional_Branch();
01202     }
01203 }
01204
01205 //-----
01206 static void AVR_Opcode_BRLO( void )
01207 {
01208     if (1 == stCPU.pstRAM->stRegisters.SREG.C)

```

```
01209     {
01210         Conditional_Branch();
01211     }
01212 }
01213
01214 //-----
01215 static void AVR_Opcode_BRMI( void )
01216 {
01217     if (1 == stCPU.pstRAM->stRegisters.SREG.N)
01218     {
01219         Conditional_Branch();
01220     }
01221 }
01222
01223 //-----
01224 static void AVR_Opcode_BRPL( void )
01225 {
01226     if (0 == stCPU.pstRAM->stRegisters.SREG.N)
01227     {
01228         Conditional_Branch();
01229     }
01230 }
01231
01232 //-----
01233 static void AVR_Opcode_BRGE( void )
01234 {
01235     if (0 == stCPU.pstRAM->stRegisters.SREG.S)
01236     {
01237         Conditional_Branch();
01238     }
01239 }
01240
01241 //-----
01242 static void AVR_Opcode_BRLT( void )
01243 {
01244     if (1 == stCPU.pstRAM->stRegisters.SREG.S)
01245     {
01246         Conditional_Branch();
01247     }
01248 }
01249
01250 //-----
01251 static void AVR_Opcode_BRHS( void )
01252 {
01253     if (1 == stCPU.pstRAM->stRegisters.SREG.H)
01254     {
01255         Conditional_Branch();
01256     }
01257 }
01258
01259 //-----
01260 static void AVR_Opcode_BRHC( void )
01261 {
01262     if (0 == stCPU.pstRAM->stRegisters.SREG.H)
01263     {
01264         Conditional_Branch();
01265     }
01266 }
01267
01268 //-----
01269 static void AVR_Opcode_BRTS( void )
01270 {
01271     if (1 == stCPU.pstRAM->stRegisters.SREG.T)
01272     {
01273         Conditional_Branch();
01274     }
01275 }
01276
01277 //-----
01278 static void AVR_Opcode_BRTC( void )
01279 {
01280     if (0 == stCPU.pstRAM->stRegisters.SREG.T)
01281     {
01282         Conditional_Branch();
01283     }
01284 }
01285
01286 //-----
01287 static void AVR_Opcode_BRVS( void )
01288 {
01289     if (1 == stCPU.pstRAM->stRegisters.SREG.V)
01290     {
01291         Conditional_Branch();
01292     }
01293 }
01294
01295 //-----
```

```

01296 static void AVR_Opcode_BRVC( void )
01297 {
01298     if (0 == stCPU.pstRAM->stRegisters.SREG.V)
01299     {
01300         Conditional_Branch();
01301     }
01302 }
01303
01304 //-----
01305 static void AVR_Opcode_BRIE( void )
01306 {
01307     if (1 == stCPU.pstRAM->stRegisters.SREG.I)
01308     {
01309         Conditional_Branch();
01310     }
01311 }
01312
01313 //-----
01314 static void AVR_Opcode_BRID( void )
01315 {
01316     if (0 == stCPU.pstRAM->stRegisters.SREG.I)
01317     {
01318         Conditional_Branch();
01319     }
01320 }
01321
01322 //-----
01323 static void AVR_Opcode_MOV( void )
01324 {
01325     *stCPU.Rd = *stCPU.Rr;
01326 }
01327
01328 //-----
01329 static void AVR_Opcode_MOVW( void )
01330 {
01331     *stCPU.Rd16 = *stCPU.Rr16;
01332 }
01333
01334 //-----
01335 static void AVR_Opcode_LDI( void )
01336 {
01337     *stCPU.Rd = stCPU.K;
01338 }
01339
01340 //-----
01341 static void AVR_Opcode_LDS( void )
01342 {
01343     *stCPU.Rd = Data_Read( stCPU.K );
01344 }
01345
01346 //-----
01347 static void AVR_Opcode_LD_X_Indirect( void )
01348 {
01349     *stCPU.Rd =
01350         Data_Read( stCPU.pstRAM->stRegisters.CORE_REGISTERS.X );
01351 }
01352
01353 //-----
01354 static void AVR_Opcode_LD_X_Indirect_Postinc( void )
01355 {
01356     *stCPU.Rd =
01357         Data_Read( stCPU.pstRAM->stRegisters.CORE_REGISTERS.X++ );
01358 }
01359
01360 //-----
01361 static void AVR_Opcode_LD_X_Indirect_Predec( void )
01362 {
01363     *stCPU.Rd =
01364         Data_Read( --stCPU.pstRAM->stRegisters.CORE_REGISTERS.X );
01365 }
01366
01367 //-----
01368 static void AVR_Opcode_LD_Y_Indirect( void )
01369 {
01370     *stCPU.Rd =
01371         Data_Read( stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y );
01372 }
01373
01374 //-----
01375 static void AVR_Opcode_LD_Y_Indirect_Postinc( void )
01376 {
01377     *stCPU.Rd =
01378         Data_Read( stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y++ );
01379 }
01380
01381 //-----
01382 static void AVR_Opcode_LD_Y_Indirect_Predec( void )

```

```

01383 {
01384     *stCPU.Rd =
01385         Data_Read( --stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y );
01386 }
01387
01388 //-----
01389 static void AVR_Opcode_LDD_Y( void )
01390 {
01391     *stCPU.Rd =
01392         Data_Read( stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y + stCPU.q );
01393 }
01394
01395 //-----
01396 static void AVR_Opcode_LD_Z_Indirect( void )
01397 {
01398     *stCPU.Rd =
01399         Data_Read( Get_ZAddress() );
01400 }
01401
01402 //-----
01403 static void AVR_Opcode_LD_Z_Indirect_Postinc( void )
01404 {
01405     *stCPU.Rd =
01406         Data_Read( Get_ZAddressPostInc() );
01407 }
01408
01409
01410 //-----
01411 static void AVR_Opcode_LD_Z_Indirect_Preddec( void )
01412 {
01413     *stCPU.Rd =
01414         Data_Read( Get_ZAddressPreDec() );
01415 }
01416
01417 //-----
01418 static void AVR_Opcode_LDD_Z( void )
01419 {
01420     *stCPU.Rd =
01421         Data_Read( Get_ZAddress() + stCPU.q );
01422 }
01423
01424 //-----
01425 static void AVR_Opcode_STS( void )
01426 {
01427     Data_Write( stCPU.K, *stCPU.Rd );
01428 }
01429
01430 //-----
01431 static void AVR_Opcode_ST_X_Indirect( void )
01432 {
01433     Data_Write( stCPU.pstRAM->stRegisters.CORE_REGISTERS.X, *stCPU.Rd );
01434 }
01435
01436 //-----
01437 static void AVR_Opcode_ST_X_Indirect_Postinc( void )
01438 {
01439     Data_Write( stCPU.pstRAM->stRegisters.CORE_REGISTERS.X++, *stCPU.Rd );
01440 }
01441
01442 //-----
01443 static void AVR_Opcode_ST_X_Indirect_Preddec( void )
01444 {
01445     Data_Write( --stCPU.pstRAM->stRegisters.CORE_REGISTERS.X, *stCPU.Rd );
01446 }
01447
01448 //-----
01449 static void AVR_Opcode_ST_Y_Indirect( void )
01450 {
01451     Data_Write( stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y, *stCPU.Rd );
01452 }
01453
01454 //-----
01455 static void AVR_Opcode_ST_Y_Indirect_Postinc( void )
01456 {
01457     Data_Write( stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y++, *stCPU.Rd );
01458 }
01459
01460 //-----
01461 static void AVR_Opcode_ST_Y_Indirect_Preddec( void )
01462 {
01463     Data_Write( --stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y, *stCPU.Rd );
01464 }
01465
01466 //-----
01467 static void AVR_Opcode_STD_Y( void )
01468 {
01469     Data_Write( stCPU.pstRAM->stRegisters.CORE_REGISTERS.Y + stCPU.q, *stCPU.Rd );

```



```

01470 }
01471
01472 //-----
01473 static void AVR_Opcode_ST_Z_Indirect( void )
01474 {
01475     Data_Write( Get_ZAddress(), *stCPU.Rd );
01476 }
01477
01478 //-----
01479 static void AVR_Opcode_ST_Z_Indirect_Postinc( void )
01480 {
01481     Data_Write( Get_ZAddressPostInc() , *stCPU.Rd );
01482 }
01483
01484 //-----
01485 static void AVR_Opcode_ST_Z_Indirect_Predec( void )
01486 {
01487     Data_Write( Get_ZAddressPreDec() , *stCPU.Rd );
01488 }
01489
01490 //-----
01491 static void AVR_Opcode_STD_Z( void )
01492 {
01493     Data_Write( Get_ZAddress() + stCPU.q, *stCPU.Rd );
01494 }
01495
01496 //-----
01497 static void AVR_Opcode_LPM( void )
01498 {
01499     uint8_t u8Temp;
01500     if (Get_ZAddress() & 0x0001)
01501     {
01502         u8Temp = (uint8_t)(stCPU.pul6ROM[ Get_ZAddress() >> 1 ] >> 8);
01503     }
01504     else
01505     {
01506         u8Temp = (uint8_t)(stCPU.pul6ROM[ Get_ZAddress() >> 1 ] & 0x00FF);
01507     }
01508
01509     stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0 = u8Temp;
01510 }
01511
01512 //-----
01513 static void AVR_Opcode_LPM_Z( void )
01514 {
01515     uint8_t u8Temp;
01516     if (Get_ZAddress() & 0x0001)
01517     {
01518         u8Temp = (uint8_t)(stCPU.pul6ROM[ Get_ZAddress() >> 1 ] >> 8);
01519     }
01520     else
01521     {
01522         u8Temp = (uint8_t)(stCPU.pul6ROM[ Get_ZAddress() >> 1 ] & 0x00FF);
01523     }
01524
01525     *stCPU.Rd = u8Temp;
01526 }
01527
01528 //-----
01529 static void AVR_Opcode_LPM_Z_Postinc( void )
01530 {
01531     uint8_t u8Temp;
01532     if (Get_ZAddress() & 0x0001)
01533     {
01534         u8Temp = (uint8_t)(stCPU.pul6ROM[ Get_ZAddressPostInc() >> 1 ] >> 8);
01535     }
01536     else
01537     {
01538         u8Temp = (uint8_t)(stCPU.pul6ROM[ Get_ZAddressPostInc() >> 1 ] & 0x00FF);
01539     }
01540
01541     *stCPU.Rd = u8Temp;
01542 }
01543
01544 //-----
01545 static void AVR_Opcode_ELPM( void )
01546 {
01547     uint8_t u8Temp;
01548     if (Get_ZAddressWithRAMP() & 0x0001)
01549     {
01550         u8Temp = (uint8_t)(stCPU.pul6ROM[ Get_ZAddressWithRAMP() >> 1 ] >> 8);
01551     }
01552     else
01553     {
01554         u8Temp = (uint8_t)(stCPU.pul6ROM[ Get_ZAddressWithRAMP() >> 1 ] & 0x00FF);
01555     }
01556

```

```

01557     stCPU.pstRAM->stRegisters.CORE_REGISTERS.r0 = u8Temp;
01558 }
01559
01560 //-----
01561 static void AVR_Opcode_ELPM_Z( void )
01562 {
01563     uint8_t u8Temp;
01564     if (Get_ZAddressWithRAMP() & 0x0001)
01565     {
01566         u8Temp = (uint8_t)(stCPU.pul6ROM[ Get_ZAddressWithRAMP() >> 1 ] >> 8);
01567     }
01568     else
01569     {
01570         u8Temp = (uint8_t)(stCPU.pul6ROM[ Get_ZAddressWithRAMP() >> 1 ] & 0x00FF);
01571     }
01572
01573     *stCPU.Rd = u8Temp;
01574 }
01575
01576 //-----
01577 static void AVR_Opcode_ELPM_Z_Postinc( void )
01578 {
01579     uint8_t u8Temp;
01580     if (Get_ZAddressWithRAMP() & 0x0001)
01581     {
01582         u8Temp = (uint8_t)(stCPU.pul6ROM[ Get_ZAddressPostIncWithRAMP() >> 1 ] >> 8);
01583     }
01584     else
01585     {
01586         u8Temp = (uint8_t)(stCPU.pul6ROM[ Get_ZAddressPostIncWithRAMP() >> 1 ] & 0x00FF);
01587     }
01588
01589     *stCPU.Rd = u8Temp;
01590
01591 }
01592
01593 //-----
01594 static void AVR_Opcode_SPM( void )
01595 {
01596 }
01597 }
01598
01599 //-----
01600 static void AVR_Opcode_SPM_Z_Postinc2( void )
01601 {
01602 }
01603 }
01604
01605 //-----
01606 static void AVR_Opcode_IN( void )
01607 {
01608     *stCPU.Rd = Data_Read( 32 + stCPU.A );
01609 }
01610
01611 //-----
01612 static void AVR_Opcode_OUT( void )
01613 {
01614     Data_Write( 32 + stCPU.A , *stCPU.Rd );
01615 }
01616
01617 //-----
01618 static void AVR_Opcode_PUSH( void )
01619 {
01620     uint32_t u32SP = (stCPU.pstRAM->stRegisters.SPL.r) |
01621                     ((uint32_t)(stCPU.pstRAM->stRegisters.SPH.r) << 8);
01622
01623     // Store contents from SP to destination register
01624     Data_Write( u32SP, *stCPU.Rd );
01625
01626     // Postdecrement the SP
01627     u32SP--;
01628
01629     // Update the SP registers
01630     stCPU.pstRAM->stRegisters.SPH.r = (uint8_t)(u32SP >> 8);
01631     stCPU.pstRAM->stRegisters.SPL.r = (uint8_t)(u32SP & 0x00FF);
01632 }
01633
01634 //-----
01635 static void AVR_Opcode_POP( void )
01636 {
01637     // Preincrement the SP
01638     uint32_t u32SP = (stCPU.pstRAM->stRegisters.SPL.r) |
01639                     ((uint16_t)(stCPU.pstRAM->stRegisters.SPH.r) << 8);
01640     u32SP++;
01641
01642     // Load contents from SP to destination register
01643     *stCPU.Rd = Data_Read( u32SP );
01644
01645     // Update the SP registers

```

```

01646     stCPU.pstRAM->stRegisters.SPH.r = (uint8_t)(u32SP >> 8);
01647     stCPU.pstRAM->stRegisters.SPL.r = (uint8_t)(u32SP & 0x00FF);
01648 }
01649
01650 //-----
01651 static void AVR_Opcode_XCH( void )
01652 {
01653     uint8_t u8Z;
01654     uint8_t u8Temp;
01655     uint32_t u32Addr = Get_ZAddress();
01656
01657     u8Z = Data_Read( u32Addr );
01658     u8Temp = *stCPU.Rd;
01659
01660     *stCPU.Rd = u8Z;
01661     Data_Write( u32Addr, u8Temp );
01662 }
01663
01664 //-----
01665 static void AVR_Opcode_LAS( void )
01666 {
01667     uint8_t u8Z;
01668     uint8_t u8Temp;
01669
01670     uint32_t u32Addr = Get_ZAddress();
01671
01672     u8Z = Data_Read( u32Addr );
01673     u8Temp = *stCPU.Rd | u8Z;
01674
01675     *stCPU.Rd = u8Z;
01676     Data_Write( u32Addr, u8Temp );
01677 }
01678
01679 //-----
01680 static void AVR_Opcode_LAC( void )
01681 {
01682     uint8_t u8Z;
01683     uint8_t u8Temp;
01684
01685     uint32_t u32Addr = Get_ZAddress();
01686
01687     u8Z = Data_Read( u32Addr );
01688     u8Temp = *stCPU.Rd & ~(u8Z);
01689     *stCPU.Rd = u8Z;
01690
01691     Data_Write( u32Addr, u8Temp );
01692 }
01693
01694 //-----
01695 static void AVR_Opcode_LAT( void )
01696 {
01697     uint8_t u8Z;
01698     uint8_t u8Temp;
01699
01700     uint32_t u32Addr = Get_ZAddress();
01701
01702     u8Z = Data_Read( u32Addr );
01703     u8Temp = *stCPU.Rd ^ u8Z;
01704     *stCPU.Rd = u8Z;
01705
01706     Data_Write( u32Addr, u8Temp );
01707 }
01708
01709 //-----
01710 static void LSL_HalfCarry_Flag( uint8_t R_ )
01711 {
01712     stCPU.pstRAM->stRegisters.SREG.H = ((R_ & 0x08) == 0x08);
01713 }
01714
01715 //-----
01716 static void Left_Carry_Flag( uint8_t R_ )
01717 {
01718     stCPU.pstRAM->stRegisters.SREG.C = ((R_ & 0x80) == 0x80);
01719 }
01720
01721 //-----
01722 static void Rotate_Overflow_Flag()
01723 {
01724     stCPU.pstRAM->stRegisters.SREG.V = ( stCPU.pstRAM->stRegisters.SREG.N ^ stCPU.pstRAM->stRegisters.SREG.
01725 C );
01726 }
01727
01728 //-----
01729 static void AVR_Opcode_LSL( void )
01730 {
01731     // Logical shift left
01732     uint8_t u8Result = 0;

```

```

01732     uint8_t u8Temp = *stCPU.Rd;
01733
01734     u8Result = (u8Temp << 1);
01735     *stCPU.Rd = u8Result;
01736
01737     // ---- Update flags ----
01738     LSL_HalfCarry_Flag( u8Result);
01739     Left_Carry_Flag( u8Temp);
01740
01741     R8_Negative_Flag( u8Result );
01742     R8_Zero_Flag( u8Result );
01743     Rotate_Overflow_Flag();
01744     Signed_Flag();
01745 }
01746
01747 //-----
01748 static void Right_Carry_Flag( uint8_t R_ )
01749 {
01750     stCPU.pstRAM->stRegisters.SREG.C = ((R_ & 0x01) == 0x01);
01751 }
01752
01753 //-----
01754 static void AVR_Opcode_LSR( void )
01755 {
01756     // Logical shift left
01757     uint8_t u8Result = 0;
01758     uint8_t u8Temp = *stCPU.Rd;
01759
01760     u8Result = (u8Temp >> 1);
01761     *stCPU.Rd = u8Result;
01762
01763     // ---- Update flags ----
01764     Right_Carry_Flag( u8Temp );
01765     stCPU.pstRAM->stRegisters.SREG.N = 0;
01766     R8_Zero_Flag( u8Result );
01767     Rotate_Overflow_Flag();
01768     Signed_Flag();
01769 }
01770
01771 //-----
01772 static void AVR_Opcode_ROL( void )
01773 {
01774     // Rotate left through carry
01775     uint8_t u8Result = 0;
01776     uint8_t u8Temp = *stCPU.Rd;
01777
01778     u8Result = (u8Temp << 1);
01779     if (stCPU.pstRAM->stRegisters.SREG.C)
01780     {
01781         u8Result |= 0x01;
01782     }
01783     *stCPU.Rd = u8Result;
01784
01785     // ---- Update flags ----
01786     Left_Carry_Flag( u8Temp );
01787     R8_Negative_Flag( u8Result );
01788     R8_Zero_Flag( u8Result );
01789     Rotate_Overflow_Flag();
01790     Signed_Flag();
01791 }
01792
01793 //-----
01794 static void AVR_Opcode_ROR( void )
01795 {
01796     // Rotate right through carry
01797     uint8_t u8Result = 0;
01798     uint8_t u8Temp = *stCPU.Rd;
01799
01800     u8Result = (u8Temp >> 1);
01801     if (stCPU.pstRAM->stRegisters.SREG.C)
01802     {
01803         u8Result |= 0x80;
01804     }
01805     *stCPU.Rd = u8Result;
01806
01807     // ---- Update flags ----
01808     Right_Carry_Flag( u8Temp );
01809     R8_Negative_Flag( u8Result );
01810     R8_Zero_Flag( u8Result );
01811     Rotate_Overflow_Flag();
01812     Signed_Flag();
01813 }
01814
01815 //-----
01816 static void AVR_Opcode_ASR( void )
01817 {
01818     // Shift all bits to the right, keeping sign bit intact

```

```

01819     uint8_t u8Result;
01820     uint8_t u8Temp = *stCPU.Rd;
01821     u8Result = (u8Temp & 0x80) | (u8Temp >> 1);
01822     *stCPU.Rd = u8Result;
01823
01824     // ---- Update flags ----
01825     Right_Carry_Flag( u8Temp );
01826     R8_Negative_Flag( u8Result );
01827     R8_Zero_Flag( u8Result );
01828     Rotate_Overflow_Flag();
01829     Signed_Flag();
01830 }
01831
01832 //-----
01833 static void AVR_Opcode_SWAP( void )
01834 {
01835     uint8_t u8temp;
01836     u8temp = ((*stCPU.Rd) >> 4) |
01837             ((*stCPU.Rd) << 4);
01838
01839     *stCPU.Rd = u8temp;
01840 }
01841
01842 //-----
01843 static void AVR_Opcode_BSET( void )
01844 {
01845     stCPU.pstRAM->stRegisters.SREG.r |= (1 << stCPU.b);
01846 }
01847
01848 //-----
01849 static void AVR_Opcode_BCLR( void )
01850 {
01851     stCPU.pstRAM->stRegisters.SREG.r &= ~(1 << stCPU.b);
01852 }
01853
01854 //-----
01855 static void AVR_Opcode_SBI( void )
01856 {
01857     uint8_t u8Temp = Data_Read( stCPU.A + 32 );
01858     u8Temp |= (1 << stCPU.b);
01859     Data_Write( stCPU.A + 32, u8Temp );
01860 }
01861
01862 //-----
01863 static void AVR_Opcode_CBI( void )
01864 {
01865     uint8_t u8Temp = Data_Read( stCPU.A + 32 );
01866     u8Temp &= ~(1 << stCPU.b);
01867     Data_Write( stCPU.A + 32, u8Temp );
01868 }
01869
01870 //-----
01871 static void AVR_Opcode_BST( void )
01872 {
01873     if ((*stCPU.Rd) & (1 << stCPU.b))
01874     {
01875         stCPU.pstRAM->stRegisters.SREG.T = 1;
01876     }
01877     else
01878     {
01879         stCPU.pstRAM->stRegisters.SREG.T = 0;
01880     }
01881 }
01882
01883 //-----
01884 static void AVR_Opcode_BLD( void )
01885 {
01886     if (stCPU.pstRAM->stRegisters.SREG.T)
01887     {
01888         *stCPU.Rd |= (1 << stCPU.b);
01889     }
01890     else
01891     {
01892         *stCPU.Rd &= ~(1 << stCPU.b);
01893     }
01894 }
01895
01896 //-----
01897 static void AVR_Opcode_BREAK( void )
01898 {
01899     // Unimplemented - since this requires debugging HW...
01900 }
01901
01902 //-----
01903 static void AVR_Opcode_SLEEP( void )
01904 {
01905     stCPU.bAsleep = true;

```

```

01906 }
01907
01908 //-----
01909 static void AVR_Opcode_WDR( void )
01910 {
01911     stCPU.u32WDTCnt = 0;    // Reset watchdog timer counter
01912 }
01913
01914 //-----
01915 AVR_Opcode AVR_Opcode_Function( uint16_t OP_ )
01916 {
01917     switch (OP_)
01918     {
01919         case 0x0000: return AVR_Opcode_NOP;
01920
01921         case 0x9409: return AVR_Opcode_IJMP;
01922         case 0x9419: return AVR_Opcode_EIJMP;
01923
01924         case 0x9508: return AVR_Opcode_RET;
01925         case 0x9509: return AVR_Opcode_ICALL;
01926         case 0x9518: return AVR_Opcode_RETI;
01927         case 0x9519: return AVR_Opcode_EICALL;
01928         case 0x9588: return AVR_Opcode_SLEEP;
01929         case 0x9598: return AVR_Opcode_BREAK;
01930         case 0x95A8: return AVR_Opcode_WDR;
01931         case 0x95C8: return AVR_Opcode_LPM;
01932         case 0x95D8: return AVR_Opcode_ELPM;
01933         case 0x95E8: return AVR_Opcode_SPM;
01934         case 0x95F8: return AVR_Opcode_SPM_Z_Postinc2;
01935     }
01936
01937     switch ( OP_ & 0xFF8F )
01938     {
01939         case 0x9408: return AVR_Opcode_BSET;
01940         case 0x9488: return AVR_Opcode_BCLR;
01941     }
01942
01943     switch ( OP_ & 0xFF88 )
01944     {
01945         case 0x0300: return AVR_Opcode_MULSU;
01946         case 0x0308: return AVR_Opcode_Fmul;
01947         case 0x0380: return AVR_Opcode_FmulS;
01948         case 0x0388: return AVR_Opcode_FmulSU;
01949     }
01950
01951     switch ( OP_ & 0xFF0F )
01952     {
01953         case 0x940B: return AVR_Opcode_DES;
01954         case 0xEF0F: return AVR_Opcode_SER;
01955     }
01956
01957     switch ( OP_ & 0xFF00 )
01958     {
01959         case 0x0100: return AVR_Opcode_MOVW;
01960         case 0x9600: return AVR_Opcode_ADIW;
01961         case 0x9700: return AVR_Opcode_SBIW;
01962
01963         case 0x9800: return AVR_Opcode_CBI;
01964         case 0x9900: return AVR_Opcode_SBIC;
01965         case 0x9A00: return AVR_Opcode_SBI;
01966         case 0x9B00: return AVR_Opcode_SBIS;
01967     }
01968
01969     switch ( OP_ & 0xFE0F )
01970     {
01971         case 0x8008: return AVR_Opcode_LD_Y_Indirect;
01972         case 0x8000: return AVR_Opcode_LD_Z_Indirect;
01973         case 0x8200: return AVR_Opcode_ST_Z_Indirect;
01974         case 0x8208: return AVR_Opcode_ST_Y_Indirect;
01975
01976         // -- Single 5-bit register...
01977         case 0x9000: return AVR_Opcode_LDS;
01978         case 0x9001: return AVR_Opcode_LD_Z_Indirect_Postinc;
01979         case 0x9002: return AVR_Opcode_LD_Z_Indirect_Predec;
01980         case 0x9004: return AVR_Opcode_LPM_Z;
01981         case 0x9005: return AVR_Opcode_LPM_Z_Postinc;
01982         case 0x9006: return AVR_Opcode_ELPM_Z;
01983         case 0x9007: return AVR_Opcode_ELPM_Z_Postinc;
01984         case 0x9009: return AVR_Opcode_LD_Y_Indirect_Postinc;
01985         case 0x900A: return AVR_Opcode_LD_Y_Indirect_Predec;
01986         case 0x900C: return AVR_Opcode_LD_X_Indirect;
01987         case 0x900D: return AVR_Opcode_LD_X_Indirect_Postinc;
01988         case 0x900E: return AVR_Opcode_LD_X_Indirect_Predec;
01989         case 0x900F: return AVR_Opcode_POP;
01990
01991         case 0x9200: return AVR_Opcode_STS;
01992         case 0x9201: return AVR_Opcode_ST_Z_Indirect_Postinc;

```

```

01993     case 0x9202: return AVR_Opcode_ST_Z_Indirect_Predec;
01994     case 0x9204: return AVR_Opcode_XCH;
01995     case 0x9205: return AVR_Opcode_LAS;
01996     case 0x9206: return AVR_Opcode_LAC;
01997     case 0x9207: return AVR_Opcode_LAT;
01998     case 0x9209: return AVR_Opcode_ST_Y_Indirect_Postinc;
01999     case 0x920A: return AVR_Opcode_ST_Y_Indirect_Predec;
02000     case 0x920C: return AVR_Opcode_ST_X_Indirect;
02001     case 0x920D: return AVR_Opcode_ST_X_Indirect_Postinc;
02002     case 0x920E: return AVR_Opcode_ST_X_Indirect_Predec;
02003     case 0x920F: return AVR_Opcode_PUSH;
02004
02005     // -- One-operand instructions
02006     case 0x9400: return AVR_Opcode_COM;
02007     case 0x9401: return AVR_Opcode_NEG;
02008     case 0x9402: return AVR_Opcode_SWAP;
02009     case 0x9403: return AVR_Opcode_INC;
02010     case 0x9405: return AVR_Opcode_ASR;
02011     case 0x9406: return AVR_Opcode_LSR;
02012     case 0x9407: return AVR_Opcode_ROR;
02013     case 0x940A: return AVR_Opcode_DEC;
02014
02015     }
02016     switch (OP_ & 0xFE0E)
02017     {
02018     case 0x940C: return AVR_Opcode_JMP;
02019     case 0x940E: return AVR_Opcode_CALL;
02020     }
02021
02022     switch (OP_ & 0xFE08)
02023     {
02024
02025     // -- BLD/BST Encoding
02026     case 0xF800: return AVR_Opcode_BLD;
02027     case 0xFA00: return AVR_Opcode_BST;
02028     // -- SBRC/SBRS Encoding
02029     case 0xFC00: return AVR_Opcode_SBRC;
02030     case 0xFE00: return AVR_Opcode_SBRS;
02031     }
02032
02033     switch (OP_ & 0xFC07)
02034     {
02035     // -- Conditional branches
02036     case 0xF000: return AVR_Opcode_BRCS;
02037     // case 0xF000: return AVR_Opcode_BRLO;           // AKA AVR_Opcode_BRCS;
02038     case 0xF001: return AVR_Opcode_BREQ;
02039     case 0xF002: return AVR_Opcode_BRMI;
02040     case 0xF003: return AVR_Opcode_BRVS;
02041     case 0xF004: return AVR_Opcode_BRLT;
02042     case 0xF006: return AVR_Opcode_BRTS;
02043     case 0xF007: return AVR_Opcode_BRIE;
02044     case 0xF400: return AVR_Opcode_BRCC;
02045     // case 0xF400: return AVR_Opcode_BRSH;           // AKA AVR_Opcode_BRCC;
02046     case 0xF401: return AVR_Opcode_BRNE;
02047     case 0xF402: return AVR_Opcode_BRPL;
02048     case 0xF403: return AVR_Opcode_BRVC;
02049     case 0xF404: return AVR_Opcode_BRGE;
02050     case 0xF405: return AVR_Opcode_BRHC;
02051     case 0xF406: return AVR_Opcode_BRTC;
02052     case 0xF407: return AVR_Opcode_BRID;
02053     }
02054
02055     switch (OP_ & 0xFC00)
02056     {
02057     // -- 4-bit register pair
02058     case 0x0200: return AVR_Opcode_MULS;
02059
02060     // -- 5-bit register pairs --
02061     case 0x0400: return AVR_Opcode_CPC;
02062     case 0x0800: return AVR_Opcode_SBC;
02063     case 0x0C00: return AVR_Opcode_ADD;
02064     // case 0x0C00: return AVR_Opcode_LSL; (!! Implemented with: " add rd, rd"
02065     case 0x1000: return AVR_Opcode_CPSE;
02066     case 0x1300: return AVR_Opcode_ROL;
02067     case 0x1400: return AVR_Opcode_CP;
02068     case 0x1C00: return AVR_Opcode_ADC;
02069     case 0x1800: return AVR_Opcode_SUB;
02070     case 0x2000: return AVR_Opcode_AND;
02071     // case 0x2000: return AVR_Opcode_TST; (!! Implemented with: " and rd, rd"
02072     case 0x2400: return AVR_Opcode_EOR;
02073     case 0x2C00: return AVR_Opcode_MOV;
02074     case 0x2800: return AVR_Opcode_OR;
02075
02076     // -- 5-bit register pairs -- Destination = R1:R0
02077     case 0x9C00: return AVR_Opcode_MUL;
02078     }
02079

```

```

02080     switch (OP_ & 0xF800)
02081     {
02082     case 0xB800: return AVR_Opcode_OUT;
02083     case 0xB000: return AVR_Opcode_IN;
02084     }
02085
02086     switch (OP_ & 0xF000)
02087     {
02088     // -- Register immediate --
02089     case 0x3000: return AVR_Opcode_CPI;
02090     case 0x4000: return AVR_Opcode_SBCI;
02091     case 0x5000: return AVR_Opcode_SUBI;
02092     case 0x6000: return AVR_Opcode_ORI; // return AVR_Opcode_SBR;
02093     case 0x7000: return AVR_Opcode_ANDI;
02094
02095     //-- 12-bit immediate
02096     case 0xC000: return AVR_Opcode_RJMP;
02097     case 0xD000: return AVR_Opcode_RCALL;
02098
02099     // -- Register immediate
02100     case 0xE000: return AVR_Opcode_LDI;
02101     }
02102
02103     switch (OP_ & 0xD208)
02104     {
02105     // -- 7-bit signed offset
02106     case 0x8000: return AVR_Opcode_LDD_Z;
02107     case 0x8008: return AVR_Opcode_LDD_Y;
02108     case 0x8200: return AVR_Opcode_STD_Z;
02109     case 0x8208: return AVR_Opcode_STD_Y;
02110     }
02111
02112     return AVR_Opcode_NOP;
02113 }
02114
02115 //-----
02116 void AVR_RunOpcode( uint16_t OP_ )
02117 {
02118     AVR_Opcode myOpcode = AVR_Opcode_Function( OP_ );
02119     myOpcode();
02120 }

```

## 4.37 src/avr\_cpu/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.*

#### 4.37.1 Detailed Description

AVR CPU - Opcode interface.

Definition in file [avr\\_opcodes.h](#).



### 4.37.2 Function Documentation

#### 4.37.2.1 AVR Opcode\_Function()

```
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

$O \leftarrow$ $P \leftarrow$ —	Opcode to return an "opcode execution" function pointer for
---------------------------------------	---

##### Returns

Opcode execution function pointer corresponding to the given opcode.

Definition at line 1915 of file [avr\\_opcodes.c](#).

#### 4.37.2.2 AVR\_RunOpcode()

```
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

$O \leftarrow$ $P \leftarrow$ —	Opcode to execute
---------------------------------------	-------------------

Definition at line 2116 of file [avr\\_opcodes.c](#).

## 4.38 avr\_opcodes.h

00001 /\*\*\*\*\*

```

00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      )\ ) |
00004 *      ((/( ((/(      \      ( ((/( | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( ) \ ) \ / ( ) | -- [ Little ] -----
00006 *      ( ) _ ( ) ) \ _ \ ( ( ( ( ) _ | -- [ AVR ] -----
00007 *      | _ | | _ ( ) _ \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ / _ \ \ V / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ / _ \ \ \ / | _ \ |
00010 *      | _ | | _ / _ \ \ \ / | _ \ |
00011 *      | _ | | _ / _ \ \ \ / | _ \ | "Yeah, it does Arduino..."
00012 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 * *****/
00021 #ifndef __AVR_OPCODES_H__
00022 #define __AVR_OPCODES_H__
00023
00024 #include <stdint.h>
00025 #include "avr_cpu.h"
00026
00027 //-----
00028 // Format opcode function jump table
00029 typedef void (*AVR_Opcode)( void );
00030 //-----
00040 AVR_Opcode AVR_Opcode_Function( uint16_t OP_ );
00041
00042 //-----
00052 void AVR_RunOpcode( uint16_t OP_ );
00053
00054 #endif

```

## 4.39 src/avr\_cpu/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-purpose registers (R0-R31), and 224 bytes for the remaining I/O registers (used by peripherals).*

### 4.39.1 Detailed Description

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

Definition in file [avr\\_registerfile.h](#).

## 4.40 avr\_registerfile.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      )\ ) |
00004 *      ((/( ((/(      \      ( ((/( | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( ) \ ) \ / ( ) | -- [ Little ] -----
00006 *      ( ) _ ( ) ) \ _ \ ( ( ( ( ) _ | -- [ AVR ] -----
00007 *      | _ | | _ ( ) _ \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ / _ \ \ V / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ / _ \ \ \ / | _ \ |
00010 *      | _ | | _ / _ \ \ \ / | _ \ |
00011 *      | _ | | _ / _ \ \ \ / | _ \ | "Yeah, it does Arduino..."
00012 * -----+-----

```

```

00012  * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013  *   See license.txt for details
00014  *****/
00021  #ifndef __AVR_REGISTERFILE_H__
00022  #define __AVR_REGISTERFILE_H__
00023
00024  //-----
00025  #include "avr_coreregs.h"
00026  #include "avr_periphregs.h"
00027
00028  //-----
00038  typedef struct
00039  {
00040      //-- 0x00
00041      AVR_CoreRegisters CORE_REGISTERS;
00042
00043      //-- 0x20
00044      AVR_PIN PINA;
00045      AVR_DDR DDRA;
00046      AVR_PORT PORTA;
00047
00048      //-- 0x23
00049      AVR_PIN PINB;
00050      AVR_DDR DDRB;
00051      AVR_PORT PORTB;
00052
00053      //-- 0x26
00054      AVR_PIN PINC;
00055      AVR_DDR DDRC;
00056      AVR_PORT PORTC;
00057
00058      //-- 0x29
00059      AVR_PIN PIND;
00060      AVR_DDR DDRD;
00061      AVR_PORT PORTD;
00062
00063      //-- 0x2C
00064      uint8_t RESERVED_0x2C;
00065      uint8_t RESERVED_0x2D;
00066      uint8_t RESERVED_0x2E;
00067      uint8_t RESERVED_0x2F;
00068      uint8_t RESERVED_0x30;
00069      uint8_t RESERVED_0x31;
00070      uint8_t RESERVED_0x32;
00071      uint8_t RESERVED_0x33;
00072      uint8_t RESERVED_0x34;
00073
00074      //-- 0x35
00075      AVR_TIFR0 TIFR0;
00076      AVR_TIFR1 TIFR1;
00077      AVR_TIFR2 TIFR2;
00078
00079      //-- 0x38
00080      uint8_t RESERVED_0x38;
00081      uint8_t RESERVED_0x39;
00082      uint8_t RESERVED_0x3A;
00083
00084      //-- 0x3B
00085      AVR_PCIFR PCIFR;
00086      AVR_EIFR EIFR;
00087      AVR_EIMSK EIMSK;
00088
00089      //-- 0x3E
00090      uint8_t GPIOR0;
00091
00092      //-- 0x3F
00093      AVR_EECR EECR;
00094
00095      //-- 0x40
00096      uint8_t EEDR;
00097      uint8_t EEARL;
00098      uint8_t EEARH;
00099
00100      //-- 0x43
00101      AVR_GTCCR GTCCR;
00102      AVR_TCCR0A TCCR0A;
00103      AVR_TCCR0B TCCR0B;
00104      uint8_t TCNT0;
00105      uint8_t OCR0A;
00106      uint8_t OCR0B;
00107
00108      //-- 0x49
00109      uint8_t RESERVED_0x49;
00110      uint8_t GPIOR1;
00111      uint8_t GPIOR2;
00112
00113      AVR_SPCR SPCR;

```

```

00114     AVR_SPSR      SPSR;
00115     uint8_t        SPDR;
00116
00117     uint8_t        RESERVED_0x4F;
00118     AVR_ACSR       ACSR;
00119
00120     uint8_t        RESERVED_0x51;
00121     uint8_t        RESERVED_0x52;
00122
00123     //-- 0x53
00124     AVR_SMCR       SMCR;
00125     AVR_MCUSR      MCUSR;
00126     AVR_MCUCR      MCUCR;
00127     uint8_t        RESERVED_0x56;
00128
00129     AVR_SPMCSR     SPMCSR;
00130     uint8_t        RESERVED_0x58;
00131     uint8_t        RESERVED_0x59;
00132     uint8_t        RESERVED_0x5A;
00133     uint8_t        RAMPZ;
00134     uint8_t        RESERVED_0x5C;
00135     AVR_SPL        SPL;
00136     AVR_SPH        SPH;
00137     AVR_SREG       SREG;
00138
00139     //-- 0x60
00140     AVR_WDTCSR     WDTCSR;
00141     AVR_CLKPR      CLKPR;
00142     uint8_t        RESERVED_0x62;
00143     uint8_t        RESERVED_0x63;
00144     AVR_PRR        PRR;
00145     uint8_t        RESERVED_0x65;
00146     uint8_t        OSCCAL;
00147     uint8_t        RESERVED_0x67;
00148
00149     AVR_PCICR      PCICR;
00150     AVR_EICRA      EICRA;
00151     uint8_t        RESERVED_0x6A;
00152
00153     AVR_PCMSK0     PCMSK0;
00154     AVR_PCMSK1     PCMSK1;
00155     AVR_PCMSK2     PCMSK2;
00156     AVR_TIMSK0     TIMSK0;
00157     AVR_TIMSK1     TIMSK1;
00158     AVR_TIMSK2     TIMSK2;
00159
00160     uint8_t        RESERVED_0x71;
00161     uint8_t        RESERVED_0x72;
00162     uint8_t        RESERVED_0x73;
00163     uint8_t        RESERVED_0x74;
00164     uint8_t        RESERVED_0x75;
00165     uint8_t        RESERVED_0x76;
00166     uint8_t        RESERVED_0x77;
00167
00168     uint8_t        ADCL;
00169     uint8_t        ADCH;
00170     AVR_ADCSRA     ADSRA;
00171     AVR_ADCSRB     ADSRB;
00172     AVR_ADMUX      ADMUX;
00173     uint8_t        RESERVED_0x7F;
00174
00175     AVR_DIDR0      DIDR0;
00176     AVR_DIDR1      DIDR1;
00177     AVR_TCCR1A     TCCR1A;
00178     AVR_TCCR1B     TCCR1B;
00179     AVR_TCCR1C     TCCR1C;
00180     uint8_t        RESERVED_0x83;
00181
00182     uint8_t        TCNT1L;
00183     uint8_t        TCNT1H;
00184     uint8_t        ICR1L;
00185     uint8_t        ICR1H;
00186     uint8_t        OCR1AL;
00187     uint8_t        OCR1AH;
00188     uint8_t        OCR1BL;
00189     uint8_t        OCR1BH;
00190
00191     uint8_t        RESERVED_0x8C;
00192     uint8_t        RESERVED_0x8D;
00193     uint8_t        RESERVED_0x8E;
00194     uint8_t        RESERVED_0x8F;
00195
00196     uint8_t        RESERVED_0x90;
00197     uint8_t        RESERVED_0x91;
00198     uint8_t        RESERVED_0x92;
00199     uint8_t        RESERVED_0x93;
00200     uint8_t        RESERVED_0x94;

```

```

00201     uint8_t      RESERVED_0x95;
00202     uint8_t      RESERVED_0x96;
00203     uint8_t      RESERVED_0x97;
00204     uint8_t      RESERVED_0x98;
00205     uint8_t      RESERVED_0x99;
00206     uint8_t      RESERVED_0x9A;
00207     uint8_t      RESERVED_0x9B;
00208     uint8_t      RESERVED_0x9C;
00209     uint8_t      RESERVED_0x9D;
00210     uint8_t      RESERVED_0x9E;
00211     uint8_t      RESERVED_0x9F;
00212
00213     uint8_t      RESERVED_0xA0;
00214     uint8_t      RESERVED_0xA1;
00215     uint8_t      RESERVED_0xA2;
00216     uint8_t      RESERVED_0xA3;
00217     uint8_t      RESERVED_0xA4;
00218     uint8_t      RESERVED_0xA5;
00219     uint8_t      RESERVED_0xA6;
00220     uint8_t      RESERVED_0xA7;
00221     uint8_t      RESERVED_0xA8;
00222     uint8_t      RESERVED_0xA9;
00223     uint8_t      RESERVED_0xAA;
00224     uint8_t      RESERVED_0xAB;
00225     uint8_t      RESERVED_0xAC;
00226     uint8_t      RESERVED_0xAD;
00227     uint8_t      RESERVED_0xAE;
00228     uint8_t      RESERVED_0xAF;
00229
00230     //--0xB0
00231     AVR_TCCR2A    TCCR2A;
00232     AVR_TCCR2B    TCCR2B;
00233     uint8_t      TCNT2;
00234     uint8_t      OCR2A;
00235     uint8_t      OCR2B;
00236
00237     uint8_t      RESERVED_0xB5;
00238     AVR_ASSR      ASSR;
00239     uint8_t      RESERVED_0xB7;
00240     uint8_t      TWBR;
00241     AVR_TWSR      TWSR;
00242     AVR_TWAR      TWAR;
00243     uint8_t      TWDR;
00244     AVR_TWCR      TWCR;
00245     AVR_TWAMR     TWAMR;
00246
00247     uint8_t      RESERVED_0xBE;
00248     uint8_t      RESERVED_0xBF;
00249
00250     //--0xC0
00251     AVR_UCSR0A    UCSR0A;
00252     AVR_UCSR0B    UCSR0B;
00253     AVR_UCSR0C    UCSR0C;
00254
00255     uint8_t      RESERVED_0xC3;
00256
00257     uint8_t      UBRR0L;
00258     uint8_t      UBRR0H;
00259     uint8_t      UDR0;
00260
00261     uint8_t      RESERVED_0xC7;
00262     uint8_t      RESERVED_0xC8;
00263     uint8_t      RESERVED_0xC9;
00264     uint8_t      RESERVED_0xCA;
00265     uint8_t      RESERVED_0xCB;
00266     uint8_t      RESERVED_0xCC;
00267     uint8_t      RESERVED_0xCD;
00268     uint8_t      RESERVED_0xCE;
00269     uint8_t      RESERVED_0xCF;
00270
00271     uint8_t      RESERVED_0xD0;
00272     uint8_t      RESERVED_0xD1;
00273     uint8_t      RESERVED_0xD2;
00274     uint8_t      RESERVED_0xD3;
00275     uint8_t      RESERVED_0xD4;
00276     uint8_t      RESERVED_0xD5;
00277     uint8_t      RESERVED_0xD6;
00278     uint8_t      RESERVED_0xD7;
00279     uint8_t      RESERVED_0xD8;
00280     uint8_t      RESERVED_0xD9;
00281     uint8_t      RESERVED_0xDA;
00282     uint8_t      RESERVED_0xDB;
00283     uint8_t      RESERVED_0xDC;
00284     uint8_t      RESERVED_0xDD;
00285     uint8_t      RESERVED_0xDE;
00286     uint8_t      RESERVED_0xDF;
00287

```

```

00288     uint8_t    RESERVED_0xE0;
00289     uint8_t    RESERVED_0xE1;
00290     uint8_t    RESERVED_0xE2;
00291     uint8_t    RESERVED_0xE3;
00292     uint8_t    RESERVED_0xE4;
00293     uint8_t    RESERVED_0xE5;
00294     uint8_t    RESERVED_0xE6;
00295     uint8_t    RESERVED_0xE7;
00296     uint8_t    RESERVED_0xE8;
00297     uint8_t    RESERVED_0xE9;
00298     uint8_t    RESERVED_0xEA;
00299     uint8_t    RESERVED_0xEB;
00300     uint8_t    RESERVED_0xEC;
00301     uint8_t    RESERVED_0xED;
00302     uint8_t    RESERVED_0xEE;
00303     uint8_t    RESERVED_0xEF;
00304
00305     uint8_t    RESERVED_0xF0;
00306     uint8_t    RESERVED_0xF1;
00307     uint8_t    RESERVED_0xF2;
00308     uint8_t    RESERVED_0xF3;
00309     uint8_t    RESERVED_0xF4;
00310     uint8_t    RESERVED_0xF5;
00311     uint8_t    RESERVED_0xF6;
00312     uint8_t    RESERVED_0xF7;
00313     uint8_t    RESERVED_0xF8;
00314     uint8_t    RESERVED_0xF9;
00315     uint8_t    RESERVED_0xFA;
00316     uint8_t    RESERVED_0xFB;
00317     uint8_t    RESERVED_0xFC;
00318     uint8_t    RESERVED_0xFD;
00319     uint8_t    RESERVED_0xFE;
00320     uint8_t    RESERVED_0xFF;
00321
00322 } AVRRegisterFile;
00323
00324
00325 #endif // __AVR_REGISTERFILE_H__

```

## 4.41 src/avr\_cpu/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

### 4.41.1 Detailed Description

Module providing functionality allowing emulator extensions to be triggered on interrupts.

Definition in file [interrupt\\_callout.c](#).

### 4.41.2 Function Documentation

#### 4.41.2.1 InterruptCallout\_Add()

```
void InterruptCallout_Add (
    InterruptCalloutFunc pfCallout_ )
```

InterruptCallout\_Add.

Add a particular callout function to be executed whenever an interrupt is called (or returned-from).

#### Parameters

<i>pf<sub>↔</sub></i> <i>Callout<sub>↔</sub></i> —	Pointer to an interrupt callout function.
--	---

Definition at line 39 of file [interrupt\\_callout.c](#).

#### 4.41.2.2 InterruptCallout\_Run()

```
void InterruptCallout_Run (
    bool bEntry_,
    uint8_t u8Vector_ )
```

InterruptCallout\_Run.

Run all interrupt callouts currently installed.

#### Parameters

<i>bEntry_</i>	true - interrupt entry, false - interrupt exit
<i>u8<sub>↔</sub></i> <i>Vector_</i>	Interrupt vector # (undefined for interrupt-exit)

Definition at line 50 of file [interrupt\\_callout.c](#).

## 4.42 interrupt\_callout.c

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( ((/(      )\      | -- [ Funkenstein ] -----
00005 *      /( ) )/( )((( ( )\      | -- [ Little ] -----
00006 *      ( ) _ ( )      )\ _ )\ (( ( ( ) ( )      | -- [ AVR ] -----
00007 *      | _ | |      ( ) _ ( )\ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \      \ V / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _ _ / _ \      \ / / | _ \      |
00010 *      | _ | | _ _ / _ \      \ / / | _ \      | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, 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*)(
00042         malloc(sizeof(*pstNewCallout)));
00043     pstNewCallout->pstNext = pstCallouts;
00044     pstNewCallout->pfCallout = pfCallout_;
00045     pstCallouts = pstNewCallout;
00046 }
00047
00048 //-----
00049 void InterruptCallout_Run( bool bEntry_, uint8_t u8Vector_ )
00050 {
00051     Interrupt_Callout_t *pstCallout = pstCallouts;
00052     while (pstCallout)
00053     {
00054         pstCallout->pfCallout( bEntry_, u8Vector_ );
00055         pstCallout = pstCallout->pstNext;
00056     }
00057 }
00058

```

## 4.43 src/avr\_cpu/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.*

### 4.43.1 Detailed Description

Module providing functionality allowing emulator extensions to be triggered on interrupts.

Definition in file [interrupt\\_callout.h](#).

### 4.43.2 Function Documentation

#### 4.43.2.1 InterruptCallout\_Add()

```
void InterruptCallout_Add (
    InterruptCalloutFunc pfCallout_ )
```

[InterruptCallout\\_Add](#).

Add a particular callout function to be executed whenever an interrupt is called (or returned-from).

#### Parameters

<i>pf↔ Callout↔ _</i>	Pointer to an interrupt callout function.
-------------------------------	---

Definition at line 39 of file [interrupt\\_callout.c](#).

#### 4.43.2.2 InterruptCallout\_Run()

```
void InterruptCallout_Run (
    bool bEntry_,
    uint8_t u8Vector_ )
```

[InterruptCallout\\_Run](#).

Run all interrupt callouts currently installed.

## Parameters

<i>bEntry_</i>	true - interrupt entry, false - interrupt exit
<i>u8Vector_</i>	Interrupt vector # (undefined for interrupt-exit)

Definition at line 50 of file [interrupt\\_callout.c](#).

## 4.44 interrupt\_callout.h

```

00001 /*****
00002  *      (      (      (      |
00003  *      )\ ) )\ )      (      |
00004  *      ((/( ((/(      \      | -- [ Funkenstein ] -----
00005  *      /( ) )/( )((( ( )\      | -- [ Little ] -----
00006  *      ( )_ ( )      )\ _ )\ (( ( ( ) ( )      | -- [ AVR ] -----
00007  *      | | _ | |      ( )_ \ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008  *      | _ | | _      / _ \      \ V / | _ /      | -- [ Runtime ] -----
00009  *      | _ | | _      / _ \      \ V / | _ \      |
00010  *      | _ | | _      / _ \      \ V / | _ \      | "Yeah, it does Arduino..."
00011  * -----+-----
00012  * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013  * See license.txt for details
00014  *****/
00022 #ifndef __INTERRUPT_CALLOUT_H__
00023 #define __INTERRUPT_CALLOUT_H__
00024
00025 #include <stdint.h>
00026 #include <stdbool.h>
00027
00028 //-----
00030 typedef void (*InterruptCalloutFunc)( bool bEntry_, uint8_t u8Vector_ );
00031
00032 //-----
00041 void InterruptCallout_Add( InterruptCalloutFunc pfCallout_ );
00042
00043 //-----
00052 void InterruptCallout_Run( bool bEntry_, uint8_t u8Vector_ );
00053
00054
00055 #endif
00056

```

## 4.45 src/avr\_cpu/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.*

### 4.45.1 Detailed Description

Extended emulator functionality allowing for functions to be triggered based on RAM-write operations.

Definition in file [write\\_callout.h](#).

### 4.45.2 Function Documentation

#### 4.45.2.1 WriteCallout\_Add()

```
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

<i>pfCallout_</i>	- Pointer to the callout function
<i>u16Addr_</i>	- Address in RAM that triggers the callout when written

Definition at line 60 of file [write\\_callout.c](#).

#### 4.45.2.2 WriteCallout\_Run()

```
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

<i>u16Addr_</i>	- Address in RAM currently being modified
<i>u8Data_</i>	- 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](#).

## 4.46 write\_callout.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ )  )\ )      (      |
00004 *      ((/( (/(      \      ( ( (/(      | -- [ Funkenstein ] -----
00005 *      /( )  /( ) )((( ( )\ )\  /( )      | -- [ Little ] -----
00006 *      ( )_ ( )      )\ _ )\ ( ( ( )_ ( )      | -- [ AVR ] -----
00007 *      | _ | | |      ( )_ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | |      / _ \ \ \ / / | _ \      | -- [ Runtime ] -----
00009 *      | _ | | _ | / _ \ \ \ / / | _ \      |
00010 *      | _ | | _ | / _ \ \ \ / / | _ \      | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00022 #ifndef __WRITE_CALLOUT_H__
00023 #define __WRITE_CALLOUT_H__
00024
00025 #include <stdint.h>
00026 #include <stdbool.h>
00027
00028 //-----
00030 typedef bool (*WriteCalloutFunc)(uint16_t u16Addr_, uint8_t u8Data_);
00031
00032 //-----
00043 void WriteCallout_Add( WriteCalloutFunc pfCallout_, uint16_t u16Addr_ );
00044
00045 //-----
00058 bool WriteCallout_Run( uint16_t u16Addr_, uint8_t u8Data_ );
00059
00060
00061 #endif
00062

```

## 4.47 src/config/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.*

### 4.47.1 Detailed Description

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

Definition in file [emu\\_config.h](#).

## 4.47.2 Macro Definition Documentation

### 4.47.2.1 CONFIG\_TRACEBUFFER\_SIZE

```
#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](#).

### 4.47.2.2 FEATURE\_USE\_JUMPTABLES

```
#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](#).

## 4.48 emu\_config.h

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ )  )\ )      (      )\ )  |
00004 *      ((/( ((/(      \      ( ((/(      | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( ) \      / ( )      | -- [ Little ] -----
00006 *      ( ) _ ( )      ) \ _ \      ( ) ( ) ( )      | -- [ AVR ] -----
00007 *      | _ | | _      ( ) _ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ / / | _ \      | -- [ Runtime ] -----
00009 *      | _ | | _      / _ \ \ \ / / | _ \      |
00010 *      | _ | | _      / _ \ \ \ / / | _ \      | "Yeah, it does Arduino..."
00011 *      -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00022 #ifndef __EMU_CONFIG_H__
00023 #define __EMU_CONFIG_H__
00024
00025 #include <stdint.h>
00026 #include <stdbool.h>
00027
00028 #define CONFIG_IO_ADDRESS_BYTES      (256)          // First bytes of address space are I/O
00029 range
00044 #define FEATURE_USE_JUMPTABLES      (1)
00045
00053 #define CONFIG_TRACEBUFFER_SIZE      (1000)
00054
00055 #endif
00056
```

## 4.49 src/config/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\_UART**, **OPTION\_NUM** }  
*Enumerated type specifying the known command-line options accepted by fI AVR.*

### 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.*

#### 4.49.1 Detailed Description

Module for managing command-line options.

Definition in file [options.c](#).

## 4.49.2 Enumeration Type Documentation

### 4.49.2.1 OptionIndex\_t

enum [OptionIndex\\_t](#)

Enumerated type specifying the known command-line options accepted by fIAVR.

#### Enumerator

OPTION_NUM	Total count of command-line options supported.
------------	--

Definition at line [44](#) of file [options.c](#).

## 4.49.3 Function Documentation

### 4.49.3.1 Options\_GetByName()

```
const char* Options_GetByName (  
    const char * szAttribute_ )
```

[Options\\_GetByName](#).

Return the parameter value associated with an option attribute.

#### Parameters

<i>szAttribute_</i>	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 [99](#) of file [options.c](#).

### 4.49.3.2 Options\_Init()

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

[Options\\_Init](#).

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

**Parameters**

<i>argc</i> ↔ —	argc, passed in from main
<i>argv</i> ↔ —	argv, passed in from main

Definition at line 199 of file [options.c](#).

**4.49.3.3 Options\_Parse()**

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

Options\_Parse.

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

**Parameters**

<i>argc</i> ↔ —	Number of arguments
<i>argv</i> ↔ —	Argument vector, passed from main().

Definition at line 188 of file [options.c](#).

**4.49.3.4 Options\_ParseElement()**

```
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**

<i>start</i> ↔ —	Starting index
<i>argc</i> ↔ —	Total number of arguments
<i>argv</i> ↔ —	Command-line argument vector



## Returns

The next index to process

Definition at line 126 of file [options.c](#).

### 4.49.3.5 Options\_PrintUsage()

```
void Options_PrintUsage (
    void )
```

Options\_PrintUsage.

Print a brief description of each command-line option and its usage.

Definition at line 206 of file [options.c](#).

### 4.49.3.6 Options\_SetDefaults()

```
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 93 of file [options.c](#).

## 4.49.4 Variable Documentation

### 4.49.4.1 astAttributes

```
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 },
    {"--uart", "Run UART over the specified TCP port", NULL, false },
}
```

Table of available commandline options.

Order must match enumeration defined above.

Definition at line 68 of file [options.c](#).

## 4.50 options.c

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( (/(      )\      | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) (( ( )\ )\ / ( ) | -- [ Little ] -----
00006 *      ( ) _ ( ) )\ _ )\ ( ) ( ) ( ) | -- [ AVR ] -----
00007 *      | | _ | |      ( ) _ ( ) \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ v / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ | / \ \ \ \ \ / | _ \ |
00010 *      | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, 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     OPTION_UART,
00059 //-- New options go here ^^^
00060     OPTION_NUM
00061 } OptionIndex_t;
00062
00063 //-----
00068 static Option_t astAttributes[OPTION_NUM] =
00069 {
00070     {"--variant", "Specify the CPU variant by model name (default - atmega328p)", NULL, false },
00071     {"--freq", "Speed (in Hz) of the simulated CPU", NULL, false },
00072     {"--hexfile", "Programming file (intel HEX binary). Mutually exclusive with --elffile ", NULL, false },
00073 },
00074     {"--elffile", "Programming file (ELF binary). Mutually exclusive with --hexfile", NULL, false },
00075     {"--debug", "Run simulator in interactive debug mode. Mutually exclusive with --gdb", NULL, true },
00076 },
00077     {"--gdb", "Run simulator as a GDB remote, on the specified port.", NULL, false },
00078     {"--silent", "Start without the flavr-banner print", NULL, true },
00079     {"--disasm", "Disassemble programming file to standard output", NULL, true },
00080     {"--trace", "Enable tracebuffer support when used in conjunction with --debug", NULL, true },
00081     {"--mark3", "Enable Mark3 kernel-aware plugin", NULL, true },
00082     {"--exitreset", "Exit simulator if a jump-to-zero operation is encountered", NULL, true },
00083     {"--profile", "Run with code profile and code coverage enabled", NULL, true },
00084     {"--uart", "Run UART over the specified TCP port", NULL, false },
00085 };
00086
00087 //-----
00093 static void Options_SetDefaults( void )
00094 {
00095     astAttributes[ OPTION_VARIANT ].szParameter = strdup( "atmega328p" );
00096     astAttributes[ OPTION_FREQ ].szParameter = strdup( "16000000" );
00097 }
00098 //-----
00099 const char *Options_GetByName (const char *szAttribute_)
00100 {
00101     uint16_t j;
00102
00103     // linear search for the correct option value.
00104     for (j = 0; j < OPTION_NUM; j++)
00105     {
00106         if (0 == strcmp(astAttributes[j].szAttribute, szAttribute_))

```

```

00107         {
00108             return (const char*)astAttributes[j].szParameter;
00109         }
00110     }
00111     return NULL;
00112 }
00113
00114 //-----
00126 static uint16_t Options_ParseElement( int start_, int argc_, char **argv_ )
00127 {
00128     // Parse out specific option parameter data for a given option attribute
00129     uint16_t i = start_;
00130     uint16_t j;
00131
00132     while (i < argc_)
00133     {
00134         // linear search for the correct option value.
00135         for (j = 0; j < OPTION_NUM; j++)
00136         {
00137             if (0 == strcmp(astAttributes[j].szAttribute, argv_[i]))
00138             {
00139                 // Match - is the option stand-alone, or does it take a parameter?
00140                 if (astAttributes[j].bStandalone)
00141                 {
00142                     // Standalone argument, auto-seed a "1" value for the parameter to
00143                     // indicate that the option was set on the commandline
00144                     astAttributes[j].szParameter = strdup("1");
00145                     return 1;
00146                 }
00147
00148                 // ensure the user provided a parameter for this attribute
00149                 if (i + 1 >= argc_)
00150                 {
00151                     fprintf( stderr, "Error: Parameter expected for attribute %s", argv_[i] );
00152                     exit(-1);
00153                 }
00154                 else if (*(char*)argv_[i+1] == '-')
00155                 {
00156                     fprintf( stderr, "Error: Parameter expected for attribute %s", argv_[i] );
00157                     exit(-1);
00158                 }
00159                 // Check to see if a parameter has already been set; if so, free the existing value
00160                 if (NULL != astAttributes[j].szParameter)
00161                 {
00162                     free(astAttributes[j].szParameter );
00163                 }
00164                 // fprintf( stderr, "Match: argv[i]=%s, argv[i+1]=%s\n", argv_[i], argv_[i+1] );
00165                 astAttributes[j].szParameter = strdup(argv_[i+1]);
00166             }
00167         }
00168         // Read attribute + parameter combo, 2 tokens
00169         return 2;
00170     }
00171
00172     // Unknown option - 1 token
00173     fprintf( stderr, "WARN: Invalid option \"%s\"", argv_[i] );
00174
00175     return 1;
00176 }
00177
00178 //-----
00188 static void Options_Parse(int argc_, char **argv_ )
00189 {
00190     uint16_t i = 1;
00191     while (i < argc_)
00192     {
00193         // Parse out token from the command line array.
00194         i += Options_ParseElement( i, argc_, argv_ );
00195     }
00196 }
00197
00198 //-----
00199 void Options_Init( int argc_, char **argv_ )
00200 {
00201     Options_SetDefaults();
00202     Options_Parse( argc_, argv_ );
00203 }
00204
00205 //-----
00206 void Options_PrintUsage(void)
00207 {
00208     int i;
00209     printf("\n Usage:\n\n"
00210            "      flavr <options>\n\n Where <options> include:\n");
00211     for (i = 0; i < OPTION_NUM; i++)
00212     {
00213         printf( "      %14s: %s", astAttributes[i].szAttribute, astAttributes[i].szDescription );

```

```

00214         if (!lastAttributes[i].bStandalone)
00215         {
00216             printf(" (takes an argument)" );
00217         }
00218         printf( "\n" );
00219     }
00220 }

```

## 4.51 src/config/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 KB * (1024)`

### Functions

- const [AVR\\_Variant\\_t](#) \* [Variant\\_GetByName](#) (const char \*szName\_)  
*Variant\_GetByName.*

### Variables

- static const [AVR\\_Vector\\_Map\\_t](#) **stSmallAtMegaVectors**
- static const [AVR\\_Vector\\_Map\\_t](#) **stMediumAtMegaVectors**
- static const [AVR\\_Vector\\_Map\\_t](#) **stLargeAtMegaVectors**
- static [AVR\\_Feature\\_Map\\_t](#) **stSmallAtMegaFeatures**
- static [AVR\\_Feature\\_Map\\_t](#) **stMediumAtMegaFeatures**
- static [AVR\\_Feature\\_Map\\_t](#) **stLargeAtMegaFeatures**
- static [AVR\\_Variant\\_t](#) **astVariants** []

### 4.51.1 Detailed Description

Module containing a table of device variants supported by flavr.

Definition in file [variant.c](#).

### 4.51.2 Function Documentation

#### 4.51.2.1 Variant\_GetByName()

```

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

<code>szName</code>	String containing a variant name to check against (i.e. "atmega328p")
---------------------	---

## Returns

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

Definition at line 196 of file [variant.c](#).

## 4.51.3 Variable Documentation

4.51.3.1 `astVariants`

```
AVR_Variant_t astVariants[] [static]
```

## Initial value:

```
=
{
    { "atmega1284p", 16 KB, 128 KB, 4 KB, &stLargeAtMegaFeatures, &stLargeAtMegaVectors },
    { "atmega1284", 16 KB, 128 KB, 4 KB, &stLargeAtMegaFeatures, &stLargeAtMegaVectors },
    { "atmega644p", 4 KB, 64 KB, 2 KB, &stMediumAtMegaFeatures, &stMediumAtMegaVectors },
    { "atmega644", 4 KB, 64 KB, 2 KB, &stMediumAtMegaFeatures, &stMediumAtMegaVectors },
    { "atmega328p", 2 KB, 32 KB, 1 KB, &stSmallAtMegaFeatures, &stSmallAtMegaVectors },
    { "atmega328", 2 KB, 32 KB, 1 KB, &stSmallAtMegaFeatures, &stSmallAtMegaVectors },
    { "atmega168pa", 1 KB, 16 KB, 0.5 KB, &stSmallAtMegaFeatures, &stSmallAtMegaVectors },
    { "atmega168", 1 KB, 16 KB, 0.5 KB, &stSmallAtMegaFeatures, &stSmallAtMegaVectors },
    { "atmega88pa", 1 KB, 8 KB, 0.5 KB, &stSmallAtMegaFeatures, &stSmallAtMegaVectors },
    { "atmega88", 1 KB, 8 KB, 0.5 KB, &stSmallAtMegaFeatures, &stSmallAtMegaVectors },
    { "atmega48pa", 0.5 KB, 4 KB, 0.25 KB, &stSmallAtMegaFeatures, &stSmallAtMegaVectors },
    { "atmega48", 0.5 KB, 4 KB, 0.25 KB, &stSmallAtMegaFeatures, &stSmallAtMegaVectors },
    { 0 }
}
```

Definition at line 178 of file [variant.c](#).

4.51.3.2 `stLargeAtMegaFeatures`

```
AVR_Feature_Map_t stLargeAtMegaFeatures [static]
```

## Initial value:

```
= {
    .bHasTimer3 = true,
    .bHasUSART1 = true,
    .bHasInt2 = true,
    .bHasPCInt3 = true
}
```

Definition at line 170 of file [variant.c](#).

#### 4.51.3.3 stMediumAtMegaFeatures

```
AVR_Feature_Map_t stMediumAtMegaFeatures [static]
```

**Initial value:**

```
= {
    .bHasTimer3 = false,
    .bHasUSART1 = false,
    .bHasInt2 = true,
    .bHasPCInt3 = true
}
```

Definition at line 162 of file [variant.c](#).

#### 4.51.3.4 stSmallAtMegaFeatures

```
AVR_Feature_Map_t stSmallAtMegaFeatures [static]
```

**Initial value:**

```
= {
    .bHasTimer3 = false,
    .bHasUSART1 = false,
    .bHasInt2 = false,
    .bHasPCInt3 = false
}
```

Definition at line 154 of file [variant.c](#).

## 4.52 variant.c

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( ((/(      \      | -- [ Funkenstein ] -----
00005 *      /( ) )/( )((( ( )\ )\ /( )      | -- [ Little ] -----
00006 *      ( )_ ( )      )\ _ )\ ( ( ( )      | -- [ AVR ] -----
00007 *      | | _ | |      ( )_ \ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ V / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _ _ | / _ \ \ \ \ / | _ \      |
00010 *      |      |      | "Yeah, it does Arduino..."
00011 *      +-----+
00012 * (c) Copyright 2014-17, 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 KB * (1024)
00030
00031 //-----
00032 // This vector table works for :
00033 // atMega48pa, 88pa, 168p, 328p
00034 static const AVR_Vector_Map_t stSmallAtMegaVectors = {
00035     .RESET = 0x00,
00036     .INT0 = 0x01,
00037     .INT1 = 0x02,
```

```

00038     .PCINT0 = 0x03,
00039     .PCINT1 = 0x04,
00040     .PCINT2 = 0x05,
00041     .WDT = 0x06,
00042     .TIMER2_COMPA = 0x07,
00043     .TIMER2_COMPB = 0x08,
00044     .TIMER2_OVF = 0x09,
00045     .TIMER1_CAPT = 0x0A,
00046     .TIMER1_COMPA = 0x0B,
00047     .TIMER1_COMPB = 0x0C,
00048     .TIMER1_OVF = 0x0D,
00049     .TIMER0_COMPA = 0x0E,
00050     .TIMER0_COMPB = 0x0F,
00051     .TIMER0_OVF = 0x10,
00052     .SPI_STC = 0x11,
00053     .USART0_RX = 0x12,
00054     .USART0_UDRE = 0x13,
00055     .USART0_TX = 0x14,
00056     .ADC = 0x15,
00057     .EE_READY = 0x16,
00058     .ANALOG_COMP = 0x17,
00059     .TWI = 0x18,
00060     .SPM_READY = 0x19,
00061     .INT2 = VECTOR_NOT_SUPPORTED,
00062     .PCINT3 = VECTOR_NOT_SUPPORTED,
00063     .USART1_RX = VECTOR_NOT_SUPPORTED,
00064     .USART1_UDRE = VECTOR_NOT_SUPPORTED,
00065     .USART1_TX = VECTOR_NOT_SUPPORTED,
00066     .TIMER3_CAPT = VECTOR_NOT_SUPPORTED,
00067     .TIMER3_COMPA = VECTOR_NOT_SUPPORTED,
00068     .TIMER3_COMPB = VECTOR_NOT_SUPPORTED,
00069     .TIMER3_OVF= VECTOR_NOT_SUPPORTED
00070 };
00071
00072 //-----
00073 // This vector table works for :
00074 // atMega644p
00075 static const AVR_Vector_Map_t stMediumAtMegaVectors = {
00076     .RESET = 0x00,
00077     .INT0 = 0x01,
00078     .INT1 = 0x02,
00079     .INT2 = 0x03,
00080     .PCINT0 = 0x04,
00081     .PCINT1 = 0x05,
00082     .PCINT2 = 0x06,
00083     .PCINT3 = 0x07,
00084     .WDT = 0x08,
00085     .TIMER2_COMPA = 0x09,
00086     .TIMER2_COMPB = 0x0A,
00087     .TIMER2_OVF = 0x0B,
00088     .TIMER1_CAPT = 0x0C,
00089     .TIMER1_COMPA = 0x0D,
00090     .TIMER1_COMPB = 0x0E,
00091     .TIMER1_OVF = 0x0F,
00092     .TIMER0_COMPA = 0x10,
00093     .TIMER0_COMPB = 0x11,
00094     .TIMER0_OVF = 0x12,
00095     .SPI_STC = 0x13,
00096     .USART0_RX = 0x14,
00097     .USART0_UDRE = 0x15,
00098     .USART0_TX = 0x16,
00099     .ANALOG_COMP = 0x17,
00100     .ADC = 0x18,
00101     .EE_READY = 0x19,
00102     .TWI = 0x1A,
00103     .SPM_READY = 0x1B,
00104     .USART1_RX = VECTOR_NOT_SUPPORTED,
00105     .USART1_UDRE = VECTOR_NOT_SUPPORTED,
00106     .USART1_TX = VECTOR_NOT_SUPPORTED,
00107     .TIMER3_CAPT = VECTOR_NOT_SUPPORTED,
00108     .TIMER3_COMPA = VECTOR_NOT_SUPPORTED,
00109     .TIMER3_COMPB = VECTOR_NOT_SUPPORTED,
00110     .TIMER3_OVF = VECTOR_NOT_SUPPORTED
00111 };
00112
00113 //-----
00114 // Map for atMega1284p
00115 static const AVR_Vector_Map_t stLargeAtMegaVectors = {
00116     .RESET = 0x00,
00117     .INT0 = 0x01,
00118     .INT1 = 0x02,
00119     .INT2 = 0x03,
00120     .PCINT0 = 0x04,
00121     .PCINT1 = 0x05,
00122     .PCINT2 = 0x06,
00123     .PCINT3 = 0x07,
00124     .WDT = 0x08,

```

```

00125     .TIMER2_COMPA = 0x09,
00126     .TIMER2_COMPB = 0x0A,
00127     .TIMER2_OVF = 0x0B,
00128     .TIMER1_CAPT = 0x0C,
00129     .TIMER1_COMPA = 0x0D,
00130     .TIMER1_COMPB = 0x0E,
00131     .TIMER1_OVF = 0x0F,
00132     .TIMER0_COMPA = 0x10,
00133     .TIMER0_COMPB = 0x11,
00134     .TIMER0_OVF = 0x12,
00135     .SPI_STC = 0x13,
00136     .USART0_RX = 0x14,
00137     .USART0_UDRE = 0x15,
00138     .USART0_TX = 0x16,
00139     .ANALOG_COMP = 0x17,
00140     .ADC = 0x18,
00141     .EE_READY = 0x19,
00142     .TWI = 0x1A,
00143     .SPM_READY = 0x1B,
00144     .USART1_RX = 0x1C,
00145     .USART1_UDRE = 0x1D,
00146     .USART1_TX = 0x1E,
00147     .TIMER3_CAPT = 0x1F,
00148     .TIMER3_COMPA = 0x20,
00149     .TIMER3_COMPB = 0x21,
00150     .TIMER3_OVF = 0x22
00151 };
00152
00153 //-----
00154 static AVR_Feature_Map_t stSmallAtMegaFeatures = {
00155     .bHasTimer3 = false,
00156     .bHasUSART1 = false,
00157     .bHasInt2 = false,
00158     .bHasPCInt3 = false
00159 };
00160
00161 //-----
00162 static AVR_Feature_Map_t stMediumAtMegaFeatures = {
00163     .bHasTimer3 = false,
00164     .bHasUSART1 = false,
00165     .bHasInt2 = true,
00166     .bHasPCInt3 = true
00167 };
00168
00169 //-----
00170 static AVR_Feature_Map_t stLargeAtMegaFeatures = {
00171     .bHasTimer3 = true,
00172     .bHasUSART1 = true,
00173     .bHasInt2 = true,
00174     .bHasPCInt3 = true
00175 };
00176
00177 //-----
00178 static AVR_Variant_t astVariants[] =
00179 {
00180     { "atmega1284p", 16 KB, 128 KB, 4 KB, &stLargeAtMegaFeatures, &stLargeAtMegaVectors },
00181     { "atmega1284", 16 KB, 128 KB, 4 KB, &stLargeAtMegaFeatures, &stLargeAtMegaVectors },
00182     { "atmega644p", 4 KB, 64 KB, 2 KB, &stMediumAtMegaFeatures, &stMediumAtMegaVectors },
00183     { "atmega644", 4 KB, 64 KB, 2 KB, &stMediumAtMegaFeatures, &stMediumAtMegaVectors },
00184     { "atmega328p", 2 KB, 32 KB, 1 KB, &stSmallAtMegaFeatures, &stSmallAtMegaVectors },
00185     { "atmega328", 2 KB, 32 KB, 1 KB, &stSmallAtMegaFeatures, &stSmallAtMegaVectors },
00186     { "atmega168pa", 1 KB, 16 KB, 0.5 KB, &stSmallAtMegaFeatures, &stSmallAtMegaVectors },
00187     { "atmega168", 1 KB, 16 KB, 0.5 KB, &stSmallAtMegaFeatures, &stSmallAtMegaVectors },
00188     { "atmega88pa", 1 KB, 8 KB, 0.5 KB, &stSmallAtMegaFeatures, &stSmallAtMegaVectors },
00189     { "atmega88", 1 KB, 8 KB, 0.5 KB, &stSmallAtMegaFeatures, &stSmallAtMegaVectors },
00190     { "atmega48pa", 0.5 KB, 4 KB, 0.25 KB, &stSmallAtMegaFeatures, &stSmallAtMegaVectors },
00191     { "atmega48", 0.5 KB, 4 KB, 0.25 KB, &stSmallAtMegaFeatures, &stSmallAtMegaVectors },
00192     { 0 }
00193 };
00194
00195 //-----
00196 const AVR_Variant_t *Variant_GetByName( const char *szName_ )
00197 {
00198     AVR_Variant_t *pstVariant = astVariants;
00199     while (pstVariant->szName)
00200     {
00201         if (0 == strcmp(pstVariant->szName, szName_ ) )
00202         {
00203             return pstVariant;
00204         }
00205         pstVariant++;
00206     }
00207     return NULL;
00208 }
00209

```



## 4.53 src/config/variant.h File Reference

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

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

### Data Structures

- struct [AVR\\_Vector\\_Map\\_t](#)
- struct [AVR\\_Feature\\_Map\\_t](#)
- struct [AVR\\_Variant\\_t](#)

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

### Macros

- #define **VECTOR\_NOT\_SUPPORTED** (0xFF)

### Functions

- const [AVR\\_Variant\\_t](#) \* [Variant\\_GetByName](#) (const char \*szName\_)  
*Variant\_GetByName.*

#### 4.53.1 Detailed Description

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

Definition in file [variant.h](#).

#### 4.53.2 Function Documentation

##### 4.53.2.1 Variant\_GetByName()

```
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

<code>sz↵ Name_</code>	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 196 of file [variant.c](#).

## 4.54 variant.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ )  )\ )  (      )\ )  |
00004 *      ((/( ((/(      \      ( ((/(      | -- [ Funkenstein ] -----
00005 *      /( )  /( )  ((( ( ( )\  )\  /( )  | -- [ Little ] -----
00006 *      ( )_ ( )_  )\ _ )\ ( ( ( )_ ( )_  | -- [ AVR ] -----
00007 *      | | _ | | _  ( )_ \ ( )_ \ / / | _ \  | -- [ Virtual ] -----
00008 *      | _ | | _ _  / _ \ \ V / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _ _ | / _ \ \ \ / | _ \      |
00010 *                                          | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __VARIANT_H__
00022 #define __VARIANT_H__
00023
00024 #include <stdbool.h>
00025 #include <stdint.h>
00026
00027 //-----
00028 #define VECTOR_NOT_SUPPORTED (0xFF) // Signal that a specific vector is not implemented on target device
00029
00030 //-----
00031 typedef struct {
00032     uint8_t RESET;
00033     uint8_t INT0;
00034     uint8_t INT1;
00035     uint8_t INT2;
00036     uint8_t PCINT0;
00037     uint8_t PCINT1;
00038     uint8_t PCINT2;
00039     uint8_t PCINT3;
00040     uint8_t WDT;
00041     uint8_t TIMER2_COMPA;
00042     uint8_t TIMER2_COMPB;
00043     uint8_t TIMER2_OVF;
00044     uint8_t TIMER1_CAPT;
00045     uint8_t TIMER1_COMPA;
00046     uint8_t TIMER1_COMPB;
00047     uint8_t TIMER1_OVF;
00048     uint8_t TIMER0_COMPA;
00049     uint8_t TIMER0_COMPB;
00050     uint8_t TIMER0_OVF;
00051     uint8_t SPI_STC;
00052     uint8_t USART0_RX;
00053     uint8_t USART0_UDRE;
00054     uint8_t USART0_TX;
00055     uint8_t ANALOG_COMP;
00056     uint8_t ADC;
00057     uint8_t EE_READY;
00058     uint8_t TWI;
00059     uint8_t SPM_READY;
00060     uint8_t USART1_RX;
00061     uint8_t USART1_UDRE;
00062     uint8_t USART1_TX;
00063     uint8_t TIMER3_CAPT;
00064     uint8_t TIMER3_COMPA;
00065     uint8_t TIMER3_COMPB;
00066     uint8_t TIMER3_OVF;
00067 } AVR_Vector_Map_t;

```

```

00068
00069 //-----
00070 typedef struct {
00071     bool bHasTimer3;
00072     bool bHasUSART1;
00073     bool bHasInt2;
00074     bool bHasPCInt3;
00075 } AVR_Feature_Map_t;
00076
00077 //-----
00082 typedef struct
00083 {
00084     const char *szName;
00085
00086     uint32_t    u32RAMSize;
00087     uint32_t    u32ROMSize;
00088     uint32_t    u32EESize;
00089
00090     const AVR_Feature_Map_t* pstFeatures;
00091     const AVR_Vector_Map_t* pstVectors;
00092
00093 } AVR_Variant_t;
00094
00095 //-----
00107 const AVR_Variant_t *Variant_GetByName( const char *szName_ );
00108
00109 #endif

```

## 4.55 src/debug/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](#) (uint32\_t u32Addr\_)  
*BreakPoint\_Insert.*
- void [BreakPoint\\_Delete](#) (uint32\_t u32Addr\_)  
*BreakPoint\_Delete.*
- bool [BreakPoint\\_EnabledAtAddress](#) (uint32\_t u32Addr\_)  
*BreakPoint\_EnabledAtAddress.*

#### 4.55.1 Detailed Description

Implements instruction breakpoints for debugging based on code path.

Definition in file [breakpoint.c](#).

#### 4.55.2 Function Documentation

#### 4.55.2.1 BreakPoint\_Delete()

```
void BreakPoint_Delete (
    uint32_t u32Addr_ )
```

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**

<i>u32↔</i> <i>Addr_</i>	Address of the breakpoint to delete.
-----------------------------	--------------------------------------

Definition at line 55 of file [breakpoint.c](#).

**4.55.2.2 BreakPoint\_EnabledAtAddress()**

```
bool BreakPoint_EnabledAtAddress (
    uint32_t u32Addr_ )
```

BreakPoint\_EnabledAtAddress.

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

**Parameters**

<i>u32↔</i> <i>Addr_</i>	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](#).

**4.55.2.3 BreakPoint\_Insert()**

```
void BreakPoint_Insert (
    uint32_t u32Addr_ )
```

BreakPoint\_Insert.

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

**Parameters**

<i>u32↔</i> <i>Addr_</i>	Address of the breakpoint.
-----------------------------	----------------------------

Definition at line 29 of file [breakpoint.c](#).

## 4.56 breakpoint.c

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( (/(      )\      | -- [ Funkenstein ] -----
00005 *      /( ) )/( )((( ( )\      | -- [ Little ] -----
00006 *      ( )_ ( )      )\ _ )\ ( ( ( )      | -- [ AVR ] -----
00007 *      | | _ | |      ( )\ ( )\ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ / / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _      / _ \ \ \ / / | _ \      |
00010 *      | _ | | _      / _ \ \ \ / / | _ \      | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, 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( uint32_t u32Addr_ )
00030 {
00031     // Don't add multiple breakpoints at the same address
00032     if (BreakPoint_EnabledAtAddress( u32Addr_ ))
00033     {
00034         return;
00035     }
00036
00037     BreakPoint_t *pstNewBreak = NULL;
00038
00039     pstNewBreak = (BreakPoint_t*)malloc( sizeof(BreakPoint_t) );
00040
00041     pstNewBreak->next = stCPU.pstBreakPoints;
00042     pstNewBreak->prev = NULL;
00043
00044     pstNewBreak->u32Addr = u32Addr_;
00045
00046     if (stCPU.pstBreakPoints)
00047     {
00048         BreakPoint_t *pstTemp = stCPU.pstBreakPoints;
00049         pstTemp->prev = pstNewBreak;
00050     }
00051     stCPU.pstBreakPoints = pstNewBreak;
00052 }
00053
00054 //-----
00055 void BreakPoint_Delete( uint32_t u32Addr_ )
00056 {
00057     BreakPoint_t *pstTemp = stCPU.pstBreakPoints;
00058
00059     while (pstTemp)
00060     {
00061         if (pstTemp->u32Addr == u32Addr_)
00062         {
00063             // Remove node -- reconnect surrounding elements
00064             BreakPoint_t *pstNext = pstTemp->next;
00065             if (pstNext)
00066             {
00067                 pstNext->prev = pstTemp->prev;
00068             }
00069
00070             BreakPoint_t *pstPrev = pstTemp->prev;
00071             if (pstPrev)
00072             {
00073                 pstPrev->next = pstTemp->next;
00074             }
00075
00076             // Adjust list-head if necessary
00077             if (pstTemp == stCPU.pstBreakPoints)
00078             {
00079                 stCPU.pstBreakPoints = pstNext;
00080             }
00081
00082             // Free the node/iterate to next node.
00083             pstPrev = pstTemp;
00084             pstTemp = pstTemp->next;
00085             free(pstPrev);
00086         }
00087         else
00088         {
00089             pstTemp = pstTemp->next;
00090         }
00091     }

```

```

00091     }
00092 }
00093
00094 //-----
00095 bool BreakPoint_EnabledAtAddress( uint32_t u32Addr_ )
00096 {
00097     BreakPoint_t *pstTemp = stCPU.pstBreakPoints;
00098
00099     while (pstTemp)
00100     {
00101         if (pstTemp->u32Addr == u32Addr_)
00102         {
00103             return true;
00104         }
00105         pstTemp = pstTemp->next;
00106     }
00107     return false;
00108 }

```

## 4.57 src/debug/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](#) (uint32\_t u32Addr\_)  
*BreakPoint\_Insert.*
- void [BreakPoint\\_Delete](#) (uint32\_t u32Addr\_)  
*BreakPoint\_Delete.*
- bool [BreakPoint\\_EnabledAtAddress](#) (uint32\_t u32Addr\_)  
*BreakPoint\_EnabledAtAddress.*

#### 4.57.1 Detailed Description

Implements instruction breakpoints for debugging based on code path.

Definition in file [breakpoint.h](#).

## 4.57.2 Function Documentation

### 4.57.2.1 BreakPoint\_Delete()

```
void BreakPoint_Delete (
    uint32_t u32Addr_ )
```

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

<i>u32↔ Addr_</i>	Address of the breakpoint to delete.
-----------------------	--------------------------------------

Definition at line 55 of file [breakpoint.c](#).

### 4.57.2.2 BreakPoint\_EnabledAtAddress()

```
bool BreakPoint_EnabledAtAddress (
    uint32_t u32Addr_ )
```

BreakPoint\_EnabledAtAddress.

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

#### Parameters

<i>u32↔ Addr_</i>	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](#).

### 4.57.2.3 BreakPoint\_Insert()

```
void BreakPoint_Insert (
    uint32_t u32Addr_ )
```

BreakPoint\_Insert.

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



## Parameters

<code>u32Addr_</code>	Address of the breakpoint.
-----------------------	----------------------------

Definition at line 29 of file [breakpoint.c](#).

## 4.58 breakpoint.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ )  )\ )  (      )\ ) |
00004 *      (( / ( (( / (      \      ( (( / ( -- [ Funkenstein ] -----
00005 *      / ( ) / ( ) ((( ( ( ) \      / ( ) | -- [ Little ] -----
00006 *      ( ) _ ( )      )\ _ \      ( ( ( ( ) _ | -- [ AVR ] -----
00007 *      | _ | |      ( ) _ \ ( ) \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ V / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ _ | / _ \ \ \ \ / | _ \ |
00010 *                                     | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __BREAKPOINT_H__
00022 #define __BREAKPOINT_H__
00023
00024 #include <stdint.h>
00025 #include <stdbool.h>
00026
00027 #include "avr_cpu.h"
00028
00029 //-----
00033 typedef struct _BreakPoint
00034 {
00035     struct _BreakPoint *next;
00036     struct _BreakPoint *prev;
00037
00038     uint32_t u32Addr;
00039 } BreakPoint_t;
00040
00041 //-----
00050 void BreakPoint_Insert( uint32_t u32Addr_ );
00051
00052 //-----
00061 void BreakPoint_Delete( uint32_t u32Addr_ );
00062
00063 //-----
00073 bool BreakPoint_EnabledAtAddress( uint32_t u32Addr_ );
00074
00075 #endif
00076

```

## 4.59 src/debug/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

### 4.59.1 Detailed Description

Code profiling (exeuction and coverage) functionality.

Definition in file [code\\_profile.c](#).

### 4.59.2 Function Documentation

#### 4.59.2.1 Profile\_Hit()

```
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

<code>u32↵ Addr_</code>	- Address in ROM/FLASH being hit.
-----------------------------	-----------------------------------

Definition at line 127 of file [code\\_profile.c](#).

## 4.59.2.2 Profile\_Init()

```
void Profile_Init (
    uint32_t u32ROMSize_ )
```

Profile\_Init.

Initialize the code profiling module

## Parameters

<code>u32ROM↵ Size_</code>	- Size of the CPU's ROM/FLASH
--------------------------------	-------------------------------

Definition at line 280 of file [code\\_profile.c](#).

## 4.59.2.3 Profile\_Print()

```
void Profile_Print (
    void )
```

Profile\_Print.

Display the cumulative profiling stats

Definition at line 214 of file [code\\_profile.c](#).

## 4.60 code\_profile.c

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/ ( ((/ (      \      (      ((/ (      | -- [ Funkenstein ] -----
00005 *      / ( ) / ( ) ((( ( ) \      ) \      / ( )      | -- [ Little ] -----
00006 *      ( ) _ ( )      ) \ _ \      ( ) ( ( ) ( )      | -- [ AVR ] -----
00007 *      | _ | |      ( ) _ \ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \      \ V / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _ _      / _ \ \      \ _ / | _ \      |
00010 *      | _ | | _ _ _      / _ \ \      \ _ / | _ \      | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, 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;
00053
00054 //-----
00055 typedef struct
00056 {
00057     uint32_t u32CodeAddress;
00058     uint64_t u64Hits;
00059     char szDisasmLine[256];
00060 } AddressCoverageTLV_t;
00061
00062 //-----
00063 static Profile_t *pstProfile = 0;
00064 static uint32_t u32ROMSize = 0;
00065
00066 //-----
00067 static TLV_t *pstFunctionCoverageTLV = NULL;
00068 static TLV_t *pstFunctionProfileTLV = NULL;
00069 static TLV_t *pstAddressCoverageTLV = NULL;
00070
00071 //-----
00072 static void Profile_TLVInit(void)
00073 {
00074     pstFunctionProfileTLV = TLV_Alloc( sizeof(FunctionProfileTLV_t));
00075     pstFunctionProfileTLV->eTag = TAG_CODE_PROFILE_FUNCTION_GLOBAL;
00076
00077     pstFunctionCoverageTLV = TLV_Alloc( sizeof(FunctionCoverageTLV_t));
00078     pstFunctionCoverageTLV->eTag = TAG_CODE_COVERAGE_FUNCTION_GLOBAL;
00079
00080     pstAddressCoverageTLV = TLV_Alloc( sizeof(AddressCoverageTLV_t));
00081     pstAddressCoverageTLV->eTag = TAG_CODE_COVERAGE_ADDRESS;
00082 }
00083
00084 //-----
00085 static void Profile_FunctionCoverage( const char *szFunc_, uint32_t u32FuncSize_, uint32_t u32HitSize_ )
00086 {
00087     FunctionCoverageTLV_t *pstData = (FunctionCoverageTLV_t*)(&
    pstFunctionCoverageTLV->au8Data[0]);
00088
00089     strcpy(pstData->szSymName, szFunc_);
00090     pstData->u32FunctionSize = u32FuncSize_;
00091     pstData->u32AddressesHit = u32HitSize_;
00092     pstFunctionCoverageTLV->u16Len = strlen(szFunc_) + 8; // Size of the static + variable data
00093
00094     TLV_Write( pstFunctionCoverageTLV );
00095 }
00096
00097 //-----
00098 static void Profile_Function( const char *szFunc_, uint64_t u64Cycles_, uint64_t u64CPUCycles_ )
00099 {
00100     FunctionProfileTLV_t *pstData = (FunctionProfileTLV_t*)(&
    pstFunctionProfileTLV->au8Data[0]);
00101
00102     strcpy(pstData->szSymName, szFunc_);
00103     pstData->u64CyclesTotal = u64Cycles_;
00104     pstData->u64CPUCycles = u64CPUCycles_;
00105 }

```

```

00106     pstFunctionProfileTLV->u16Len = strlen(szFunc_) + 16; // Size of the static + variable data
00107
00108     TLV_Write( pstFunctionProfileTLV );
00109 }
00110
00111 //-----
00112 static void Profile_AddressCoverage( const char *szDisasm_, uint32_t u32Addr_, uint64_t u64Hits_ )
00113 {
00114     AddressCoverageTLV_t *pstData = (AddressCoverageTLV_t*) (&
pstAddressCoverageTLV->au8Data[0]);
00115
00116     strcpy(pstData->szDisasmLine, szDisasm_);
00117
00118     pstData->u32CodeAddress = u32Addr_;
00119     pstData->u64Hits = u64Hits_;
00120
00121     pstAddressCoverageTLV->u16Len = strlen(szDisasm_) + 12;
00122
00123     TLV_Write( pstAddressCoverageTLV );
00124 }
00125
00126 //-----
00127 void Profile_Hit( uint32_t u32Addr_ )
00128 {
00129     pstProfile[ u32Addr_ ].u64EpochHit++;
00130     pstProfile[ u32Addr_ ].u64TotalHit++;
00131
00132     Debug_Symbol_t *pstSym = pstProfile[ u32Addr_ ].pstSym;
00133     if (pstSym)
00134     {
00135         pstSym->u64EpochRefs++;
00136         pstSym->u64TotalRefs++;
00137     }
00138 }
00139
00140 //-----
00141 void Profile_ResetEpoch(void)
00142 {
00143     // Reset the epoch counters for all addresses
00144     int i;
00145     for (i = 0; i < u32ROMSize; i++)
00146     {
00147         pstProfile[i].u64EpochHit = 0;
00148     }
00149
00150     // Reset the per-symbol epoch counters
00151     Debug_Symbol_t *pstSym;
00152     int iSymCount = Symbol_Get_Func_Count();
00153     for (i = 0; i < iSymCount; i++)
00154     {
00155         pstSym = Symbol_Func_At_Index(i);
00156         pstSym->u64EpochRefs = 0;
00157     }
00158 }
00159
00160 //-----
00161 void Profile_PrintCoverageDisassembly(void)
00162 {
00163     Debug_Symbol_t *pstSym;
00164     int iSymCount = Symbol_Get_Func_Count();
00165     int i;
00166     int j;
00167
00168     printf( "=====\n");
00169     printf( "Detailed Code Coverage\n");
00170     printf( "=====\n");
00171     // Go through all of our symbols and show which instructions have actually
00172     // been hit.
00173     for (i = 0; i < iSymCount; i++)
00174     {
00175         pstSym = Symbol_Func_At_Index(i);
00176
00177         if (!pstSym)
00178         {
00179             break;
00180         }
00181
00182         printf("%s:\n", pstSym->szName);
00183         j = pstSym->u32StartAddr;
00184         while (j <= (int)pstSym->u32EndAddr)
00185         {
00186             uint16_t OP = stCPU.pu16ROM[j];
00187             stCPU.u32PC = (uint16_t)j;
00188
00189             if (pstProfile[j].u64TotalHit)
00190             {
00191                 printf( "[X]" );

```

```

00192         }
00193         else
00194         {
00195             printf( "[ ]" );
00196         }
00197         printf( " 0x%04X: [0x%04X] ", stCPU.u32PC, OP);
00198
00199         AVR_Decode(OP);
00200
00201         char szBuf[256];
00202         AVR_Disasm_Function(OP)(szBuf);
00203         printf( "%s", szBuf );
00204
00205         Profile_AddressCoverage( szBuf, stCPU.u32PC, pstProfile[j].
u64TotalHit );
00206
00207         j += AVR_Opcode_Size(OP);
00208     }
00209     printf("\n");
00210 }
00211 }
00212
00213 //-----
00214 void Profile_Print(void)
00215 {
00216     uint64_t u64TotalCycles = 0;
00217
00218     Debug_Symbol_t *pstSym;
00219     int iSymCount = Symbol_Get_Func_Count();
00220     int i;
00221     for (i = 0; i < iSymCount; i++)
00222     {
00223         pstSym = Symbol_Func_At_Index(i);
00224         u64TotalCycles += pstSym->u64TotalRefs;
00225     }
00226     printf("\n\nTotal cycles spent in known functions: %llu\n\n", u64TotalCycles);
00227
00228     printf( "=====\n");
00229     printf( "%60s: CPU utilization(%%)\n", "Function");
00230     printf( "=====\n");
00231     for (i = 0; i < iSymCount; i++)
00232     {
00233         pstSym = Symbol_Func_At_Index(i);
00234         printf( "%60s: %0.3f\n",
00235             pstSym->szName,
00236             100.0 * (double) (pstSym->u64TotalRefs) / (double) (u64TotalCycles) );
00237         Profile_Function( pstSym->szName, pstSym->u64TotalRefs, u64TotalCycles );
00238     }
00239
00240     printf( "=====\n");
00241     printf( "Code coverage summary:\n");
00242     printf( "=====\n");
00243     int iGlobalHits = 0;
00244     int iGlobalMisses = 0;
00245     for (i = 0; i < iSymCount; i++)
00246     {
00247         pstSym = Symbol_Func_At_Index(i);
00248         int j;
00249         int iHits = 0;
00250         int iMisses = 0;
00251
00252         for (j = pstSym->u32StartAddr; j < pstSym->u32EndAddr; j++)
00253         {
00254             if (pstProfile[j].u64TotalHit)
00255             {
00256                 iHits++;
00257                 iGlobalHits++;
00258             }
00259             else
00260             {
00261                 iMisses++;
00262                 iGlobalMisses++;
00263             }
00264
00265             // If this is a 2-opcode instruction, skip the next word, as to not skew the results
00266             uint16_t OP = stCPU.pu16ROM[j];
00267             if (2 == AVR_Opcode_Size(OP))
00268             {
00269                 j++;
00270             }
00271         }
00272         printf("%60s: %0.3f\n", pstSym->szName, 100.0 * (double)iHits/(double)(iHits + iMisses));
00273         Profile_FunctionCoverage(pstSym->szName, iHits + iMisses, iHits);
00274     }
00275     printf( "\n[Global Code Coverage] : %0.3f\n",
00276         100.0 * (double)iGlobalHits/(double)(iGlobalHits + iGlobalMisses));
00277

```

```

00278 }
00279 //-----
00280 void Profile_Init( uint32_t u32ROMSize_ )
00281 {
00282     // Allocate a lookup table, one entry per address in ROM to allow us to
00283     // gather code-coverage and code-profiling information.
00284     uint32_t u32BufSize = sizeof(Profile_t) * u32ROMSize_ ;
00285     u32ROMSize = u32ROMSize_;
00286     pstProfile = (Profile_t*)malloc( u32BufSize );
00287     memset( pstProfile, 0, u32BufSize );
00288
00289     // Go through the list of symbols, and associate each function with its
00290     // address range in the lookup table.
00291     int iFuncs = Symbol_Get_Func_Count();
00292     int i;
00293     for (i = 0; i < iFuncs; i++)
00294     {
00295         Debug_Symbol_t *pstSym = Symbol_Func_At_Index( i );
00296         int j;
00297         if (pstSym)
00298         {
00299             for (j = pstSym->u32StartAddr; j < pstSym->u32EndAddr; j++)
00300             {
00301                 pstProfile[j].pstSym = pstSym;
00302             }
00303         }
00304     }
00305
00306     Profile_TLVInit();
00307
00308     atexit( Profile_Print );
00309     atexit( Profile_PrintCoverageDissassembly );
00310 }

```

## 4.61 src/debug/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.*

### 4.61.1 Detailed Description

Code profiling (exeuction and coverage) functionality.

Definition in file [code\\_profile.h](#).

### 4.61.2 Function Documentation

#### 4.61.2.1 Profile\_Hit()

```
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

<i>u32Addr_</i>	- Address in ROM/FLASH being hit.
-----------------	-----------------------------------

Definition at line 127 of file [code\\_profile.c](#).

#### 4.61.2.2 Profile\_Init()

```
void Profile_Init (
    uint32_t u32ROMSize_ )
```

Profile\_Init.

Initialize the code profiling module

##### Parameters

<i>u32ROMSize_</i>	- Size of the CPU's ROM/FLASH
--------------------	-------------------------------

Definition at line 280 of file [code\\_profile.c](#).

#### 4.61.2.3 Profile\_Print()

```
void Profile_Print (
    void )
```

Profile\_Print.

Display the cumulative profiling stats

Definition at line 214 of file [code\\_profile.c](#).



## 4.62 code\_profile.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( ((/(      (      | -- [ Funkenstein ] -----
00005 *      /( ) )/( )((( ( )\      | -- [ Little ] -----
00006 *      ( )_ ( )      )\ _ )\ ( ( ( ) ( )      | -- [ AVR ] -----
00007 *      | _ | |      ( )_ \ ( )\ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \      \ v / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _ _ | / _ \ \      \ _ / | _ \      |
00010 *                                          | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __CODE_PROFILE_H__
00022 #define __CODE_PROFILE_H__
00023
00024 #include <stdint.h>
00025
00026 //-----
00034 void Profile_Init( uint32_t u32ROMSize_ );
00035
00036 //-----
00045 void Profile_Hit( uint32_t u32Addr_ );
00046
00047 //-----
00054 void Profile_Print(void);
00055
00056
00057 #endif
00058

```

## 4.63 src/debug/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

### 4.63.1 Detailed Description

Symbolic debugging support for data and functions.

Definition in file [debug\\_sym.c](#).

### 4.63.2 Function Documentation

#### 4.63.2.1 Symbol\_Add\_Func()

```
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

<i>sz</i> ↔ <i>Name_</i>	- Name of the symbol (string)
<i>u32</i> ↔ <i>Addr_</i>	- Start address of the function
<i>u32Len</i> ↔ <i>_</i>	- Size of the function (in bytes)

Definition at line 36 of file [debug\\_sym.c](#).

#### 4.63.2.2 Symbol\_Add\_Obj()

```
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

<i>sz↔ Name_</i>	- Name of the symbol (string)
<i>u32↔ Addr_</i>	- Start address of the object
<i>u32Len↔ _</i>	- Size of the object (in bytes)

Definition at line 51 of file [debug\\_sym.c](#).

## 4.63.2.3 Symbol\_Find\_Func\_By\_Name()

```
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

<i>sz↔ Name_</i>	- 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](#).

## 4.63.2.4 Symbol\_Find\_Obj\_By\_Name()

```
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

<i>sz↔ Name_</i>	- 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](#).

**4.63.2.5 Symbol\_Func\_At\_Index()**

```
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**

<i>u32↔ Index_</i>	- 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](#).

**4.63.2.6 Symbol\_Get\_Func\_Count()**

```
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](#).

#### 4.63.2.7 Symbol\_Get\_Obj\_Count()

```
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`.

#### 4.63.2.8 Symbol\_Obj\_At\_Index()

```
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

$u32 \leftarrow$ <u>Index</u>	- 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`.

#### 4.64 debug\_sym.c

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      )\ ) |
00004 *      ( )/( ( )/(      \      (      ( )/(      | -- [ Funkenstein ] -----
00005 *      /( )/( / )\ ((( ( )\ )\      / ( )\      | -- [ Little ] -----
00006 *      ( )\ ( )\      )\ )\ )\ ( )\ ( )\      | -- [ AVR ] -----
00007 *      | _ | | |      ( )\ ( )\ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _ |      \ / \ \ \ / \ / | _ | \      | -- [ Runtime ] -----
00009 *      | _ | | _ |      / \ \ \ \ / \ / | _ | \      |
00010 *                                                    | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00022 #include "debug_sym.h"
00023 #include <stdint.h>

```

```

00024 #include <stdio.h>
00025 #include <stdlib.h>
00026 #include <string.h>
00027
00028 //-----
00029 static Debug_Symbol_t *pstFuncSymbols = 0;
00030 static uint32_t u32FuncCount = 0;
00031
00032 static Debug_Symbol_t *pstObjSymbols = 0;
00033 static uint32_t u32ObjCount = 0;
00034
00035 //-----
00036 void Symbol_Add_Func( const char *szName_, const uint32_t u32Addr_, const uint32_t u32Len_ )
00037 {
00038     pstFuncSymbols = (Debug_Symbol_t*)realloc( pstFuncSymbols, (u32FuncCount + 1) * sizeof(
Debug_Symbol_t));
00039     Debug_Symbol_t *pstNew = &pstFuncSymbols[u32FuncCount];
00040
00041     pstNew->eType = DBG_FUNC;
00042     pstNew->szName = strdup( szName_ );
00043     pstNew->u32StartAddr = u32Addr_;
00044     pstNew->u32EndAddr = u32Addr_ + u32Len_ - 1;
00045     pstNew->u64EpochRefs = 0;
00046     pstNew->u64TotalRefs = 0;
00047     u32FuncCount++;
00048 }
00049
00050 //-----
00051 void Symbol_Add_Obj( const char *szName_, const uint32_t u32Addr_, const uint32_t u32Len_ )
00052 {
00053     pstObjSymbols = (Debug_Symbol_t*)realloc( pstObjSymbols, (u32ObjCount + 1) * sizeof(
Debug_Symbol_t));
00054     Debug_Symbol_t *pstNew = &pstObjSymbols[u32ObjCount];
00055
00056     pstNew->eType = DBG_OBJ;
00057     pstNew->szName = strdup( szName_ );
00058     pstNew->u32StartAddr = u32Addr_;
00059     pstNew->u32EndAddr = u32Addr_ + u32Len_ - 1;
00060
00061     u32ObjCount++;
00062 }
00063
00064
00065 //-----
00066 uint32_t Symbol_Get_Obj_Count( void )
00067 {
00068     return u32ObjCount;
00069 }
00070
00071 //-----
00072 uint32_t Symbol_Get_Func_Count( void )
00073 {
00074     return u32FuncCount;
00075 }
00076
00077 //-----
00078 Debug_Symbol_t *Symbol_Func_At_Index( uint32_t u32Index_ )
00079 {
00080     if (u32Index_ >= u32FuncCount)
00081     {
00082         return 0;
00083     }
00084     return &pstFuncSymbols[u32Index_];
00085 }
00086
00087 //-----
00088 Debug_Symbol_t *Symbol_Obj_At_Index( uint32_t u32Index_ )
00089 {
00090     if (u32Index_ >= u32ObjCount)
00091     {
00092         return 0;
00093     }
00094     return &pstObjSymbols[u32Index_];
00095 }
00096
00097 //-----
00098 Debug_Symbol_t *Symbol_Find_Func_By_Name( const char *szName_ )
00099 {
00100     uint32_t i = 0;
00101     for (i = 0; i < u32FuncCount; i++)
00102     {
00103         if (0 == strcmp( szName_, pstFuncSymbols[i].szName ))
00104         {
00105             return &pstFuncSymbols[i];
00106         }
00107     }
00108     return 0;

```

```

00109 }
00110
00111 //-----
00112 Debug_Symbol_t *Symbol_Find_Obj_By_Name( const char *szName_ )
00113 {
00114     uint32_t i = 0;
00115     for (i = 0; i < u32ObjCount; i++)
00116     {
00117         if (0 == strcmp(szName_, pstObjSymbols[i].szName))
00118         {
00119             return &pstObjSymbols[i];
00120         }
00121     }
00122     return 0;
00123 }
00124
00125

```

## 4.65 src/debug/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.*

### 4.65.1 Detailed Description

Symbolic debugging support for data and functions.

Definition in file [debug\\_sym.h](#).

### 4.65.2 Function Documentation

#### 4.65.2.1 Symbol\_Add\_Func()

```
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

<i>szName_</i>	- Name of the symbol (string)
<i>u32Addr_</i>	- Start address of the function
<i>u32Len_</i>	- Size of the function (in bytes)

Definition at line 36 of file [debug\\_sym.c](#).

#### 4.65.2.2 Symbol\_Add\_Obj()

```
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

<i>szName_</i>	- Name of the symbol (string)
<i>u32Addr_</i>	- Start address of the object
<i>u32Len_</i>	- Size of the object (in bytes)



Definition at line 51 of file [debug\\_sym.c](#).

#### 4.65.2.3 Symbol\_Find\_Func\_By\_Name()

```
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

<i>szName_</i>	- 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](#).

#### 4.65.2.4 Symbol\_Find\_Obj\_By\_Name()

```
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

<i>szName_</i>	- 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](#).

#### 4.65.2.5 Symbol\_Func\_At\_Index()

```
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

<i>u32↔ Index_</i>	- 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](#).

#### 4.65.2.6 Symbol\_Get\_Func\_Count()

```
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](#).

#### 4.65.2.7 Symbol\_Get\_Obj\_Count()

```
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](#).

#### 4.65.2.8 Symbol\_Obj\_At\_Index()

```
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

<code>u32↔ Index_</code>	- 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](#).

## 4.66 debug\_sym.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/ ( ((/ (      \      ( ((/ (      | -- [ Funkenstein ] -----
00005 *      /( ) )/( ) ((( ( )\      / ( )      | -- [ Little ] -----
00006 *      ( ) _ ( )      )\ _ )\ ( ( ( ( )      | -- [ AVR ] -----
00007 *      | _ | |      ( ) _ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \      \ V / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _ _ | / _ \      \ / / | _ \      |
00010 *      | _ | | _ _ | / _ \      \ / / | _ \      | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __DEBUG_SYM_H__
00022 #define __DEBUG_SYM_H__
00023
00024 #include <stdint.h>
00025
00026 //-----
00027 typedef enum
00028 {
00029     DBG_OBJ = 0,
00030     DBG_FUNC,
00031 } Debug_t;
00032
00033 //-----
00034
00035 //-----
00036 typedef struct
00037 {
00038     Debug_t      eType;
00039     uint32_t      u32StartAddr;
00040     uint32_t      u32EndAddr;
00041     const char *szName;
00042
00043     uint64_t      u64TotalRefs;
00044     uint64_t      u64EpochRefs;
00045 } Debug_Symbol_t;
00046
00047 //-----
00057 void Symbol_Add_Func( const char *szName_, const uint32_t u32Addr_, const uint32_t u32Len_ )
00058 ;
00059 //-----
00070 void Symbol_Add_Obj( const char *szName_, const uint32_t u32Addr_, const uint32_t u32Len_ );
00071
00072 //-----
00080 uint32_t Symbol_Get_Obj_Count( void );
00081
00082 //-----
00090 uint32_t Symbol_Get_Func_Count( void );
00091
00092 //-----
00102 Debug_Symbol_t *Symbol_Func_At_Index( uint32_t u32Index_ );
00103
00104 //-----
00114 Debug_Symbol_t *Symbol_Obj_At_Index( uint32_t u32Index_ );
00115
00116 //-----
00126 Debug_Symbol_t *Symbol_Find_Func_By_Name( const char *szName_ );
00127
00128 //-----
00137 Debug_Symbol_t *Symbol_Find_Obj_By_Name( const char *szName_ );
00138
00139 #endif

```

## 4.67 src/debug/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.*

### 4.67.1 Function Documentation

#### 4.67.1.1 ELF\_PrintHeader()

```
void ELF_PrintHeader (
    const uint8_t * pau8Buffer_ )
```

*ELF\_PrintHeader.*

Print the contents of a loaded ELF file's header data to standard output.

#### Parameters

<i>pau8↵ Buffer_</i>	Buffer containing the loaded ELF contents
--------------------------	---

Definition at line 33 of file [elf\\_print.c](#).

#### 4.67.1.2 ELF\_PrintProgramHeaders()

```
void ELF_PrintProgramHeaders (
    const uint8_t * pau8Buffer_ )
```

ELF\_PrintProgramHeaders.

Print the list of program headers stored in the loaded ELF file .

#### Parameters

<i>pau8↵ Buffer_</i>	Buffer containing the loaded ELF contents
--------------------------	---

Definition at line 246 of file [elf\\_print.c](#).

#### 4.67.1.3 ELF\_PrintSections()

```
void ELF_PrintSections (
    const uint8_t * pau8Buffer_ )
```

ELF\_PrintSections.

Print a list of named sections contained in the loaded ELF file.

#### Parameters

<i>pau8↵ Buffer_</i>	Buffer containing the loaded ELF contents
--------------------------	---

Definition at line 147 of file [elf\\_print.c](#).

#### 4.67.1.4 ELF\_PrintSymbols()

```
void ELF_PrintSymbols (
    const uint8_t * pau8Buffer_ )
```

ELF\_PrintSymbols.

Print a list of ELF Symbols contained in the loaded ELF file.

#### Parameters

<i>pau8↵ Buffer_</i>	Buffer containing the loaded ELF contents
--------------------------	---

Definition at line 192 of file [elf\\_print.c](#).

## 4.68 elf\_print.c

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

```

00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( ((/(      (      | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) (( ( ) \ ) \ / ( ) | -- [ Little ] -----
00006 *      ( ) _ ( )      ) \ _ \ ( ( ( _ ) | -- [ AVR ] -----
00007 *      | _ | | _      ( _ \ ( ) \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ V / / | _ \ | -- [ Runtime ] -----
00009 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00010 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00011 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00012 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00013 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00014 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00015 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00016 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00017 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00018 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00019 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00020 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00021 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00022 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00023 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00024 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00025 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00026 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00027 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00028 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00029 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00030 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00031 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00032 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00033 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00034 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00035 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00036 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00037 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00038 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00039 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00040 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00041 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00042 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00043 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00044 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00045 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00046 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00047 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00048 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00049 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00050 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00051 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00052 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00053 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00054 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00055 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00056 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00057 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00058 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00059 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00060 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00061 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00062 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00063 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00064 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00065 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00066 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00067 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00068 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00069 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00070 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00071 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00072 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00073 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00074 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00075 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00076 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00077 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00078 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00079 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00080 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00081 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00082 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00083 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00084 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00085 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00086 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00087 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00088 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00089 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00090 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00091 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00092 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00093 *      | _ | | _      / _ \ \ \ V / / | _ \ |
00094 *      | _ | | _      / _ \ \ \ V / / | _ \ |

```

```

00095     case ELF_OSABI_OPENBSD:      printf( "OpenBSD]\n" ); break;
00096     default:
00097         printf( "unknown (0x%02X)]\n", pstHeader->u8IdentABI );
00098         break;
00099     }
00100
00101     printf( "--[ABI Version: 0x%02X]\n", pstHeader->u8IdentABIVersion );
00102
00103     printf( "--[Binary Type: " );
00104     switch (pstHeader->u16Type)
00105     {
00106     case ELF_TYPE_RELOCATABLE:    printf( "Relocatable]\n"); break;
00107     case ELF_TYPE_EXECUTABLE:    printf( "Executable]\n"); break;
00108     case ELF_TYPE_SHARED:        printf( "Shared]\n"); break;
00109     case ELF_TYPE_CORE:          printf( "Core]\n"); break;
00110     default:
00111         printf( "unknown (0x%04X)]\n", pstHeader->u16Type );
00112         break;
00113     }
00114
00115     printf( "--[Machine Type: " );
00116     switch (pstHeader->u16Machine)
00117     {
00118     case ELF_MACHINE_SPARC:       printf( "SPARC]\n" ); break;
00119     case ELF_MACHINE_X86:        printf( "x86]\n" ); break;
00120     case ELF_MACHINE_MIPS:       printf( "MIPS]\n" ); break;
00121     case ELF_MACHINE_POWERPC:    printf( "PowerPC]\n" ); break;
00122     case ELF_MACHINE_ARM:        printf( "ARM]\n" ); break;
00123     case ELF_MACHINE_SUPERH:     printf( "SuperH]\n" ); break;
00124     case ELF_MACHINE_IA64:       printf( "IA64]\n" ); break;
00125     case ELF_MACHINE_X86_64:     printf( "x86-64]\n" ); break;
00126     case ELF_MACHINE_AARCH64:    printf( "AArch64]\n" ); break;
00127     case ELF_MACHINE_AVR:        printf( "Atmel AVR]\n" ); break;
00128     default:
00129         printf( "unknown (0x%04X)]\n", pstHeader->u16Machine );
00130         break;
00131     }
00132
00133     printf( "--[Version: 0x%08X]\n",          pstHeader->u32Version );
00134     printf( "--[Entry Point: 0x%08X]\n",      pstHeader->u32EntryPoint );
00135     printf( "--[Program Header Offset: 0x%08X]\n", pstHeader->u32PHOffset );
00136     printf( "--[Section Header Offset: 0x%08X]\n", pstHeader->u32SHOffset );
00137     printf( "--[Flags: 0x%08X]\n",            pstHeader->u32Flags );
00138     printf( "--[Elf Header Size: %d]\n",       pstHeader->u16EHSize );
00139     printf( "--[Program Header Size: %d]\n",   pstHeader->u16PHSize );
00140     printf( "--[Program Header Count: %d]\n",  pstHeader->u16PHNum );
00141     printf( "--[Section Header Size: %d]\n",   pstHeader->u16SHSize );
00142     printf( "--[Section Header Count: %d]\n",  pstHeader->u16SHNum );
00143     printf( "--[Section Header Index: %d]\n",  pstHeader->u16SHIndex );
00144 }
00145
00146 //-----
00147 void ELF_PrintSections( const uint8_t *pau8Buffer_ )
00148 {
00149     ElfHeader_t *pstHeader = (ElfHeader_t*)pau8Buffer_;
00150     uint32_t u32StringOffset = ELF_GetHeaderStringTableOffset( pau8Buffer_ );
00151
00152     uint32_t u32Offset;
00153     uint16_t u16SHCount;
00154
00155     u32Offset = pstHeader->u32SHOffset;
00156     u16SHCount = pstHeader->u16SHNum;
00157
00158     while (u16SHCount)
00159     {
00160         ElfSectionHeader_t *pstSHeader = (ElfSectionHeader_t*)&pau8Buffer_[u32Offset];
00161         printf( "\n--[Section header @ 0x%08X]\n", u32Offset );
00162         printf( "--[Name          %s]\n", &pau8Buffer_[u32StringOffset + pstSHeader->u32Name] );
00163
00164         printf( "--[Type          " );
00165         switch (pstSHeader->u32Type)
00166         {
00167         case ELF_SECTION_TYPE_NULL:    printf( "NULL]\n" ); break;
00168         case ELF_SECTION_TYPE_NOBITS:  printf( "NOBITS]\n" ); break;
00169         case ELF_SECTION_TYPE_PROGBITS: printf( "PROGBITS]\n" ); break;
00170         case ELF_SECTION_TYPE_STRTAB:  printf( "STRTAB]\n" ); break;
00171         case ELF_SECTION_TYPE_SYMTAB:  printf( "SYMTAB]\n" ); break;
00172         default:
00173             printf( " (unknown) 0x%08X]\n", pstSHeader->u32Type );
00174             break;
00175         }
00176
00177         printf( "--[Flags          @ 0x%08X]\n", pstSHeader->u32Flags );
00178         printf( "--[Address        @ 0x%08X]\n", pstSHeader->u32Address );
00179         printf( "--[Offset        @ 0x%08X]\n", pstSHeader->u32Offset );
00180         printf( "--[Size          @ 0x%08X]\n", pstSHeader->u32Size );
00181         printf( "--[Link          @ 0x%08X]\n", pstSHeader->u32Link );

```

```

00182         printf( "--[Info      @ 0x%08X]\n", pstSHeader->u32Info );
00183         printf( "--[Alignment @ 0x%08X]\n", pstSHeader->u32Alignment );
00184         printf( "--[Entry Size @ 0x%08X]\n", pstSHeader->u32EntrySize );
00185         //--
00186         u16SHCount--;
00187         u32Offset += (pstHeader->u16SHSize);
00188     }
00189 }
00190
00191 //-----
00192 void ELF_PrintSymbols( const uint8_t *pau8Buffer_ )
00193 {
00194     // Get a pointer to the section header for the symbol table
00195     uint32_t u32Offset = ELF_GetSymbolTableOffset( pau8Buffer_ );
00196     ElfSectionHeader_t *pstSymHeader = (ElfSectionHeader_t*)(&pau8Buffer_[u32Offset]);
00197
00198     // Get a pointer to the section header for the symbol table's strings
00199     u32Offset = ELF_GetSymbolStringTableOffset( pau8Buffer_ );
00200     ElfSectionHeader_t *pstStrHeader = (ElfSectionHeader_t*)(&pau8Buffer_[u32Offset]);
00201
00202     // Iterate through the symbol table section, printing out the details of each.
00203     uint32_t u32SymOffset = pstSymHeader->u32Offset;
00204     ElfSymbol_t *pstSymbol = (ElfSymbol_t*)(&pau8Buffer_[u32SymOffset]);
00205
00206     printf( "VALUE SIZE TYPE SCOPE ID NAME\n");
00207     while (u32SymOffset < (pstSymHeader->u32Offset + pstSymHeader->u32Size))
00208     {
00209         printf( "%08X, ", pstSymbol->u32Value );
00210         printf( "%5d, ", pstSymbol->u32Size );
00211         uint8_t u8Type = pstSymbol->u8Info & 0x0F;
00212         switch (u8Type)
00213         {
00214             case 0:      printf( "NOTYPE, " ); break;
00215             case 1:      printf( "OBJECT, " ); break;
00216             case 2:      printf( "FUNC, " ); break;
00217             case 3:      printf( "SECTION, " ); break;
00218             case 4:      printf( "FILE, " ); break;
00219             default:     printf( "Unknown (%02X), ", u8Type); break;
00220         }
00221         u8Type = (pstSymbol->u8Info >> 4) & 0x0F;
00222         switch (u8Type)
00223         {
00224             case 0:      printf( "LOCAL, " ); break;
00225             case 1:      printf( "GLOBAL, " ); break;
00226             case 2:      printf( "WEAK, " ); break;
00227             default:     printf( "Unknown (%02X), ", u8Type); break;
00228         }
00229
00230         if (65521 == pstSymbol->u16SHIndex) // 65521 == special value "ABS"
00231         {
00232             printf( " ABS, ");
00233         }
00234         else
00235         {
00236             printf( "%5d, ", pstSymbol->u16SHIndex );
00237         }
00238         printf( "%s\n", &pau8Buffer_[pstSymbol->u32Name + pstStrHeader->u32Offset] );
00239
00240         u32SymOffset += pstSymHeader->u32EntrySize;
00241         pstSymbol = (ElfSymbol_t*)(&pau8Buffer_[u32SymOffset]);
00242     }
00243 }
00244
00245 //-----
00246 void ELF_PrintProgramHeaders( const uint8_t *pau8Buffer_ )
00247 {
00248     ElfHeader_t *pstHeader = (ElfHeader_t*)(pau8Buffer_);
00249     uint32_t u32Offset = pstHeader->u32PHOffset;
00250     uint32_t u16Count = pstHeader->u16PHNum;
00251
00252     while (u16Count)
00253     {
00254         ElfProgramHeader_t *pstProgHeader = (ElfProgramHeader_t*)(&pau8Buffer_[u32Offset]);
00255         printf( "Program Header:\n" );
00256         printf( "--[Type:      %08X]\n", pstProgHeader->u32Type );
00257         printf( "--[Offset:    %08X]\n", pstProgHeader->u32Offset );
00258         printf( "--[VAddr:     %08X]\n", pstProgHeader->u32VirtualAddress );
00259         printf( "--[PAddr:     %08X]\n", pstProgHeader->u32PhysicalAddress );
00260         printf( "--[FileSize:  %08X]\n", pstProgHeader->u32FileSize );
00261         printf( "--[MemSize:   %08X]\n", pstProgHeader->u32MemSize );
00262         printf( "--[Flags:     %08X]\n", pstProgHeader->u32Flags );
00263         printf( "--[Alignment: %08X]\n", pstProgHeader->u32Alignment );
00264
00265         //--
00266         u32Offset += pstHeader->u16PHSize;
00267         u16Count--;
00268     }

```



```
00269 }
```

## 4.69 src/debug/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.*

### 4.69.1 Detailed Description

Functions to print information from ELF files.

Definition in file [elf\\_print.h](#).

### 4.69.2 Function Documentation

#### 4.69.2.1 ELF\_PrintHeader()

```
void ELF_PrintHeader (
    const uint8_t * pau8Buffer_ )
```

[ELF\\_PrintHeader.](#)

Print the contents of a loaded ELF file's header data to standard output.

#### Parameters

<i>pau8Buffer_</i>	Buffer containing the loaded ELF contents
--------------------	---

Definition at line 33 of file [elf\\_print.c](#).

#### 4.69.2.2 ELF\_PrintProgramHeaders()

```
void ELF_PrintProgramHeaders (
    const uint8_t * pau8Buffer_ )
```

ELF\_PrintProgramHeaders.

Print the list of program headers stored in the loaded ELF file .

##### Parameters

<i>pau8↵ Buffer_</i>	Buffer containing the loaded ELF contents
--------------------------	---

Definition at line 246 of file [elf\\_print.c](#).

#### 4.69.2.3 ELF\_PrintSections()

```
void ELF_PrintSections (
    const uint8_t * pau8Buffer_ )
```

ELF\_PrintSections.

Print a list of named sections contained in the loaded ELF file.

##### Parameters

<i>pau8↵ Buffer_</i>	Buffer containing the loaded ELF contents
--------------------------	---

Definition at line 147 of file [elf\\_print.c](#).

#### 4.69.2.4 ELF\_PrintSymbols()

```
void ELF_PrintSymbols (
    const uint8_t * pau8Buffer_ )
```

ELF\_PrintSymbols.

Print a list of ELF Symbols contained in the loaded ELF file.

## Parameters

<i>pau8</i> ↵ <i>Buffer_</i>	Buffer containing the loaded ELF contents
---------------------------------	---

Definition at line 192 of file [elf\\_print.c](#).

## 4.70 elf\_print.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( (/( (      (      | -- [ Funkenstein ] -----
00005 *      /( ) /( ) ((( ( ) \      | -- [ Little ] -----
00006 *      ( ) _ ( )      )\ _ )\ ( ( ( _ )      | -- [ AVR ] -----
00007 *      | _ | _ |      ( _ \ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | _ |      / _ \ \ v / | _ /      | -- [ Runtime ] -----
00009 *      | _ | _ |      / _ \ \ \ / | _ \      |
00010 *                                          | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __ELF_PRINT_H__
00022 #define __ELF_PRINT_H__
00023
00024 #include "elf_types.h"
00025 #include <stdint.h>
00026
00027 //-----
00035 void ELF_PrintHeader( const uint8_t *pau8Buffer_ );
00036
00037 //-----
00045 void ELF_PrintSections( const uint8_t *pau8Buffer_ );
00046
00047 //-----
00055 void ELF_PrintSymbols( const uint8_t *pau8Buffer_ );
00056
00057 //-----
00065 void ELF_PrintProgramHeaders( const uint8_t *pau8Buffer_ );
00066
00067 #endif //__ELF_PRINT_H__

```

## 4.71 src/debug/elf\_types.h File Reference

Defines and types used by ELF loader and supporting functionality.

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

## 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)`

## Functions

- `struct __attribute__ ((packed))`

## Variables

- `ElfHeader_t`
- `ElfProgramHeader_t`
- `ElfSectionHeader_t`
- `ElfSymbol_t`

### 4.71.1 Detailed Description

Defines and types used by ELF loader and supporting functionality.

Definition in file [elf\\_types.h](#).

## 4.72 elf\_types.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( (/(      (      | -- [ Funkenstein ] -----
00005 *      /(\ ) /(\ )(((\ ( )\ /(\ ) | -- [ Little ] -----
00006 *      (\)_(\ )      )\ _ )\ ((\ ((\ ) | -- [ AVR ] -----
00007 *      | _ | |      (\)_(\ )\ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ / / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ | / _ \ \ \ / / | _ \ |
00010 *      | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __ELF_TYPES_H__
00022 #define __ELF_TYPES_H__
00023
00024 #include <stdint.h>
00025
00026 //-----
00027 #define ELF_MAGIC_NUMBER      ((uint32_t)0x464C457F) // "~ELF"
00028
00029 #define ELF_CLASS_32BIT      ((uint8_t)1)
00030 #define ELF_CLASS_64BIT      ((uint8_t)2)
00031
00032 #define ELF_ENDIAN_LITTLE      ((uint8_t)1)
00033 #define ELF_ENDIAN_BIG      ((uint8_t)2)
00034
00035 #define ELF_IDENT_VERSION_ORIGINAL ((uint8_t)1)
00036
00037 #define ELF_OSABI_SYSV      ((uint8_t)0x00)
00038 #define ELF_OSABI_HPUX      ((uint8_t)0x01)
00039 #define ELF_OSABI_NETBSD      ((uint8_t)0x02)
00040 #define ELF_OSABI_LINUX      ((uint8_t)0x03)
00041 #define ELF_OSABI_SOLARIS      ((uint8_t)0x06)
00042 #define ELF_OSABI_AIX      ((uint8_t)0x07)
00043 #define ELF_OSABI_TRIX      ((uint8_t)0x08)
00044 #define ELF_OSABI_FREEBSD      ((uint8_t)0x09)
00045 #define ELF_OSABI_OPENBSD      ((uint8_t)0x0C)
00046
00047 #define ELF_TYPE_RELOCATABLE      ((uint8_t)0x01)
00048 #define ELF_TYPE_EXECUTABLE      ((uint8_t)0x02)
00049 #define ELF_TYPE_SHARED      ((uint8_t)0x03)
00050 #define ELF_TYPE_CORE      ((uint8_t)0x04)
00051
00052 #define ELF_MACHINE_SPARC      ((uint16_t)0x02)
00053 #define ELF_MACHINE_X86      ((uint16_t)0x03)
00054 #define ELF_MACHINE_MIPS      ((uint16_t)0x08)
00055 #define ELF_MACHINE_POWERPC      ((uint16_t)0x14)
00056 #define ELF_MACHINE_ARM      ((uint16_t)0x28)
00057 #define ELF_MACHINE_SUPERH      ((uint16_t)0x2A)
00058 #define ELF_MACHINE_IA64      ((uint16_t)0x32)
00059 #define ELF_MACHINE_X86_64      ((uint16_t)0x3E)
00060 #define ELF_MACHINE_AVR      ((uint16_t)0x53)
00061 #define ELF_MACHINE_AARCH64      ((uint16_t)0xB7)
00062
00063 #define ELF_VERSION_ORIGINAL      ((uint32_t)1)
00064
00065 #define ELF_SECTION_TYPE_NULL      ((uint32_t)0)
00066 #define ELF_SECTION_TYPE_PROGBITS      ((uint32_t)1)
00067 #define ELF_SECTION_TYPE_SYMTAB      ((uint32_t)2)
00068 #define ELF_SECTION_TYPE_STRTAB      ((uint32_t)3)
00069 #define ELF_SECTION_TYPE_NOBITS      ((uint32_t)8)
00070
00071 //-----
00072 typedef struct __attribute__((packed))
00073 {
00074     // (Explicit line breaks to show 32-bit alignment)
00075     //---- 0x00
00076     uint32_t u32IdentMagicNumber;
00077
00078     //---- 0x04
00079     uint8_t u8IdentFormat;
00080     uint8_t u8IdentEndianness;
00081     uint8_t u8IdentVersion;
00082     uint8_t u8IdentABI;
00083
00084     //---- 0x08
00085     uint8_t u8IdentABIVersion;
00086     uint8_t u8Pad1[7];
00087
00088     //---- 0x10
00089     uint16_t u16Type;
00090     uint16_t u16Machine;

```

```

00091
00092 //---- 0x14
00093 uint32_t    u32Version;
00094
00095 //---- 0x18
00096 uint32_t    u32EntryPoint;
00097
00098 //---- 0x1C
00099 uint32_t    u32PHOffset;
00100
00101 //---- 0x20
00102 uint32_t    u32SHOffset;
00103
00104 //---- 0x24
00105 uint32_t    u32Flags;
00106
00107 //---- 0x28
00108 uint16_t    u16EHSize;
00109 uint16_t    u16PHSize;
00110
00111 //---- 0x2C
00112 uint16_t    u16PHNum;
00113 uint16_t    u16SHSize;
00114
00115 //---- 0x30
00116 uint16_t    u16SHNum;
00117 uint16_t    u16SHIndex;
00118
00119 } ElfHeader_t;
00120
00121 //-----
00122 typedef struct __attribute__((packed))
00123 {
00124     uint32_t    u32Type;
00125     uint32_t    u32Offset;
00126     uint32_t    u32VirtualAddress;
00127     uint32_t    u32PhysicalAddress;
00128     uint32_t    u32FileSize;
00129     uint32_t    u32MemSize;
00130     uint32_t    u32Flags;
00131     uint32_t    u32Alignment;
00132 } ElfProgramHeader_t;
00133
00134 //-----
00135 typedef struct __attribute__((packed))
00136 {
00137     uint32_t    u32Name;
00138     uint32_t    u32Type;
00139     uint32_t    u32Flags;
00140     uint32_t    u32Address;
00141     uint32_t    u32Offset;
00142     uint32_t    u32Size;
00143     uint32_t    u32Link;
00144     uint32_t    u32Info;
00145     uint32_t    u32Alignment;
00146     uint32_t    u32EntrySize;
00147 } ElfSectionHeader_t;
00148
00149 //-----
00150 typedef struct __attribute__((packed))
00151 {
00152     uint32_t    u32Name;
00153     uint32_t    u32Value;
00154     uint32_t    u32Size;
00155     uint8_t     u8Info;
00156     uint8_t     u8Other;
00157     uint16_t    u16SHIndex;
00158 } ElfSymbol_t;
00159
00160
00161 #endif //__ELF_TYPES_H__

```

## 4.73 src/debug/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 [blsInteractive](#)  
*"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** []

## 4.73.1 Detailed Description

Interactive debugging support.

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

Definition in file [interactive.c](#).

## 4.73.2 Typedef Documentation

### 4.73.2.1 Interactive\_Handler

```
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](#).



### 4.73.3 Function Documentation

#### 4.73.3.1 Interactive\_Break()

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

Interactive\_Break.

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

##### Parameters

<i>sz↔ Command_</i>	command-line data passed in by the user.
-------------------------	--

##### Returns

false - continue interactive debugging

Definition at line 478 of file [interactive.c](#).

#### 4.73.3.2 Interactive\_BreakFunc()

```
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

<i>sz↔ Command_</i>	command-line data passed in by the user.
-------------------------	--

##### Returns

false - continue interactive debugging

Definition at line 667 of file [interactive.c](#).

#### 4.73.3.3 Interactive\_CheckAndExecute()

```
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](#).

#### 4.73.3.4 Interactive\_Continue()

```
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

<i>sz↔ Command_</i>	commnd-line data passed in by the user
-------------------------	--

##### Returns

true - exit interactive debugging

Definition at line 470 of file [interactive.c](#).

#### 4.73.3.5 Interactive\_Disasm()

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

Interactive\_Disasm.

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

##### Parameters

<i>sz↔ Command_</i>	command-line data passed in by the user.
-------------------------	--

**Returns**

false - continue interactive debugging

Definition at line 646 of file [interactive.c](#).

**4.73.3.6 Interactive\_EE()**

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

Interactive\_EE.

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

**Parameters**

<i>sz↔ Command_</i>	command-line data passed in by the user.
-------------------------	--

**Returns**

false - continue interactive debugging

Definition at line 586 of file [interactive.c](#).

**4.73.3.7 Interactive\_Help()**

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

Interactive\_Help.

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

**Parameters**

<i>sz↔ Command_</i>	command-line data passed in by the user.
-------------------------	--

**Returns**

false - continue interactive debugging

Definition at line 633 of file [interactive.c](#).

#### 4.73.3.8 Interactive\_Init()

```
void Interactive_Init (
    TraceBuffer_t * pstTrace_ )
```

Interactive\_Init.

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

##### Parameters

<i>pst↔ Trace_</i>	Pointer to the tracebuffer object
------------------------	-----------------------------------

Definition at line 382 of file [interactive.c](#).

#### 4.73.3.9 Interactive\_ListFunc()

```
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

<i>sz↔ Command_</i>	command-line data passed in by the user.
-------------------------	--

##### Returns

false - continue interactive debugging

Definition at line 783 of file [interactive.c](#).

#### 4.73.3.10 Interactive\_ListObj()

```
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

<i>sz↔ Command_</i>	command-line data passed in by the user.
-------------------------	--

## Returns

false - continue interactive debugging

Definition at line 763 of file [interactive.c](#).

## 4.73.3.11 Interactive\_Quit()

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

Interactive\_Quit.

Stop debugging, and exit fI AVR.

## Parameters

<i>sz↔ Command_</i>	command-line data passed in by the user.
-------------------------	--

## Returns

N/A - does not return (program terminates)

Definition at line 620 of file [interactive.c](#).

## 4.73.3.12 Interactive\_RAM()

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

Interactive\_RAM.

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

## Parameters

<i>sz↔ Command_</i>	command-line data passed in by the user.
-------------------------	--

**Returns**

false - continue interactive debugging

Definition at line 559 of file [interactive.c](#).

**4.73.3.13 Interactive\_Registers()**

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

Interactive\_Registers.

Display the contents of the core CPU registers on the console

**Parameters**

<i>sz↔ Command_</i>	command-line data passed in by the user.
-------------------------	--

**Returns**

false - continue interactive debugging

Definition at line 613 of file [interactive.c](#).

**4.73.3.14 Interactive\_ROM()**

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

Interactive\_ROM.

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

**Parameters**

<i>sz↔ Command_</i>	command-line data passed in by the user.
-------------------------	--

**Returns**

false - continue interactive debugging

Definition at line 532 of file [interactive.c](#).

#### 4.73.3.15 Interactive\_Set()

```
void Interactive_Set (
    void )
```

Interactive\_Set.

Enable interactive-debug mode on the next instruction cycle.

Definition at line 361 of file [interactive.c](#).

#### 4.73.3.16 Interactive\_Step()

```
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

<i>sz↵ Command_</i>	commnd-line data passed in by the user
-------------------------	--

##### Returns

true - exit interactive debugging

Definition at line 626 of file [interactive.c](#).

#### 4.73.3.17 Interactive\_Trace()

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

Interactive\_Trace.

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

##### Parameters

<i>sz↵ Command_</i>	command-line data passed in by the user.
-------------------------	--

**Returns**

false - continue interactive debugging

Definition at line 660 of file [interactive.c](#).

**4.73.3.18 Interactive\_Watch()**

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

Interactive\_Watch.

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

**Parameters**

<i>sz↔ Command_</i>	command-line data passed in by the user.
-------------------------	--

**Returns**

false - continue interactive debugging

Definition at line 506 of file [interactive.c](#).

**4.73.3.19 Interactive\_WatchObj()**

```
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**

<i>sz↔ Command_</i>	command-line data passed in by the user.
-------------------------	--

**Returns**

false - continue interactive debugging

Definition at line 711 of file [interactive.c](#).



## 4.73.4 Variable Documentation

### 4.73.4.1 astCommands

```
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](#).

## 4.74 interactive.c

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( (/(      )\      | -- [ Funkenstein ] -----
00005 *      /( ) )/( ) ((( ( )\ )\ /( )      | -- [ Little ] -----
00006 *      ( )_ ( )      )\ _ )\ ( ( ( )      | -- [ AVR ] -----
00007 *      | | _ | |      ( )_ \ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ V / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _ _ | / _ \ \ \ / | _ \      |
00010 *      |                                     | "Yeah, it does Arduino..."
00011 *      +-----+
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00023 #include "emu_config.h"
00024 #include "avr_cpu.h"
00025 #include "avr_cpu_print.h"
00026 #include "watchpoint.h"
00027 #include "breakpoint.h"
00028 #include "avr_disasm.h"
00029 #include "trace_buffer.h"
00030 #include "debug_sym.h"
00031 #include "write_callout.h"
00032
00033 #include <stdint.h>
```

```

00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037
00038 //-----
00046 typedef bool (*Interactive_Handler)( char *szCommand_ );
00047
00048 //-----
00052 typedef struct
00053 {
00054     const char *szCommand;
00055     const char *szDescription;
00056     Interactive_Handler pfHandler;
00057 } Interactive_Command_t;
00058
00059 //-----
00060 static bool bIsInteractive;
00061 static bool bRetrigger;
00062
00063 static TraceBuffer_t *pstTrace = 0;
00064
00065 //-----
00075 static bool Interactive_Continue( char *szCommand_ );
00076
00077 //-----
00087 static bool Interactive_Step( char *szCommand_ );
00088
00089 //-----
00098 static bool Interactive_Break( char *szCommand_ );
00099
00100 //-----
00109 static bool Interactive_Watch( char *szCommand_ );
00110
00111 //-----
00120 static bool Interactive_ROM( char *szCommand_ );
00121
00122 //-----
00131 static bool Interactive_RAM( char *szCommand_ );
00132
00133 //-----
00142 static bool Interactive_EE( char *szCommand_ );
00143
00144 //-----
00153 static bool Interactive_Registers( char *szCommand_ );
00154
00155 //-----
00164 static bool Interactive_Quit( char *szCommand_ );
00165
00166 //-----
00176 static bool Interactive_Help( char *szCommand_ );
00177
00178 //-----
00187 static bool Interactive_Disasm( char *szCommand_ );
00188
00189 //-----
00198 static bool Interactive_Trace( char *szCommand_ );
00199
00200 //-----
00211 static bool Interactive_BreakFunc( char *szCommand_ );
00212
00213 //-----
00224 static bool Interactive_WatchObj( char *szCommand_ );
00225
00226 //-----
00236 static bool Interactive_ListObj( char *szCommand_ );
00237
00238 //-----
00248 static bool Interactive_ListFunc( char *szCommand_ );
00249
00250 //-----
00251 // Command-handler table
00252 static Interactive_Command_t astCommands[] =
00253 {
00254     { "registers", "Dump registers to console", Interactive_Registers },
00255     { "continue", "Continue execution", Interactive_Continue },
00256     { "disasm", "Show disassembly", Interactive_Disasm },
00257     { "trace", "Dump tracebuffer to console", Interactive_Trace },
00258     { "break", "Toggle breakpoint at address", Interactive_Break },
00259     { "watch", "Toggle watchpoint at address", Interactive_Watch },
00260     { "lfunc", "List Functions", Interactive_ListFunc },
00261     { "help", "List commands", Interactive_Help },
00262     { "step", "Step to next instruction", Interactive_Step },
00263     { "quit", "Quit emulator", Interactive_Quit },
00264     { "lobj", "List Objects", Interactive_ListObj },
00265     { "bsym", "Toggle breakpoint at function referenced by symbol",
Interactive_BreakFunc },
00266     { "wobj", "Toggle watchpoint on object referenced by symbol",

```

```

    Interactive_WatchObj },
00267 { "reg",      "Dump registers to console", Interactive_Registers },
00268 { "rom",      "Dump x bytes of ROM to console", Interactive_ROM },
00269 { "ram",      "Dump x bytes of RAM to console", Interactive_RAM },
00270 { "ee",      "Dump x bytes of RAM to console", Interactive_EE },
00271 { "b",      "toggle breakpoint at address", Interactive_Break },
00272 { "c",      "continue execution", Interactive_Continue },
00273 { "d",      "show disassembly", Interactive_Disasm },
00274 { "w",      "toggle watchpoint at address", Interactive_Watch },
00275 { "q",      "Quit emulator", Interactive_Quit },
00276 { "s",      "Step to next instruction", Interactive_Step },
00277 { "t",      "Dump tracebuffer to console", Interactive_Trace},
00278 { "h",      "List commands", Interactive_Help },
00279 { 0 }
00280 };
00281
00282 //-----
00283 static bool Interactive_Execute_i( void )
00284 {
00285     // Interactive mode - grab a line from standard input.
00286     char szCmdBuf[256];
00287     int iCmd = 0;
00288
00289     printf( "> " );
00290
00291     // Bail if stdin reaches EOF...
00292     if (0 == fgets(szCmdBuf, 255, stdin))
00293     {
00294         printf("[EOF]\n");
00295         exit(0);
00296     }
00297
00298     iCmd = strlen(szCmdBuf);
00299     if ( iCmd <= 1 )
00300     {
00301         printf("\n");
00302         iCmd = 0;
00303     }
00304     else
00305     {
00306         szCmdBuf[ iCmd - 1 ] = 0;
00307     }
00308
00309     // Compare command w/elements in the command table
00310     Interactive_Command_t *pstCommand = astCommands;
00311     bool bFound = false;
00312     bool bContinue = false;
00313
00314     while (pstCommand->szCommand)
00315     {
00316         if ( (0 == strcmp(pstCommand->szCommand, szCmdBuf, strlen(pstCommand->
szCmdBuf)))
00317             && ( szCmdBuf[ strlen(pstCommand->szCommand) ] == ' ' ||
szCmdBuf[ strlen(pstCommand->szCommand) ] == '\0' ||
szCmdBuf[ strlen(pstCommand->szCommand) ] == '\n' ||
szCmdBuf[ strlen(pstCommand->szCommand) ] == '\r' ) )
00321         {
00322
00323             // printf( "Found match: %s\n", pstCommand->szCommand );
00324             bFound = true;
00325             bContinue = pstCommand->pHandler( szCmdBuf );
00326             break;
00327         }
00328         // Next command
00329         pstCommand++;
00330     }
00331
00332     if (!bFound)
00333     {
00334         printf( "Invalid Command\n");
00335     }
00336
00337     return bContinue;
00338 }
00339
00340 //-----
00341 void Interactive_CheckAndExecute( void )
00342 {
00343     // If we're in non-interactive mode (i.e. native execution), then return
00344     // out instantly.
00345     if (false == bIsInteractive)
00346     {
00347         if (false == bRetrigger)
00348         {
00349             return;
00350         }
00351         bIsInteractive = true;

```

```

00352         bRetrigger = false;
00353     }
00354     printf( "Debugging @ Address [0x%X]\n", stCPU.u32PC );
00355
00356     // Keep attempting to parse commands until a valid one was encountered
00357     while (!Interactive_Execute_i()) { /* Do Nothing */ }
00358 }
00359
00360 //-----
00361 void Interactive_Set( void )
00362 {
00363     bIsInteractive = true;
00364     bRetrigger = false;
00365 }
00366
00367 //-----
00368 bool Interactive_WatchpointCallback( uint16_t u16Addr_, uint8_t u8Val_ )
00369 {
00370     if (WatchPoint_EnabledAtAddress(u16Addr_))
00371     {
00372         Interactive_Set();
00373         printf( "Watchpoint @ 0x%04X hit. Old Value => %d, New Value => %d\n",
00374             u16Addr_,
00375             stCPU.pstRAM->au8RAM[ u16Addr_ ],
00376             u8Val_ );
00377     }
00378     return true;
00379 }
00380
00381 //-----
00382 void Interactive_Init( TraceBuffer_t *pstTrace_ )
00383 {
00384     pstTrace = pstTrace_;
00385     bIsInteractive = false;
00386     bRetrigger = false;
00387
00388     // Add the watchpoint handler as a wildcard callout (i.e. every write
00389     // triggers is, it's up to the callout to handle filtering on its own).
00390     WriteCallout_Add( Interactive_WatchpointCallback, 0 );
00391 }
00392
00393 //-----
00394 static bool Token_ScanNext( char *szCommand_, int iStart_, int *piTokenStart_, int *piTokenLen_ )
00395 {
00396     int i = iStart_;
00397
00398     // Parse leading whitespace
00399     while ( (szCommand_[i] == ' ') ||
00400         (szCommand_[i] == '\t') ||
00401         (szCommand_[i] == '\r') ||
00402         (szCommand_[i] == '\n') ) { i++; }
00403
00404     // Check null termination
00405     if (szCommand_[i] == '\0' )
00406     {
00407         return false;
00408     }
00409
00410     // Parse token
00411     *piTokenStart_ = i;
00412     while ( (szCommand_[i] != ' ') &&
00413         (szCommand_[i] != '\t') &&
00414         (szCommand_[i] != '\r') &&
00415         (szCommand_[i] != '\n') &&
00416         (szCommand_[i] != '\0') ) { i++; }
00417     *piTokenLen_ = (i - *piTokenStart_);
00418
00419     // printf( "Start, Len: %d, %d\n", i, *piTokenLen_ );
00420     return true;
00421 }
00422
00423 //-----
00424 static bool Token_DiscardNext( char *szCommand_, int iStart_, int *piNextTokenStart_ )
00425 {
00426     int iTempStart;
00427     int iTempLen;
00428     if (!Token_ScanNext(szCommand_, iStart_, &iTempStart, &iTempLen ))
00429     {
00430         return false;
00431     }
00432     *piNextTokenStart_ = iTempStart + iTempLen + 1;
00433     return true;
00434 }
00435
00436 //-----
00437
00438

```

```

00439 //-----
00440 static bool Token_ReadNextHex( char *szCommand_, int iStart_, int *piNextTokenStart_, unsigned int *puiVal_
    )
00441 {
00442     int iTempStart = iStart_;
00443     int iTempLen;
00444     if (!Token_ScanNext( szCommand_, iStart_, &iTempStart, &iTempLen ))
00445     {
00446         return false;
00447     }
00448     szCommand_[iTempStart + iTempLen] = 0;
00449     if (0 == sscanf( &szCommand_[iTempStart], "%x", puiVal_ ))
00450     {
00451         if (0 == sscanf( &szCommand_[iTempStart], "x%x", puiVal_ ))
00452         {
00453             if (0 == sscanf( &szCommand_[iTempStart], "0x%x", puiVal_ ))
00454             {
00455                 printf( "Missing Argument\n" );
00456                 return false;
00457             }
00458         }
00459     }
00460     *piNextTokenStart_ = iTempStart + iTempLen + 1;
00461     return true;
00462 }
00463
00464 //-----
00465 static bool Interactive_Continue( char *szCommand_ )
00466 {
00467     bIsInteractive = false;
00468     bRetrigger = false;
00469     return true;
00470 }
00471
00472 //-----
00473 static bool Interactive_Break( char *szCommand_ )
00474 {
00475     unsigned int uiAddr;
00476     int iTokenStart;
00477     if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
00478     {
00479         return false;
00480     }
00481     if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiAddr))
00482     {
00483         return false;
00484     }
00485     if (BreakPoint_EnabledAtAddress( (uint32_t)uiAddr))
00486     {
00487         BreakPoint_Delete( (uint32_t)uiAddr);
00488     }
00489     else
00490     {
00491         BreakPoint_Insert( (uint32_t)uiAddr);
00492     }
00493     return false;
00494 }
00495
00496 //-----
00497 static bool Interactive_Watch( char *szCommand_ )
00498 {
00499     unsigned int uiAddr;
00500     int iTokenStart;
00501     if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
00502     {
00503         return false;
00504     }
00505     if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiAddr))
00506     {
00507         return false;
00508     }
00509     if (WatchPoint_EnabledAtAddress( (uint16_t)uiAddr))
00510     {
00511         WatchPoint_Delete( (uint16_t)uiAddr);
00512     }
00513 }

```

```
00525     else
00526     {
00527         WatchPoint_Insert( (uint16_t)uiAddr);
00528     }
00529     return false;
00530 }
00531 //-----
00532 static bool Interactive_ROM( char *szCommand_ )
00533 {
00534     unsigned int uiAddr;
00535     unsigned int uiLen;
00536     int iTokenStart;
00537
00538     if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
00539     {
00540         return false;
00541     }
00542
00543     if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiAddr))
00544     {
00545         return false;
00546     }
00547
00548     if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiLen))
00549     {
00550         return false;
00551     }
00552
00553     print_rom( (uint32_t)uiAddr, (uint16_t)uiLen );
00554
00555     return false;
00556 }
00557 //-----
00558 static bool Interactive_RAM( char *szCommand_ )
00559 {
00560     unsigned int uiAddr;
00561     unsigned int uiLen;
00562     int iTokenStart;
00563
00564     if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
00565     {
00566         return false;
00567     }
00568
00569     if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiAddr))
00570     {
00571         return false;
00572     }
00573
00574     if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiLen))
00575     {
00576         return false;
00577     }
00578
00579     print_ram( (uint16_t)uiAddr, (uint16_t)uiLen );
00580
00581     return false;
00582 }
00583 //-----
00584 static bool Interactive_EE( char *szCommand_ )
00585 {
00586     unsigned int uiAddr;
00587     unsigned int uiLen;
00588     int iTokenStart;
00589
00590     if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
00591     {
00592         return false;
00593     }
00594
00595     if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiAddr))
00596     {
00597         return false;
00598     }
00599
00600     if (!Token_ReadNextHex( szCommand_, iTokenStart, &iTokenStart, &uiLen))
00601     {
00602         return false;
00603     }
00604
00605     printf( "Dump EEPROM [%x:%x]\n", uiAddr, uiLen );
00606
00607     return false;
00608 }
00609
00610 }
```

```

00612 //-----
00613 static bool Interactive_Registers( char *szCommand_ )
00614 {
00615     print_core_regs();
00616     return false;
00617 }
00618
00619 //-----
00620 static bool Interactive_Quit( char *szCommand_ )
00621 {
00622     exit(0);
00623 }
00624
00625 //-----
00626 static bool Interactive_Step( char *szCommand_ )
00627 {
00628     bRetrigger = true; // retrigger debugging on next loop
00629     return true;
00630 }
00631
00632 //-----
00633 static bool Interactive_Help( char *szCommand_ )
00634 {
00635     Interactive_Command_t *pstCommand_ = astCommands;
00636     printf( "FLAVR interactive debugger commands:\n");
00637     while (pstCommand_->szCommand)
00638     {
00639         printf( "    %s: %s\n", pstCommand_->szCommand, pstCommand_->
szDescription );
00640         pstCommand_++;
00641     }
00642     return false;
00643 }
00644
00645 //-----
00646 static bool Interactive_Disasm( char *szCommand_ )
00647 {
00648     char szBuf[256];
00649     uint16_t OP = stCPU.pu16ROM[stCPU.u32PC];
00650
00651     printf("0x%04X: [0x%04X] ", stCPU.u32PC, OP);
00652     AVR_Decode(OP);
00653     AVR_Disasm_Function(OP)(szBuf);
00654     printf( "%s", szBuf );
00655
00656     return false;
00657 }
00658
00659 //-----
00660 static bool Interactive_Trace( char *szCommand_ )
00661 {
00662     TraceBuffer_Print( pstTrace, TRACE_PRINT_COMPACT | TRACE_PRINT_DISASSEMBLY );
00663     return false;
00664 }
00665
00666 //-----
00667 static bool Interactive_BreakFunc( char *szCommand_ )
00668 {
00669     unsigned int uiAddr;
00670     unsigned int uiLen;
00671     int iTokenStart;
00672     int iEnd;
00673
00674     if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
00675     {
00676         return false;
00677     }
00678
00679     if (!Token_ScanNext( szCommand_, iTokenStart, &iEnd, &uiLen ) )
00680     {
00681         return false;
00682     }
00683
00684     szCommand_[iTokenStart+uiLen] = 0;
00685
00686     char *szName = &szCommand_[iTokenStart];
00687     Debug_Symbol_t *pstSym = Symbol_Find_Func_By_Name( szName );
00688
00689     if (!pstSym)
00690     {
00691         printf( "Unknown function: %s", szName );
00692         return false;
00693     }
00694     printf( "Name: %s, Start Addr: %x, End Addr: %x\n", pstSym->szName, pstSym->
u32StartAddr, pstSym->u32EndAddr );
00695
00696     if (BreakPoint_EnabledAtAddress(pstSym->

```

```

    u32StartAddr))
00697 {
00698     printf( "Removing breakpoint @ 0x%08X\n", pstSym->u32StartAddr );
00699     BreakPoint_Delete( pstSym->u32StartAddr );
00700 }
00701 else
00702 {
00703     printf( "Inserting breakpoint @ 0x%08X\n", pstSym->u32StartAddr );
00704     BreakPoint_Insert( pstSym->u32StartAddr );
00705 }
00706
00707 return false;
00708 }
00709
00710 //-----
00711 static bool Interactive_WatchObj( char *szCommand_ )
00712 {
00713     unsigned int uiAddr;
00714     unsigned int uiLen;
00715     int iTokenStart;
00716     int iEnd;
00717
00718     if (!Token_DiscardNext( szCommand_, 0, &iTokenStart))
00719     {
00720         return false;
00721     }
00722
00723     if (!Token_ScanNext( szCommand_, iTokenStart, &iEnd, &uiLen ))
00724     {
00725         return false;
00726     }
00727
00728     szCommand_[iTokenStart+uiLen] = 0;
00729
00730     char *szName = &szCommand_[iTokenStart];
00731     Debug_Symbol_t *pstSym = Symbol_Find_Obj_By_Name( szName );
00732
00733     if (!pstSym)
00734     {
00735         printf( "Unknown object: %s", szName );
00736         return false;
00737     }
00738     printf( "Name: %s, Start Addr: %x, End Addr: %x\n", pstSym->szName, pstSym->
u32StartAddr, pstSym->u32EndAddr );
00739
00740     if (WatchPoint_EnabledAtAddress( pstSym->
u32StartAddr))
00741     {
00742         printf( "Removing watchpoint @ 0x%04X\n", pstSym->u32StartAddr );
00743         uint32_t i;
00744         for (i = pstSym->u32StartAddr; i <= pstSym->u32EndAddr; i++)
00745         {
00746             WatchPoint_Delete( i );
00747         }
00748     }
00749     else
00750     {
00751         printf( "Inserting watchpoint @ 0x%04X\n", pstSym->u32StartAddr );
00752         uint32_t i;
00753         for (i = pstSym->u32StartAddr; i <= pstSym->u32EndAddr; i++)
00754         {
00755             WatchPoint_Insert( i );
00756         }
00757     }
00758
00759     return false;
00760 }
00761
00762 //-----
00763 static bool Interactive_ListObj( char *szCommand_ )
00764 {
00765     uint32_t u32Count = Symbol_Get_Obj_Count();
00766     uint32_t i;
00767     printf( "Listing objects:\n" );
00768     for (i = 0; i < u32Count; i++)
00769     {
00770         Debug_Symbol_t *pstSymbol = Symbol_Obj_At_Index(i);
00771         if (!pstSymbol)
00772         {
00773             break;
00774         }
00775
00776         printf( "%d: %s\n", i, pstSymbol->szName );
00777     }
00778     printf( " done\n" );
00779     return false;
00780 }

```



```

00781
00782 //-----
00783 static bool Interactive_ListFunc( char *szCommand_ )
00784 {
00785     uint32_t u32Count = Symbol_Get_Func_Count();
00786     uint32_t i;
00787     printf( "Listing functions:\n" );
00788     for (i = 0; i < u32Count; i++)
00789     {
00790         Debug_Symbol_t *pstSymbol = Symbol_Func_At_Index(i);
00791         if (!pstSymbol)
00792         {
00793             break;
00794         }
00795         printf( "%d: %s\n", i, pstSymbol->szName );
00796     }
00797     printf( " done\n" );
00798     return false;
00799 }
00800 }

```

## 4.75 src/debug/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.*

#### 4.75.1 Detailed Description

Interactive debugging support.

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

Definition in file [interactive.h](#).

#### 4.75.2 Function Documentation



```

00010 *                                     | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00023 #ifndef __INTERACTIVE_H__
00024 #define __INTERACTIVE_H__
00025
00026 #include "emu_config.h"
00027 #include "avr_cpu.h"
00028 #include "trace_buffer.h"
00029
00030 //-----
00037 void Interactive_CheckAndExecute( void );
00038
00039 //-----
00045 void Interactive_Set( void );
00046
00047 //-----
00056 void Interactive_Init( TraceBuffer_t *pstTrace_);
00057
00058 #endif

```

## 4.77 src/debug/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.*

### 4.77.1 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](#).

## 4.77.2 Function Documentation

### 4.77.2.1 TraceBuffer\_Init()

```
void TraceBuffer_Init (
    TraceBuffer_t * pstTraceBuffer_ )
```

TraceBuffer\_Init Initialize a tracebuffer prior to use.

#### Parameters

<i>pstTraceBuffer_</i>	Pointer to the tracebuffer to initialize
------------------------	--

Definition at line 35 of file [trace\\_buffer.c](#).

### 4.77.2.2 TraceBuffer\_LoadElement()

```
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

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

Definition at line 67 of file [trace\\_buffer.c](#).

### 4.77.2.3 TraceBuffer\_Print()

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

TraceBuffer\_Print Print the raw contents of a tracebuffer to standard output.

## Parameters

<i>pstTrace↔ Buffer_</i>	Pointer to the tracebuffer to print
<i>eFormat_</i>	Formatting type for the print

Definition at line 120 of file [trace\\_buffer.c](#).

## 4.77.2.4 TraceBuffer\_PrintElement()

```
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

<i>pst↔ Element_</i>	Pointer to the trace element to print .
<i>eFormat_</i>	Formatting type for the print

Definition at line 75 of file [trace\\_buffer.c](#).

## 4.77.2.5 TraceBuffer\_StoreFromCPU()

```
void TraceBuffer_StoreFromCPU (
    TraceBuffer_t * pstTraceBuffer_ )
```

TraceBuffer\_StoreFromCPU Store a trace element in the tracebuffer at its current head index.

## Parameters

<i>pstTrace↔ Buffer_</i>	Pointer to the tracebuffer to store into
------------------------------	--

Definition at line 41 of file [trace\\_buffer.c](#).

## 4.78 trace\_buffer.c

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

```

00002 *      (      (      (      |
00003 *      \) ) \) )      (      |
00004 *      ((/( ((/(      (      | -- [ Funkenstein ] -----
00005 *      /( ) / ) ((( ( ) \ ) \ / ) | -- [ Little ] -----
00006 *      ( ) _ ( )      \ ) _ \ ( ) ( ) ( ) | -- [ AVR ] -----
00007 *      | _ | | _      ( ) _ \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ / / | _ \ | -- [ Runtime ] -----
00009 *      | _ | | _      / _ \ \ \ / / | _ \ |
00010 *      | _ | | _      / _ \ \ \ / / | _ \ |
00011 *      | _ | | _      / _ \ \ \ / / | _ \ |
00012 *      | _ | | _      / _ \ \ \ / / | _ \ |
00013 *      | _ | | _      / _ \ \ \ / / | _ \ |
00014 *      | _ | | _      / _ \ \ \ / / | _ \ |
00015 *      | _ | | _      / _ \ \ \ / / | _ \ |
00016 *      | _ | | _      / _ \ \ \ / / | _ \ |
00017 *      | _ | | _      / _ \ \ \ / / | _ \ |
00018 *      | _ | | _      / _ \ \ \ / / | _ \ |
00019 *      | _ | | _      / _ \ \ \ / / | _ \ |
00020 *      | _ | | _      / _ \ \ \ / / | _ \ |
00021 *      | _ | | _      / _ \ \ \ / / | _ \ |
00022 *      | _ | | _      / _ \ \ \ / / | _ \ |
00023 *      | _ | | _      / _ \ \ \ / / | _ \ |
00024 *      | _ | | _      / _ \ \ \ / / | _ \ |
00025 *      | _ | | _      / _ \ \ \ / / | _ \ |
00026 *      | _ | | _      / _ \ \ \ / / | _ \ |
00027 *      | _ | | _      / _ \ \ \ / / | _ \ |
00028 *      | _ | | _      / _ \ \ \ / / | _ \ |
00029 *      | _ | | _      / _ \ \ \ / / | _ \ |
00030 *      | _ | | _      / _ \ \ \ / / | _ \ |
00031 *      | _ | | _      / _ \ \ \ / / | _ \ |
00032 *      | _ | | _      / _ \ \ \ / / | _ \ |
00033 *      | _ | | _      / _ \ \ \ / / | _ \ |
00034 *      | _ | | _      / _ \ \ \ / / | _ \ |
00035 *      | _ | | _      / _ \ \ \ / / | _ \ |
00036 *      | _ | | _      / _ \ \ \ / / | _ \ |
00037 *      | _ | | _      / _ \ \ \ / / | _ \ |
00038 *      | _ | | _      / _ \ \ \ / / | _ \ |
00039 *      | _ | | _      / _ \ \ \ / / | _ \ |
00040 *      | _ | | _      / _ \ \ \ / / | _ \ |
00041 *      | _ | | _      / _ \ \ \ / / | _ \ |
00042 *      | _ | | _      / _ \ \ \ / / | _ \ |
00043 *      | _ | | _      / _ \ \ \ / / | _ \ |
00044 *      | _ | | _      / _ \ \ \ / / | _ \ |
00045 *      | _ | | _      / _ \ \ \ / / | _ \ |
00046 *      | _ | | _      / _ \ \ \ / / | _ \ |
00047 *      | _ | | _      / _ \ \ \ / / | _ \ |
00048 *      | _ | | _      / _ \ \ \ / / | _ \ |
00049 *      | _ | | _      / _ \ \ \ / / | _ \ |
00050 *      | _ | | _      / _ \ \ \ / / | _ \ |
00051 *      | _ | | _      / _ \ \ \ / / | _ \ |
00052 *      | _ | | _      / _ \ \ \ / / | _ \ |
00053 *      | _ | | _      / _ \ \ \ / / | _ \ |
00054 *      | _ | | _      / _ \ \ \ / / | _ \ |
00055 *      | _ | | _      / _ \ \ \ / / | _ \ |
00056 *      | _ | | _      / _ \ \ \ / / | _ \ |
00057 *      | _ | | _      / _ \ \ \ / / | _ \ |
00058 *      | _ | | _      / _ \ \ \ / / | _ \ |
00059 *      | _ | | _      / _ \ \ \ / / | _ \ |
00060 *      | _ | | _      / _ \ \ \ / / | _ \ |
00061 *      | _ | | _      / _ \ \ \ / / | _ \ |
00062 *      | _ | | _      / _ \ \ \ / / | _ \ |
00063 *      | _ | | _      / _ \ \ \ / / | _ \ |
00064 *      | _ | | _      / _ \ \ \ / / | _ \ |
00065 *      | _ | | _      / _ \ \ \ / / | _ \ |
00066 *      | _ | | _      / _ \ \ \ / / | _ \ |
00067 *      | _ | | _      / _ \ \ \ / / | _ \ |
00068 *      | _ | | _      / _ \ \ \ / / | _ \ |
00069 *      | _ | | _      / _ \ \ \ / / | _ \ |
00070 *      | _ | | _      / _ \ \ \ / / | _ \ |
00071 *      | _ | | _      / _ \ \ \ / / | _ \ |
00072 *      | _ | | _      / _ \ \ \ / / | _ \ |
00073 *      | _ | | _      / _ \ \ \ / / | _ \ |
00074 *      | _ | | _      / _ \ \ \ / / | _ \ |
00075 *      | _ | | _      / _ \ \ \ / / | _ \ |
00076 *      | _ | | _      / _ \ \ \ / / | _ \ |
00077 *      | _ | | _      / _ \ \ \ / / | _ \ |
00078 *      | _ | | _      / _ \ \ \ / / | _ \ |
00079 *      | _ | | _      / _ \ \ \ / / | _ \ |
00080 *      | _ | | _      / _ \ \ \ / / | _ \ |
00081 *      | _ | | _      / _ \ \ \ / / | _ \ |
00082 *      | _ | | _      / _ \ \ \ / / | _ \ |
00083 *      | _ | | _      / _ \ \ \ / / | _ \ |
00084 *      | _ | | _      / _ \ \ \ / / | _ \ |
00085 *      | _ | | _      / _ \ \ \ / / | _ \ |
00086 *      | _ | | _      / _ \ \ \ / / | _ \ |
00087 *      | _ | | _      / _ \ \ \ / / | _ \ |
00088 *      | _ | | _      / _ \ \ \ / / | _ \ |
00089 *      | _ | | _      / _ \ \ \ / / | _ \ |

```

```

00090
00091     stCPU.u32PC = u16TempPC;
00092 }
00093
00094 if (eFormat_ & TRACE_PRINT_COMPACT)
00095 {
00096     printf( "%04X ", pstElement_>u16SP );
00097
00098     int i;
00099     for (i = 0; i < 32; i++)
00100     {
00101         printf( "%02X ", pstElement_>stCoreRegs.r[i] );
00102     }
00103     printf( "\n" );
00104 }
00105 if (eFormat_ & TRACE_PRINT_REGISTERS)
00106 {
00107     uint8_t i;
00108     for (i = 0; i < 32; i++)
00109     {
00110         printf( "[R%02d] = 0x%02X\n", i, pstElement_>stCoreRegs.r[i] );
00111     }
00112     printf("[SP]   = 0x%04X\n", pstElement_>u16SP );
00113     printf("[PC]   = 0x%04X\n", (uint16_t)pstElement_>u32PC );
00114     printf("[SREG]= 0x%02X", pstElement_>u8SR );
00115     printf( "\n" );
00116 }
00117 }
00118
00119 //-----
00120 void TraceBuffer_Print( TraceBuffer_t *pstTraceBuffer_,
TracePrintFormat_t eFormat_ )
00121 {
00122     int i;
00123     for (i = pstTraceBuffer_>u32Index; i < CONFIG_TRACEBUFFER_SIZE; i++)
00124     {
00125         TraceBuffer_PrintElement(&pstTraceBuffer_>
astTraceStep[i], eFormat_ );
00126     }
00127     for (i = 0; i < pstTraceBuffer_>u32Index; i++)
00128     {
00129         TraceBuffer_PrintElement(&pstTraceBuffer_>
astTraceStep[i], eFormat_ );
00130     }
00131 }

```

## 4.79 src/debug/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.*

### 4.79.1 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](#).

### 4.79.2 Function Documentation

#### 4.79.2.1 TraceBuffer\_Init()

```
void TraceBuffer_Init (
    TraceBuffer\_t * pstTraceBuffer_ )
```

[TraceBuffer\\_Init](#) Initialize a tracebuffer prior to use.

##### Parameters

<a href="#">pstTraceBuffer_</a>	Pointer to the tracebuffer to initialize
---------------------------------	--

Definition at line 35 of file [trace\\_buffer.c](#).

#### 4.79.2.2 TraceBuffer\_LoadElement()

```
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

<i>pstTrace↔ Buffer_</i>	Pointer to a tracebuffer to load from
<i>pstElement_</i>	Pointer to a trace element structure to store data into
<i>u32Element_</i>	Index of the element in the tracebuffer to read

Definition at line 67 of file [trace\\_buffer.c](#).

## 4.79.2.3 TraceBuffer\_Print()

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

TraceBuffer\_Print Print the raw contents of a tracebuffer to standard output.

## Parameters

<i>pstTrace↔ Buffer_</i>	Pointer to the tracebuffer to print
<i>eFormat_</i>	Formatting type for the print

Definition at line 120 of file [trace\\_buffer.c](#).

## 4.79.2.4 TraceBuffer\_PrintElement()

```
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

<i>pst↔ Element_</i>	Pointer to the trace element to print .
<i>eFormat_</i>	Formatting type for the print

Definition at line 75 of file [trace\\_buffer.c](#).

#### 4.79.2.5 TraceBuffer\_StoreFromCPU()

```
void TraceBuffer_StoreFromCPU (
    TraceBuffer_t * pstTraceBuffer_ )
```

TraceBuffer\_StoreFromCPU Store a trace element in the tracebuffer at its current head index.

##### Parameters

<i>pstTraceBuffer_</i>	Pointer to the tracebuffer to store into
------------------------	--

Definition at line 41 of file [trace\\_buffer.c](#).

## 4.80 trace\_buffer.h

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ )  )\ )      (      |
00004 *      ((/( ((/(      \      ( ((/(      -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( ) \      / ( )      -- [ Little ] -----
00006 *      ( ) _ ( )      ) \ _ ) ( ( ( ) ( )      -- [ AVR ] -----
00007 *      | _ | | _      ( ) \ ( ) \ \ / / | _ \      -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ / / | _ \      -- [ Runtime ] -----
00009 *      | _ | | _      / _ \ \ \ / / | _ \      |
00010 *                                          | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00023 #ifndef __TRACE_BUFFER_H__
00024 #define __TRACE_BUFFER_H__
00025
00026 #include <stdint.h>
00027
00028 #include "emu_config.h"
00029 #include "avr_cpu.h"
00030
00031 //-----
00035 typedef struct
00036 {
00037     uint64_t    u64Counter;
00038     uint64_t    u64CycleCount;
00039     uint16_t    u16OpCode;
00040     uint16_t    u32PC;
00041     uint16_t    u16SP;
00042     uint8_t     u8SR;
00043
00044     AVR_CoreRegisters stCoreRegs;
00045 } TraceElement_t;
00046
00047 //-----
00053 typedef struct
00054 {
00055     TraceElement_t astTraceStep[ CONFIG_TRACEBUFFER_SIZE ];
00056     uint32_t        u32Index;
00057 } TraceBuffer_t;
00058
00059 //-----
00064 typedef enum
00065 {
00066     TRACE_PRINT_COMPACT      = 1,
00067     TRACE_PRINT_REGISTERS   = 2,
00068     TRACE_PRINT_DISASSEMBLY = 4
00069 } TracePrintFormat_t;
00070
00071 //-----
00077 void TraceBuffer_Init( TraceBuffer_t *pstTraceBuffer_ );
00078
00079 //-----
00087 void TraceBuffer_StoreFromCPU( TraceBuffer_t *pstTraceBuffer_ );
00088
```

```

00089 //-----
00100 void TraceBuffer_LoadElement( TraceBuffer_t *pstTraceBuffer_,
    TraceElement_t *pstElement_, uint32_t u32Element_ );
00101
00102 //-----
00109 void TraceBuffer_PrintElement( TraceElement_t *pstElement_,
    TracePrintFormat_t eFormat_ );
00110
00111 //-----
00120 void TraceBuffer_Print( TraceBuffer_t *pstTraceBuffer_,
    TracePrintFormat_t eFormat_ );
00121
00122 #endif

```

## 4.81 src/debug/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.*

#### 4.81.1 Detailed Description

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

Definition in file [watchpoint.c](#).

#### 4.81.2 Function Documentation

##### 4.81.2.1 WatchPoint\_Delete()

```

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**

<i>u16↔</i> <i>Addr_</i>	Address to remove data watchpoints from (if any)
-----------------------------	--

Definition at line 57 of file [watchpoint.c](#).

**4.81.2.2 WatchPoint\_EnabledAtAddress()**

```
bool WatchPoint_EnabledAtAddress (
    uint16_t u16Addr_ )
```

WatchPoint\_EnabledAtAddress.

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

**Parameters**

<i>u16↔</i> <i>Addr_</i>	Address to check
-----------------------------	------------------

**Returns**

true if watchpoint is installed at the specified address

Definition at line 97 of file [watchpoint.c](#).

**4.81.2.3 WatchPoint\_Insert()**

```
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**

<i>u16↔</i> <i>Addr_</i>	Address of the watchpoint.
-----------------------------	----------------------------

Definition at line 31 of file [watchpoint.c](#).

## 4.82 watchpoint.c

```

00001
00002 /*****
00003 *      (      (      (      |
00004 *      )\ )  )\ )      (      |
00005 *      ((/( ((/(      )\      ( ((/(      | -- [ Funkenstein ] -----
00006 *      /( )  /( )  ((( ( ) \  )\  /( )      | -- [ Little ] -----
00007 *      ( ) _ ( )      )\ _ )\  ( ( ) ( ( )      | -- [ AVR ] -----
00008 *      | _ | _ |      ( ) _ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00009 *      | _ | | _ | / _ \ \ \ v / | _ /      | -- [ Runtime ] -----
00010 *      | _ | | _ | / _ \ \ \ \ / | _ \      |
00011 *      | _ | | _ | / _ \ \ \ \ / | _ \      | "Yeah, it does Arduino..."
00012 * -----+-----
00013 * (c) Copyright 2014-17, 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 ul6Addr_ )
00032 {
00033     // Don't add multiple watchpoints at the same address
00034     if (WatchPoint_EnabledAtAddress( ul6Addr_ ))
00035     {
00036         return;
00037     }
00038
00039     WatchPoint_t *pstNewWatch = NULL;
00040
00041     pstNewWatch = (WatchPoint_t*)malloc( sizeof(WatchPoint_t) );
00042
00043     pstNewWatch->next = stCPU.pstWatchPoints;
00044     pstNewWatch->prev = NULL;
00045
00046     pstNewWatch->ul6Addr = ul6Addr_;
00047
00048     if (stCPU.pstWatchPoints)
00049     {
00050         WatchPoint_t *pstTemp = stCPU.pstWatchPoints;
00051         pstTemp->prev = pstNewWatch;
00052     }
00053     stCPU.pstWatchPoints = pstNewWatch;
00054 }
00055
00056 //-----
00057 void WatchPoint_Delete( uint16_t ul6Addr_ )
00058 {
00059     WatchPoint_t *pstTemp = stCPU.pstWatchPoints;
00060
00061     while (pstTemp)
00062     {
00063         if (pstTemp->ul6Addr == ul6Addr_)
00064         {
00065             // Remove node -- reconnect surrounding elements
00066             WatchPoint_t *pstNext = pstTemp->next;
00067             if (pstNext)
00068             {
00069                 pstNext->prev = pstTemp->prev;
00070             }
00071
00072             WatchPoint_t *pstPrev = pstTemp->prev;
00073             if (pstPrev)
00074             {
00075                 pstPrev->next = pstTemp->next;
00076             }
00077
00078             // Adjust list-head if necessary
00079             if (pstTemp == stCPU.pstWatchPoints)
00080             {
00081                 stCPU.pstWatchPoints = pstNext;
00082             }
00083
00084             // Free the node/iterate to next node.
00085             pstPrev = pstTemp;
00086             pstTemp = pstTemp->next;
00087             free(pstPrev);
00088         }
00089         else
00090         {
00091             pstTemp = pstTemp->next;

```

```

00092     }
00093 }
00094 }
00095
00096 //-----
00097 bool WatchPoint_EnabledAtAddress( uint16_t u16Addr_ )
00098 {
00099     WatchPoint_t *pstTemp = stCPU.pstWatchPoints;
00100
00101     while (pstTemp)
00102     {
00103         if (pstTemp->u16Addr == u16Addr_)
00104         {
00105             return true;
00106         }
00107         pstTemp = pstTemp->next;
00108     }
00109     return false;
00110 }

```

## 4.83 src/debug/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.*

#### 4.83.1 Detailed Description

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

Definition in file [watchpoint.h](#).

## 4.83.2 Function Documentation

### 4.83.2.1 WatchPoint\_Delete()

```
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

<i>u16Addr_</i>	Address to remove data watchpoints from (if any)
-----------------	--

Definition at line 57 of file [watchpoint.c](#).

### 4.83.2.2 WatchPoint\_EnabledAtAddress()

```
bool WatchPoint_EnabledAtAddress (
    uint16_t u16Addr_ )
```

WatchPoint\_EnabledAtAddress.

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

#### Parameters

<i>u16Addr_</i>	Address to check
-----------------	------------------

#### Returns

true if watchpoint is installed at the specified address

Definition at line 97 of file [watchpoint.c](#).

### 4.83.2.3 WatchPoint\_Insert()

```
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

<code>u16Addr_</code>	Address of the watchpoint.
-----------------------	----------------------------

Definition at line 31 of file [watchpoint.c](#).

## 4.84 watchpoint.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( ()/(      (      | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( ) \ ) / ( )      | -- [ Little ] -----
00006 *      ( ) _ ( )      )\ _ )\ ( ( ( ) _      | -- [ AVR ] -----
00007 *      | _ | _ |      ( ) _ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ V / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _ _ | / _ \ \ \ \ / | _ \      |
00010 *      |      |      | "Yeah, it does Arduino..."
00011 *      -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00022 #ifndef __WATCHPOINT_H__
00023 #define __WATCHPOINT_H__
00024
00025 #include <stdint.h>
00026 #include <stdbool.h>
00027
00028 #include "avr_cpu.h"
00029
00030 //-----
00031 typedef struct _WatchPoint
00032 {
00033     struct _WatchPoint *next;
00034     struct _WatchPoint *prev;
00035
00036     uint16_t u16Addr;
00037 } WatchPoint_t;
00038
00039 //-----
00048 void WatchPoint_Insert( uint16_t u16Addr_ );
00049
00050 //-----
00059 void WatchPoint_Delete( uint16_t u16Addr_ );
00060
00061 //-----
00070 bool WatchPoint_EnabledAtAddress( uint16_t u16Addr_ );
00071
00072 #endif
00073

```

## 4.85 src/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**

### 4.85.1 Detailed Description

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

Definition in file [flavr.c](#).

## 4.86 flavr.c

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( ((/(      )\      | -- [ Funkenstein ] -----
00005 *      /( ) )/( )((( ( )\ )\ /( )      | -- [ Little ] -----
00006 *      ( )_ ( )      )\ _ )\ ( ( ( )      | -- [ AVR ] -----
00007 *      | | _ | |      ( )\ ( )\ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ / / | _ /      | -- [ Runtime ] -----
00009 *      |_ | | _ | / / \ \ \ / | _ \      |
00010 *      | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00022 #include <stdio.h>
00023 #include <stdlib.h>
00024 #include <string.h>
00025 #include <stdint.h>
00026
00027 #include "emu_config.h"
00028 #include "variant.h"
00029
00030 //-----
00031 #include "avr_coreregs.h"
00032 #include "avr_periphregs.h"
00033 #include "avr_op_cycles.h"
00034 #include "avr_op_decode.h"
00035 #include "avr_op_size.h"
00036 #include "avr_cpu_print.h"
00037 #include "avr_cpu.h"
00038 #include "avr_loader.h"
00039
00040 //-----
00041 #include "mega_uart.h"
00042 #include "mega_eint.h"
00043 #include "mega_timer16.h"
00044 #include "mega_timer8.h"
00045 #include "mega_eeprom.h"
00046
00047 //-----
00048 #include "avr_disasm.h"
00049 #include "trace_buffer.h"
00050 #include "options.h"
00051 #include "interactive.h"
00052 #include "breakpoint.h"
00053 #include "watchpoint.h"
00054 #include "kernel_aware.h"
00055 #include "code_profile.h"
00056 #include "tlv_file.h"
00057 #include "gdb_rsp.h"
00058
00059 //-----
00060 typedef enum
00061 {
00062     EEPROM_TOO_BIG,
00063     RAM_TOO_BIG,
00064     RAM_TOO_SMALL,
00065     ROM_TOO_BIG,
00066     INVALID_HEX_FILE,
00067     INVALID_VARIANT,
00068     INVALID_DEBUG_OPTIONS
00069 } ErrorReason_t;
00070
00071 //-----
00072 static TraceBuffer_t stTraceBuffer;
00073
00074 //-----
00075 void splash(void)
00076 {
00077     printf(
00078         " * -----+-----\n"
00079         " *      (      (      (      | \n"
00080         " *      )\ ) )\ )      (      | \n"
00081         " *      ((/( ((/(      )\      | -- [ Funkenstein ] -----\n"
00082         " *      /( ) )/( )((( ( )\ )\ /( )      | -- [ Little ] -----\n"
00083         " *      ( )_ ( )      )\ _ )\ ( ( ( )      | -- [ AVR ] -----\n"
00084         " *      | | _ | |      ( )\ ( )\ \ / / | _ \      | -- [ Virtual ] -----\n"
00085         " *      | _ | | _      / _ \ \ \ / / | _ /      | -- [ Runtime ] -----\n"
00086         " *      |_ | | _ | / / \ \ \ / | _ \      | \n"
00087         " *      | "From the makers of Mark3!" \n"
00088         " * -----+-----\n"
00089         " * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved\n"
00090         " * See license.txt for details\n"
00091     );

```

```

00092 }
00093
00094 //-----
00095 void error_out( ErrorReason_t eReason_ )
00096 {
00097     switch (eReason_)
00098     {
00099         case EEPROM_TOO_BIG:
00100             printf( "EEPROM Size specified is too large\n" );
00101             break;
00102         case RAM_TOO_BIG:
00103             printf( "RAM Size specified is too large\n" );
00104             break;
00105         case RAM_TOO_SMALL:
00106             printf( "RAM Size specified is too small\n" );
00107             break;
00108         case ROM_TOO_BIG:
00109             printf( "ROM Size specified is too large\n" );
00110             break;
00111         case INVALID_HEX_FILE:
00112             printf( "HEX Programming file cannot be loaded\n");
00113             break;
00114         case INVALID_VARIANT:
00115             printf( "Unknown variant not supported\n");
00116             break;
00117         case INVALID_DEBUG_OPTIONS:
00118             printf( "GDB and built-in interactive debugger are mutually exclusive\n");
00119         default:
00120             printf( "Some other reason\n" );
00121     }
00122
00123     Options_PrintUsage();
00124
00125     exit (-1);
00126 }
00127
00128 //-----
00129 void emulator_loop(void)
00130 {
00131     bool bUseTrace = false;
00132     bool bProfile = false;
00133     bool bUseGDB = false;
00134
00135     if ( Options_GetByName("--trace") && Options_GetByName("--debug") )
00136     {
00137         bUseTrace = true;
00138     }
00139
00140     if ( Options_GetByName("--profile") )
00141     {
00142         bProfile = true;
00143     }
00144
00145     if ( Options_GetByName("--gdb") )
00146     {
00147         bUseGDB = true;
00148     }
00149
00150     while (1)
00151     {
00152         // Check to see if we've hit a breakpoint
00153         if (BreakPoint_EnabledAtAddress(stCPU.u32PC))
00154         {
00155             if (bUseGDB)
00156             {
00157                 GDB_Set();
00158             }
00159             else
00160             {
00161                 Interactive_Set();
00162             }
00163         }
00164
00165         // Check to see if we're in interactive debug mode, and thus need to wait for input
00166         if (bUseGDB)
00167         {
00168             GDB_CheckAndExecute();
00169         }
00170         else
00171         {
00172             Interactive_CheckAndExecute();
00173         }
00174
00175         // Store the current CPU state into the tracebuffer
00176         if (bUseTrace)
00177         {
00178             TraceBuffer_StoreFromCPU(&stTraceBuffer);

```

```

00179     }
00180
00181     // Run code profiling logic
00182     if (bProfile)
00183     {
00184         Profile_Hit(stCPU.u32PC);
00185     }
00186
00187     // Execute a machine cycle
00188     CPU_RunCycle();
00189 }
00190 // doesn't return, except by quitting from debugger, or by signal.
00191 }
00192
00193 //-----
00194 void add_plugins(void)
00195 {
00196     CPU_AddPeriph(&stUART);
00197     CPU_AddPeriph(&stEINT_a);
00198     CPU_AddPeriph(&stEINT_b);
00199     CPU_AddPeriph(&stTimer16);
00200     CPU_AddPeriph(&stTimer16a);
00201     CPU_AddPeriph(&stTimer16b);
00202     CPU_AddPeriph(&stTimer8);
00203     CPU_AddPeriph(&stTimer8a);
00204     CPU_AddPeriph(&stTimer8b);
00205     CPU_AddPeriph(&stEEPROM);
00206 }
00207
00208 //-----
00209 void flavr_disasm(void)
00210 {
00211     uint32_t u32Size;
00212
00213     u32Size = stCPU.u32ROMSize / sizeof(uint16_t);
00214     stCPU.u32PC = 0;
00215
00216     while (stCPU.u32PC < u32Size)
00217     {
00218         uint16_t OP = stCPU.pu16ROM[stCPU.u32PC];
00219         char szBuf[256];
00220
00221         printf("0x%04X: [0x%04X] ", stCPU.u32PC, OP);
00222         AVR_Decode(OP);
00223         AVR_Disasm_Function(OP) (szBuf);
00224         printf(" %s", szBuf );
00225         stCPU.u32PC += AVR_Opcode_Size(OP);
00226     }
00227     exit(0);
00228 }
00229
00230 //-----
00231 void emulator_init(void)
00232 {
00233     AVR_CPU_Config_t stConfig;
00234
00235     // -- Initialize the emulator based on command-line args
00236     const AVR_Variant_t *pstVariant;
00237
00238     pstVariant = Variant_GetByName( Options_GetByName("--variant") );
00239     if (!pstVariant)
00240     {
00241         error_out( INVALID_VARIANT );
00242     }
00243
00244     if (Options_GetByName("--exitreset"))
00245     {
00246         stConfig.bExitOnReset = true;
00247     }
00248     else
00249     {
00250         stConfig.bExitOnReset = false;
00251     }
00252
00253     stConfig.u32EESize = pstVariant->u32EESize;
00254     stConfig.u32RAMSize = pstVariant->u32RAMSize;
00255     stConfig.u32ROMSize = pstVariant->u32ROMSize;
00256     stConfig.pstFeatureMap = pstVariant->pstFeatures;
00257     stConfig.pstVectorMap = pstVariant->pstVectors;
00258
00259     if (stConfig.u32EESize >= 32768)
00260     {
00261         error_out( EEPROM_TOO_BIG );
00262     }
00263
00264     if (stConfig.u32RAMSize >= 65535)
00265     {

```

```

00266     error_out( RAM_TOO_BIG );
00267 }
00268 else if (stConfig.u32RAMSize < 256)
00269 {
00270     error_out( RAM_TOO_SMALL );
00271 }
00272
00273 if (stConfig.u32ROMSize >= (256*1024))
00274 {
00275     error_out( ROM_TOO_BIG );
00276 }
00277
00278 CPU_Init(&stConfig);
00279
00280 TraceBuffer_Init( &stTraceBuffer );
00281
00282 if (Options_GetByName("--hexfile"))
00283 {
00284     if (!AVR_Load_HEX( Options_GetByName("--hexfile") ))
00285     {
00286         error_out( INVALID_HEX_FILE );
00287     }
00288 }
00289 else if (Options_GetByName("--elffile"))
00290 {
00291     if (!AVR_Load_ELF( Options_GetByName("--elffile") ))
00292     {
00293         error_out( INVALID_HEX_FILE );
00294     }
00295 }
00296 else
00297 {
00298     error_out( INVALID_HEX_FILE );
00299 }
00300
00301 if (Options_GetByName("--disasm"))
00302 {
00303     // terminates after disassembly is complete
00304     flavr_disasm();
00305 }
00306
00307 if (Options_GetByName("--debug"))
00308 {
00309     Interactive_Init( &stTraceBuffer );
00310 }
00311 if (Options_GetByName("--gdb"))
00312 {
00313     GDB_Init();
00314 }
00315
00316 // Only insert a breakpoint/enter interactive debugging mode if specified.
00317 // Otherwise, start with the emulator running.
00318 if (Options_GetByName("--debug") && Options_GetByName("--gdb"))
00319 {
00320     error_out( INVALID_DEBUG_OPTIONS );
00321 }
00322 if (Options_GetByName("--debug"))
00323 {
00324     BreakPoint_Insert( 0 );
00325 }
00326
00327 add_plugins();
00328
00329 if (Options_GetByName("--mark3") || Options_GetByName("--profile"))
00330 {
00331     // Initialize tag-length-value code if we're running with code
00332     // profiling or kernel-aware debugging, since they generate a
00333     // lot of data that's better stored in a binary format for
00334     // efficiency.
00335     TLV_WriteInit( "flavr.tlv" );
00336 }
00337
00338 if (Options_GetByName("--mark3"))
00339 {
00340     // Mark3 kernel-aware mode should only be enabled on-demand
00341     KernelAware_Init();
00342 }
00343
00344 if (Options_GetByName("--profile"))
00345 {
00346     Profile_Init( stConfig.u32ROMSize );
00347     atexit( Profile_Print );
00348 }
00349 }
00350
00351 //-----
00352 int main( int argc, char **argv )

```

```

00353 {
00354
00355     // Initialize all emulator data
00356     Options_Init(argc, argv);
00357
00358     if (!Options_GetByName("--silent"))
00359     {
00360         splash();
00361     }
00362
00363     emulator_init();
00364
00365     // Run the emulator/debugger loop.
00366     emulator_loop();
00367
00368     return 0;
00369
00370 }

```

## 4.87 src/kernel\_aware/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

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ )  )\ )      (      )\ )      |
00004 *      ( )/( ( )/(      )\      (      ( )/(      | -- [ Funkenstein ] -----
00005 *      /( ) )/( ) ((( ( )\      )\      /( )      | -- [ Little ] -----
00006 *      ( )_ ( )      )\ _ )\      ( ) ( ) ( )      | -- [ AVR ] -----
00007 *      | _ | |      ( )_ \ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \      \ v / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _ _ | / \ \ \      \ /      | _ | _ |      |
00010 *      |      |      |      |      |      |      |      |
00011 *      +-----+-----+-----+-----+-----+-----+
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #include <stdio.h>
00022 #include <string.h>
00023 #include <stdlib.h>
00024
00025 #include <stdint.h>
00026 #include <SDL/SDL.h>
00027
00028 #include "kernel_aware.h"
00029 #include "debug_sym.h"
00030 #include "write_callout.h"
00031 #include "interrupt_callout.h"
00032
00033 //-----
00034 #define GFX_RES_X      (128)
00035 #define GFX_RES_Y      (160)
00036 #define GFX_SCALE      (3)
00037
00038 //-----
00039 typedef struct
00040 {
00041     uint16_t usX;
00042     uint16_t usY;
00043     uint32_t uColor;
00044 } DrawPoint_t;
00045
00046 //-----
00047 static SDL_Surface *pstScreen = 0;
00048
00049 //-----
00050 void KA_Graphics_Close(void)
00051 {
00052     if (pstScreen)
00053     {
00054         SDL_FreeSurface(pstScreen);
00055     }
00056     SDL_Quit();
00057 }
00058
00059 //-----
00060 void KA_Graphics_ClearScreen(void)
00061 {
00062     memset( pstScreen->pixels, 0, sizeof(uint16_t) * (GFX_RES_X*GFX_SCALE) * (GFX_RES_Y*GFX_SCALE) );
00063 }
00064
00065 //-----
00066 void KA_Graphics_DrawPoint(DrawPoint_t *pstPoint_)
00067 {
00068     uint32_t *pixels = (uint32_t*)pstScreen->pixels;
00069
00070     // printf( "X:%d Y:%d C=%08X\n", pstPoint_->usX, pstPoint_->usY, pstPoint_->uColor );
00071     if ((pstPoint_->usX < GFX_RES_X) && (pstPoint_->usY < GFX_RES_Y))
00072     {
00073         int i,j;
00074         for (i = 0; i < GFX_SCALE; i++)
00075         {
00076             for (j = 0; j < GFX_SCALE; j++)
00077             {

```

```

00078             pixels[ ((uint32_t)((pstPoint_>usY*GFX_SCALE)+i) * (GFX_RES_X*GFX_SCALE) ) +
00079                     (uint32_t)((pstPoint_>usX*GFX_SCALE)+j) ] = (uint32_t)pstPoint_>
00080             uColor;
00081         }
00082     }
00083 }
00084
00085 //-----
00086 void KA_Graphics_Flip(void)
00087 {
00088     if (pstScreen)
00089     {
00090         SDL_Flip(pstScreen);
00091     }
00092 }
00093
00094 //-----
00095 bool KA_Graphics_Command( uint16_t ul6Addr_, uint8_t u8Data_ )
00096 {
00097     Debug_Symbol_t *pstSymbol = Symbol_Find_Obj_By_Name( "g_pclPoint"
00098 );
00099     switch( u8Data_ )
00100     {
00101         case 1:
00102             if (pstSymbol)
00103             {
00104                 uint16_t ul6PointAddr = *(uint16_t*)&stCPU.pstRAM->au8RAM[ pstSymbol->
00105                 u32StartAddr ];
00106                 DrawPoint_t *pstPoint = (DrawPoint_t*)&stCPU.pstRAM->au8RAM[
00107                 ul6PointAddr ];
00108                 KA_Graphics_DrawPoint( pstPoint );
00109             }
00110             break;
00111         case 2:
00112             KA_Graphics_Flip();
00113             break;
00114         case 0:
00115             default:
00116                 break;
00117     }
00118     return true;
00119 }
00120 //-----
00121 void KA_Graphics_Init(void)
00122 {
00123     Debug_Symbol_t *pstSymbol = 0;
00124     pstSymbol = Symbol_Find_Obj_By_Name( "g_u8GfxCommand" );
00125
00126     // Use pstSymbol's address to get a pointer to the current thread.
00127     if (!pstSymbol)
00128     {
00129         fprintf(stderr, "Kernel-aware graphics driver not found\n" );
00130         return;
00131     }
00132
00133     // Ensure that we actually have the information we need at a valid address
00134     uint16_t ul6CurrPtr = (uint16_t)(pstSymbol->u32StartAddr & 0x0000FFFF);
00135     if (!ul6CurrPtr)
00136     {
00137         fprintf(stderr, "Invalid address for graphics driver global\n" );
00138         return;
00139     }
00140
00141     // Add a callback so that when g_pstCurrent changes, we can update our
00142     // locally-tracked statistics.
00143     WriteCallout_Add( KA_Graphics_Command, ul6CurrPtr );
00144
00145     SDL_Init( SDL_INIT_EVERYTHING );
00146     pstScreen = SDL_SetVideoMode( GFX_RES_X * GFX_SCALE, GFX_RES_Y * GFX_SCALE, 32, SDL_SWSURFACE);
00147     fprintf(stderr, "Kernel-Aware Graphics Installed\n");
00148
00149     atexit( KA_Graphics_Close );
00150 }
00151 }

```

## 4.89 src/kernel\_aware/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

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/ ( ((/ (      )\ ) | -- [ Funkenstein ] -----
00005 *      /( ) /( ) ((( ( )\ ) \ /( ) | -- [ Little ] -----
00006 *      ( ) _ ( )      )\ _ )\ ( ( ( ) _ | -- [ AVR ] -----
00007 *      | | _ | |      ( ) _ ( ) \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ V / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _      / _ \ \ \ V / | _ / |
00010 *      | "Yeah, it does Arduino..."
00011 *      -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __KA_TRACE__
00022 #define __KA_TRACE__
00023
00024 #include "kernel_aware.h"
00025 //-----
00026 //void KA_Graphics_Init( void );
00027
00028 #endif
00029
```

## 4.91 src/kernel\_aware/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 KA\_Interrupt\_Init()

```
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

```
00001 /*****
00002 *      ( )      ( )      ( )      |
00003 *      )\ )      )\ )      ( )      |
00004 *      ((/( ((/( ((/( ((/( ((/( | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( ) ) \ / ( ) | -- [ Little ] -----
00006 *      ( ) _ ( ) ) \ _ ) \ ( ) ( ) ( ) | -- [ AVR ] -----
00007 *      | | _ | | ( ) _ ( ) \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ / _ \ \ \ / / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ / _ \ \ \ / / | _ \ |
00010 *      | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, 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 <time.h>
00026
00027 #include "avr_cpu.h"
00028 #include "kernel_aware.h"
00029 #include "ka_interrupt.h"
00030 #include "write_callout.h"
```

```

00031 #include "interrupt_callout.h"
00032 #include "tlv_file.h"
00033
00034 //-----
00035 static TLV_t *pstTLV = NULL;
00036
00037 //-----
00038 typedef struct
00039 {
00040     uint64_t    u64TimeStamp;
00041     uint8_t     u8Vector;
00042     bool        bEntry;
00043 } Mark3Interrupt_TLV_t;
00044
00045 //-----
00046 static void KA_Interrupt( bool bEntry_, uint8_t u8Vector_ )
00047 {
00048     Mark3Interrupt_TLV_t stData;
00049     stData.u64TimeStamp = stCPU.u64CycleCount;
00050     stData.u8Vector = u8Vector_;
00051     stData.bEntry = bEntry_;
00052     memcpy( &(pstTLV->au8Data[0]), &stData, sizeof(stData) );
00053     TLV_Write(pstTLV);
00054 }
00055
00056 //-----
00057 void KA_Interrupt_Init(void)
00058 {
00059     pstTLV = TLV_Alloc( sizeof(Mark3Interrupt_TLV_t) );
00060     if (!pstTLV)
00061     {
00062         return;
00063     }
00064     pstTLV->eTag = TAG_KERNEL_AWARE_INTERRUPT;
00065     pstTLV->u16Len = sizeof(Mark3Interrupt_TLV_t);
00066     InterruptCallout_Add( KA_Interrupt );
00067 }

```

## 4.93 src/kernel\_aware/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



## 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

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ )  )\ )  (      )\ )  |
00004 *      ()/( ()/(      \      ( ( ()/(      | -- [ Funkenstein ] -----
00005 *      /( )  /( )  ((( )\ ) \ /( )  | -- [ Little ] -----
00006 *      ( )|( )      )\ _ \ ( ) ( ( )  | -- [ AVR ] -----
00007 *      | | _ | |      ( ) \ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ / / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _ _ | / \ \ \ \ / \ | _ | \      |
00010 *      |      |      |      |      |      |      | "Yeah, it does Arduino..."
00011 *      |      |      |      |      |      |      |
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 *      See license.txt for details
00014 *****/
00021 #include <stdio.h>
00022 #include <stdint.h>
00023 #include <stdlib.h>
00024 #include <string.h>
00025 #include <stdbool.h>
00026
00027 #include <SDL/SDL.h>
00028
00029 #include "ka_joystick.h"
00030 #include "write_callout.h"
00031 #include "debug_sym.h"
00032 #include "avr_cpu.h"
00033
00034 //-----
00035 #define FLAVR_JOY_UP          0x01
00036 #define FLAVR_JOY_DOWN       0x02
00037 #define FLAVR_JOY_LEFT       0x04
00038 #define FLAVR_JOY_RIGHT      0x08
00039 #define FLAVR_JOY_FIRE       0x10
00040
00041 //-----
00042 static uint8_t u8Val = 0;
00043
00044 //-----
00045 static bool KA_Scan_Joystick( uint16_t u16Addr_, uint8_t u8Data_ )
00046 {
00047     Debug_Symbol_t *pstSymbol = 0;
00048     pstSymbol = Symbol_Find_Obj_By_Name( "g_u8FlavrJoy" );
00049
00050     if (!pstSymbol)
00051     {
00052         fprintf(stderr, "Invalid joystick scan register\n");
00053         return true;
00054     }
00055
00056     uint16_t u16Addr = (uint16_t) (pstSymbol->u32StartAddr & 0x0000FFFF);
00057
00058     SDL_Event stEvent;
00059

```

```

00060     while (SDL_PollEvent(&stEvent))
00061     {
00062         switch (stEvent.type)
00063         {
00064             case SDL_KEYDOWN:
00065             {
00066                 switch( stEvent.key.keysym.sym )
00067                 {
00068                     case SDLK_UP:
00069                         u8Val |= FLAVR_JOY_UP;
00070                         break;
00071                     case SDLK_DOWN:
00072                         u8Val |= FLAVR_JOY_DOWN;
00073                         break;
00074                     case SDLK_LEFT:
00075                         u8Val |= FLAVR_JOY_LEFT;
00076                         break;
00077                     case SDLK_RIGHT:
00078                         u8Val |= FLAVR_JOY_RIGHT;
00079                         break;
00080                     case SDLK_a:
00081                         u8Val |= FLAVR_JOY_FIRE;
00082                         break;
00083                     case SDLK_ESCAPE:
00084                         exit(0);
00085                         break;
00086                     default:
00087                         break;
00088                 }
00089             }
00090             break;
00091             case SDL_KEYUP:
00092             {
00093                 switch( stEvent.key.keysym.sym )
00094                 {
00095                     case SDLK_UP:
00096                         u8Val &= ~FLAVR_JOY_UP;
00097                         break;
00098                     case SDLK_DOWN:
00099                         u8Val &= ~FLAVR_JOY_DOWN;
00100                         break;
00101                     case SDLK_LEFT:
00102                         u8Val &= ~FLAVR_JOY_LEFT;
00103                         break;
00104                     case SDLK_RIGHT:
00105                         u8Val &= ~FLAVR_JOY_RIGHT;
00106                         break;
00107                     case SDLK_a:
00108                         u8Val &= ~FLAVR_JOY_FIRE;
00109                         break;
00110                     default:
00111                         break;
00112                 }
00113             }
00114             break;
00115             default:
00116                 break;
00117         }
00118     }
00119
00120     stCPU.pstRAM->au8RAM[ u16Addr ] = u8Val;
00121
00122     return true;
00123 }
00124
00125 //-----
00126 void KA_Joystick_Init( void )
00127 {
00128     Debug_Symbol_t *pstSymbol = 0;
00129     pstSymbol = Symbol_Find_Obj_By_Name( "g_u8FlavrJoyUp" );
00130
00131     if (!pstSymbol)
00132     {
00133         fprintf(stderr, "Kernel-aware joystick driver not found\n" );
00134         return;
00135     }
00136
00137     // Ensure that we actually have the information we need at a valid address
00138     uint16_t u16CurrPtr = (uint16_t)(pstSymbol->u32StartAddr & 0x0000FFFF);
00139     if (!u16CurrPtr)
00140     {
00141         fprintf(stderr, "Invalid address for joystick driver global\n" );
00142         return;
00143     }
00144
00145     // Add a callback so that when a joystick scan is requested, we parse keyboard input
00146     WriteCallout_Add( KA_Scan_Joystick, u16CurrPtr );

```

```
00147
00148 }
```

## 4.97 src/kernel\_aware/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

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( (/(      )\      | -- [ Funkenstein ] -----
00005 *      /( ) )/( ) ((( ( ) \      | -- [ Little ] -----
00006 *      ( ) _ ( )      )\ _ )\ ( ( ( ) _      | -- [ AVR ] -----
00007 *      | | _ | |      ( ) _ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ v / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _      / _ \ \ \ \ / | _ \      |
00010 *      *      | "Yeah, it does Arduino..."
00011 *      * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __KA_JOYSTICK_H__
00022 #define __KA_JOYSTICK_H__
00023
00024 #include "kernel_aware.h"
00031 //void KA_Joystick_Init( void );
00032
00033 #endif // __KA_JOYSTICK_H__
```

## 4.99 src/kernel\_aware/ka\_profile.c File Reference

Mark3 RTOS Kernel-Aware Profiling.

```
#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\_Profile\_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 Profiling.

Definition in file [ka\\_profile.c](#).

### 4.99.2 Function Documentation

#### 4.99.2.1 KA\_Profile\_Init()

```
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 u64ProfileEpochStart

```
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

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/ ( ((/ (      )\ ) | -- [ Funkenstein ] -----
00005 *      /( ) ) /( ) )((( ( )\ )\ / ( ) | -- [ Little ] -----
00006 *      ( ) _ | ( ) )      )\ _ )\ ( ) ( ( ) ( ) | -- [ AVR ] -----
00007 *      | _ | | | ( ) _ ( ) \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | | _ / _ \ \ V / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ | / _ \ \ \ / | _ \ |
00010 *      | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #include "kernel_aware.h"
00022 #include "debug_sym.h"
00023 #include "write_callout.h"
00024 #include "interrupt_callout.h"
00025 #include "ka_profile.h"
00026 #include "tlv_file.h"
00027
00028 #include <stdint.h>
00029 #include <stdio.h>
00030 #include <stdlib.h>
00031 #include <string.h>
00032 #include <time.h>
00033
00034 //-----
00037 static uint64_t u64ProfileEpochStart = 0;
00038 static uint64_t u64ProfileTotal = 0;
00039 static uint64_t u64ProfileCount = 0;
00040 static char szNameBuffer[32] = {};
00041 static TLV_t *pstTLV = NULL;
00042
00043 //-----
00044 typedef struct
00045 {
00046     uint64_t u64Timestamp;
00047     uint64_t u64ProfileCount;
00048     uint64_t u64ProfileTotalCycles;
00049     char      szName[32];
00050 } Mark3Profile_TLV_t;
00051
00052 //-----
00053 static void KA_PrintProfileResults(void)
00054 {
00055     Mark3Profile_TLV_t stTLV;
00056
00057     stTLV.u64ProfileCount      = u64ProfileCount;
00058     stTLV.u64ProfileTotalCycles = u64ProfileTotal;
00059     stTLV.u64Timestamp         = stCPU.u64CycleCount;
00060
00061     strcpy( stTLV.szName, szNameBuffer );
00062     memcpy( pstTLV->au8Data, &stTLV, sizeof(Mark3Profile_TLV_t) );
00063
00064     printf( "%s: %llu, %llu, %llu\n", stTLV.szName, stTLV.u64Timestamp, stTLV.
00065             u64ProfileCount, stTLV.u64ProfileTotalCycles );
00066 }
```

```

00066     TLV_Write( pstTLV );
00067 }
00068
00069 //-----
00070 void KA_Command_Profile_Begin(void)
00071 {
00072     u64ProfileCount = 0;
00073     u64ProfileTotal = 0;
00074     u64ProfileEpochStart = 0;
00075
00076     Debug_Symbol_t *pstSymbol = Symbol_Find_Obj_By_Name( "g_stKADData"
00077 );
00078     if (!pstSymbol)
00079     {
00080         return;
00081     }
00082     uint16_t u16NamePtr = *((uint16_t*)&stCPU.pstRAM->au8RAM[ pstSymbol->
u32StartAddr ]);
00083     const char *szName = (const char*)&stCPU.pstRAM->au8RAM[ u16NamePtr ];
00084     if (szName)
00085     {
00086         strcpy( szNameBuffer, szName );
00087     }
00088     else
00089     {
00090         strcpy( szNameBuffer, "(NONE)" );
00091     }
00092 }
00093 }
00094
00095 //-----
00096 void KA_Command_Profile_Start(void)
00097 {
00098     // Profile stop or reset
00099     u64ProfileEpochStart = stCPU.u64CycleCount;
00100 }
00101
00102 //-----
00103 void KA_Command_Profile_Stop(void)
00104 {
00105     u64ProfileTotal += (stCPU.u64CycleCount - u64ProfileEpochStart);
00106     u64ProfileEpochStart = 0;
00107     u64ProfileCount++;
00108 }
00109 }
00110
00111 //-----
00112 void KA_Command_Profile_Report(void)
00113 {
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 src/kernel\_aware/ka\_profile.h File Reference

Mark3 RTOS Kernel-Aware Profiling.

### Functions

- void [KA\\_Profile\\_Init](#) (void)  
*KA\_Profile\_Init.*
- void [KA\\_Command\\_Profile\\_Begin](#) (void)  
*KA\_Command\_Profile\_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 Profiling.

Definition in file [ka\\_profile.h](#).

### 4.101.2 Function Documentation

#### 4.101.2.1 KA\_Profile\_Init()

```
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

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/ ( ((/ (      \      ( ( ((/ (      | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( ) \      / ( )      | -- [ Little ] -----
00006 *      ( ) _ ( )      ) \ _ \ ( ( ) ( ) ( )      | -- [ AVR ] -----
00007 *      | _ | |      ( ) _ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _ / _ \ \ v / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _ _ / _ / \ \ / | _ \      |
00010 *                                          | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __KA_PROFILE_H__
00022 #define __KA_PROFILE_H__
00023
00024 //-----
00031 void KA\_Profile\_Init(void);
00032
00033 //-----
00037 void KA\_Command\_Profile\_Begin(void);
00038
00039 //-----
00043 void KA\_Command\_Profile\_Start(void);
00044
00045 //-----
00049 void KA\_Command\_Profile\_Stop(void);
00050
00051 //-----
00055 void KA\_Command\_Profile\_Report(void);
00056
00057 #endif
```

## 4.103 src/kernel\_aware/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\_)



```

00064
00066     uint16_t u16StackSize;
00067
00069     uint16_t u16CurrentThreadList;
00070     uint16_t u16OwnerThreadList;
00071
00073     uint16_t u16EntryPoint;
00074
00076     void *m_pvArg;
00077
00079     uint16_t u16Quantum;
00080
00081 } Mark3_Thread_t;
00082
00083 //-----
00084 typedef struct
00085 {
00086     Mark3_Thread_t *pstThread;
00087     uint8_t u8ThreadID;
00088     uint64_t u64TotalCycles;
00089     uint64_t u64EpockCycles;
00090     bool bActive;
00091 } Mark3_Thread_Info_t;
00092
00093 //-----
00094 typedef struct
00095 {
00096     uint64_t u64Timestamp;
00097     uint16_t u16StackMargin;
00098     uint8_t u8ThreadID;
00099     uint8_t u8ThreadPri;
00100 } Mark3ContextSwitch_TLV_t;
00101
00102 //-----
00103 static uint64_t u64IdleTime = 0;
00104 static FILE *fKernelState = NULL;
00105 static FILE *fInterrupts = NULL;
00106 static Mark3_Thread_Info_t *pstThreadInfo = NULL;
00107 static uint16_t u16NumThreads = 0;
00108
00109 static Mark3_Thread_t *pstLastThread = NULL;
00110 static uint64_t u64LastTime = 0;
00111 static uint8_t u8LastPri = 255;
00112 //-----
00113 static TLV_t *pstTLV = NULL;
00114
00115 //-----
00116 static void Mark3KA_AddKnownThread( Mark3_Thread_t *pstThread_ )
00117 {
00118     // Bail if the thread pointer is NULL
00119     if (!pstThread_ || ((uint32_t)pstThread_ == (uint32_t)stCPU.pstRAM->au8RAM))
00120     {
00121         return;
00122     }
00123
00124     // Check to see if a thread has already been tagged at this address
00125     bool bExists = false;
00126     if (pstThreadInfo)
00127     {
00128         int i;
00129         for (i = 0; i < u16NumThreads; i++)
00130         {
00131             Mark3_Thread_t *pstThread = pstThreadInfo[i].pstThread;
00132             // If there are other threads that exist at this address,
00133             if (pstThread == pstThread_)
00134             {
00135                 // If the stored thread's ID is different than the ID being presented here,
00136                 // then it's a dynamic thread involved. Create a new threadinfo object to track it.
00137                 if (pstThreadInfo[i].u8ThreadID != pstThread->u8ThreadID)
00138                 {
00139                     pstThreadInfo[i].bActive = false;
00140                 }
00141                 // Thread IDs are the same, thread has already been tracked, don't do anything.
00142                 else
00143                 {
00144                     bExists = true;
00145                 }
00146             }
00147         }
00148     }
00149
00150     // If not already known, add the thread to the list of known threads.
00151     if (!bExists)
00152     {
00153         u16NumThreads++;
00154         pstThreadInfo = (Mark3_Thread_Info_t*)realloc(pstThreadInfo, sizeof(
Mark3_Thread_Info_t) * u16NumThreads);

```

```

00155
00156     pstThreadInfo[u16NumThreads - 1].pstThread = pstThread_;
00157     pstThreadInfo[u16NumThreads - 1].u64EpockCycles = 0;
00158     pstThreadInfo[u16NumThreads - 1].u64TotalCycles = 0;
00159     pstThreadInfo[u16NumThreads - 1].u8ThreadID = pstThread_>u8ThreadID;
00160     pstThreadInfo[u16NumThreads - 1].bActive = true;
00161 }
00162 }
00163
00164 //-----
00165 Mark3_Thread_t *Mark3KA_GetCurrentThread(void)
00166 {
00167     Debug_Symbol_t *pstSymbol = 0;
00168
00169     pstSymbol = Symbol_Find_Obj_By_Name( "g_pclCurrent" );
00170
00171     // Use pstSymbol's address to get a pointer to the current thread.
00172     if (!pstSymbol)
00173     {
00174         return 0;
00175     }
00176
00177     uint16_t u16CurrPtr = (uint16_t)(pstSymbol->u32StartAddr & 0x0000FFFF);
00178     if (!u16CurrPtr)
00179     {
00180         return 0;
00181     }
00182
00183     // Now that we have the address of g_pstCurrent, dereference the pointer
00184     // to get the address of the current thread.
00185
00186     uint16_t u16CurrAddr = ((uint16_t)(stCPU.pstRAM->au8RAM[ u16CurrPtr + 1 ]) << 8) +
00187         stCPU.pstRAM->au8RAM[ u16CurrPtr ];
00188
00189     // Return a pointer to the thread as it is in memory.
00190     return (Mark3_Thread_t*)(&stCPU.pstRAM->au8RAM[ u16CurrAddr ]);
00191 }
00192
00193 //-----
00194 static uint8_t Mark3KA_GetCurrentPriority(void)
00195 {
00196     Mark3_Thread_t *pstThread = Mark3KA_GetCurrentThread();
00197     if (!pstThread)
00198     {
00199         return 0;
00200     }
00201     uint8_t *pucData = (uint8_t*)pstThread;
00202
00203     // If the curpriority member is set, it means we're in the middle of
00204     // priority inheritance. If it's zero, return the normal priority
00205     if (0 == pstThread->u8CurPriority)
00206     {
00207         return pstThread->u8Priority;
00208     }
00209     return pstThread->u8CurPriority;
00210 }
00211
00212 //-----
00213 static uint16_t Mark3KA_GetStackMargin( Mark3_Thread_t *pstThread_ )
00214 {
00215     uint16_t u16StackBase = pstThread_>u16StackPtr;
00216     uint16_t u16StackSize = pstThread_>u16StackSize;
00217
00218     int i;
00219
00220     for (i = 0; i < u16StackSize; i++)
00221     {
00222         if (255 != stCPU.pstRAM->au8RAM[ u16StackBase + i ])
00223         {
00224             return (uint16_t)i;
00225         }
00226     }
00227
00228     return u16StackSize;
00229 }
00230
00231 //-----
00232 static uint16_t Mark3KA_GetCurrentStackMargin(void)
00233 {
00234     return Mark3KA_GetStackMargin( Mark3KA_GetCurrentThread() );
00235 }
00236
00237 //-----
00238 static bool KA_StackWarning( uint16_t u16Addr_, uint8_t u8Data_ )
00239 {
00240     if (u8Data_ != 0xFF && stCPU.pstRAM->au8RAM[ u16Addr_ ] == 0xFF )
00241     {

```

```

00242         fprintf( stderr, "[WARNING] Near stack-overflow detected - Thread %d, Stack Margin %d\n",
00243                 Mark3KA_GetCurrentThread()->u8ThreadID,
00244                 Mark3KA_GetCurrentStackMargin() );
00245     }
00246     return true;
00247 }
00248
00249 //-----
00250 static bool KA_ThreadChange( uint16_t u16Addr_, uint8_t u8Data_ )
00251 {
00252     uint8_t u8Pri = Mark3KA_GetCurrentPriority();
00253     uint8_t u8Thread = Mark3KA_GetCurrentThread()->u8ThreadID;
00254     uint16_t u16Margin = Mark3KA_GetCurrentStackMargin();
00255
00256     // -- Add context switch instrumentation to TLV
00257     Mark3ContextSwitch_TLV_t stData;
00258
00259     stData.u8ThreadID = u8Thread;
00260     stData.u8ThreadPri = u8Pri;
00261     stData.u16StackMargin = u16Margin;
00262     stData.u64Timestamp = stCPU.u64CycleCount;
00263
00264     memcpy( &(pstTLV->au8Data[0]), &stData, sizeof(stData) );
00265     TLV_Write( pstTLV );
00266
00267     if (u8LastPri == 0)
00268     {
00269         u64IdleTime += (stCPU.u64CycleCount - u64LastTime);
00270     }
00271
00272     // Track this as a known-thread internally for future reporting.
00273     Mark3KA_AddKnownThread( Mark3KA_GetCurrentThread() );
00274
00275     if (pstLastThread && u64LastTime)
00276     {
00277         Mark3_Thread_t *pstThread;
00278         int i;
00279         for ( i = 0; i < u16NumThreads; i++ )
00280         {
00281             if ( (pstLastThread == pstThreadInfo[i].pstThread) &&
00282                 (pstLastThread->u8ThreadID == pstThreadInfo[i].u8ThreadID) )
00283             {
00284                 pstThreadInfo[i].u64TotalCycles += stCPU.u64CycleCount - u64LastTime;
00285             }
00286         }
00287     }
00288
00289     u64LastTime = stCPU.u64CycleCount;
00290     u8LastPri = u8Pri;
00291
00292     // Add watchpoints on active thread stack at 32-bytes from the end
00293     // of the stack. That way, we can immediately detect stack smashing threats
00294     // without having to hunt.
00295
00296     uint16_t u16StackWarning = Mark3KA_GetCurrentThread()->u16StackPtr + 32;
00297     WriteCallout_Add( KA_StackWarning, u16StackWarning );
00298
00299     // Cache the current thread for use as the "last run" thread in
00300     // subsequent iterations
00301     pstLastThread = Mark3KA_GetCurrentThread();
00302
00303     return true;
00304 }
00305
00306 //-----
00307 void KA_PrintThreadInfo(void)
00308 {
00309     int i;
00310     uint64_t u64TrackedThreadTime = 0;
00311
00312     uint16_t u16LastThread = (uint16_t)((void*)Mark3KA_GetCurrentThread() - (void*)&stCPU.pstRAM->au8RAM[0]
);
00313
00314     KA_ThreadChange( u16LastThread, 0 );
00315
00316     for ( i = 0; i < u16NumThreads; i++ )
00317     {
00318         u64TrackedThreadTime += pstThreadInfo[i].u64TotalCycles;
00319     }
00320
00321     printf( "ThreadID, ThreadAddr, TotalCycles, PercentCPU, IsActive, Prio, StackMargin\n");
00322     for ( i = 0; i < u16NumThreads; i++ )
00323     {
00324         printf( "%d, %04X, %llu, %0.3f, %d, %d, %d\n",
00325                 pstThreadInfo[i].u8ThreadID,
00326                 (uint16_t)((void*)(pstThreadInfo[i].pstThread) - (void*)&stCPU.pstRAM->au8RAM[0])),
00327                 pstThreadInfo[i].u64TotalCycles,

```



```

00328         (double)pstThreadInfo[i].u64TotalCycles / u64TrackedThreadTime * 100.0f,
00329         pstThreadInfo[i].bActive,
00330         (pstThreadInfo[i].bActive ? pstThreadInfo[i].pstThread->
    u8Priority : 0),
00331         (pstThreadInfo[i].bActive ? Mark3KA_GetStackMargin(pstThreadInfo[i].pstThread) : 0)
00332     ) ;
00333 }
00334 }
00335
00336 //-----
00337 void KA_Thread_Init( void )
00338 {
00339     Debug_Symbol_t *pstSymbol = 0;
00340     pstSymbol = Symbol_Find_Obj_By_Name( "g_pclCurrent" );
00341
00342     // Use pstSymbol's address to get a pointer to the current thread.
00343     if (!pstSymbol)
00344     {
00345         return;
00346     }
00347
00348     // Ensure that we actually have the information we need at a valid address
00349     uint16_t ul6CurrPtr = (uint16_t)(pstSymbol->u32StartAddr & 0x0000FFFF);
00350     if (!ul6CurrPtr)
00351     {
00352         return;
00353     }
00354
00355     // Add a callback so that when g_pstCurrent changes, we can update our
00356     // locally-tracked statistics.
00357     WriteCallout_Add( KA_ThreadChange , ul6CurrPtr + 1 );
00358
00359     pstTLV = TLV_Alloc( sizeof(Mark3ContextSwitch_TLV_t) );
00360     pstTLV->eTag = TAG_KERNEL_AWARE_CONTEXT_SWITCH;
00361     pstTLV->ul6Len = sizeof(Mark3ContextSwitch_TLV_t);
00362
00363     atexit( KA_PrintThreadInfo );
00364 }
00365
00366 //-----
00367 char *KA_Get_Thread_Info_XML(uint8_t **thread_ids, uint16_t *thread_count)
00368 {
00369     char *ret = (char*)malloc(4096);
00370     char *writer = ret;
00371     uint8_t *new_ids;
00372
00373     if (ul6NumThreads && thread_ids)
00374     {
00375         new_ids = (uint8_t*)malloc(ul6NumThreads);
00376         *thread_ids = new_ids;
00377     }
00378
00379     writer += sprintf( writer,
00380         "<threads>" );
00381
00382     if (!ul6NumThreads) {
00383         writer += sprintf( writer,
00384             " <thread id=\"0\" core=\"0\">"
00385             " System Thread - Priority N/A [Running] "
00386             " </thread>");
00387     }
00388
00389     int i;
00390     int count = 0;
00391     for (i = 0; i < ul6NumThreads; i++)
00392     {
00393         if (pstThreadInfo[i].bActive)
00394         {
00395             if (pstThreadInfo[i].u8ThreadID == 255)
00396             {
00397                 writer += sprintf(writer,
00398                     " <thread id=\"255\" core=\"0\">"
00399                     " Mark3 Thread - Priority 0 [IDLE]" );
00400             }
00401             else if (pstThreadInfo[i].u8ThreadID == Mark3KA_GetCurrentThread()->u8ThreadID)
00402             {
00403                 writer += sprintf(writer,
00404                     " <thread id=\"%d\" core=\"0\">"
00405                     " Mark3 Thread - Priority %d [Running] " ,
00406                     pstThreadInfo[i].u8ThreadID,
00407                     pstThreadInfo[i].pstThread->u8CurPriority );
00408             }
00409             else
00410             {
00411                 writer += sprintf(writer,
00412                     " <thread id=\"%d\" core=\"0\">"
00413                     " Mark3 Thread - Priority %d" ,

```

```

00414         pstThreadInfo[i].u8ThreadID,
00415         pstThreadInfo[i].pstThread->u8CurPriority );
00416     }
00417     if (thread_ids)
00418     {
00419         new_ids[count++] = pstThreadInfo[i].u8ThreadID;
00420     }
00421 }
00422 writer += sprintf( writer, " </thread>");
00423 }
00424
00425 sprintf( writer, "</threads>" );
00426 if (thread_count)
00427 {
00428     *thread_count = count;
00429 }
00430 return ret;
00431 }
00432
00433 //-----
00434 Mark3_Context_t *KA_Get_Thread_Context(uint8_t id_)
00435 {
00436     int i;
00437     for (i = 0; i < u16NumThreads; i++)
00438     {
00439         if (pstThreadInfo[i].bActive)
00440         {
00441             if (pstThreadInfo[i].u8ThreadID == id_)
00442             {
00443                 Mark3_Context_t *new_ctx = (Mark3_Context_t*)malloc(sizeof(
Mark3_Context_t));
00444                 uint16_t context_addr = pstThreadInfo[i].pstThread->
u16StackTopPtr;
00445
00446                 new_ctx->SPH = stCPU.pstRAM->au8RAM[context_addr - 1];
00447                 new_ctx->SPL = stCPU.pstRAM->au8RAM[context_addr];
00448
00449                 int j = 0;
00450                 for (i = 31; i >= 0; i--)
00451                 {
00452                     new_ctx->r[i] = stCPU.pstRAM->au8RAM[context_addr + 1 + j];
00453                     j++;
00454                 }
00455                 new_ctx->SREG = stCPU.pstRAM->au8RAM[context_addr + 33];
00456                 uint16_t PC = *(uint16_t*)(stCPU.pstRAM->au8RAM[context_addr + 34]);
00457                 PC = ((PC & 0xFF00)>>8) | ((PC & 0x00FF) << 8);
00458                 new_ctx->PC = PC;
00459
00460                 return new_ctx;
00461             }
00462         }
00463     }
00464     return NULL;
00465 }
00466
00467 //-----
00468 int KA_Get_Thread_ID(void)
00469 {
00470     return Mark3KA_GetCurrentThread()->u8ThreadID;
00471 }
00472
00473 //-----
00474 int KA_Get_Thread_Priority( int id_ )
00475 {
00476     int i;
00477     for (i = 0; i < u16NumThreads; i++)
00478     {
00479         if (pstThreadInfo[i].bActive)
00480         {
00481             if (pstThreadInfo[i].u8ThreadID == id_)
00482             {
00483                 return pstThreadInfo[i].pstThread->u8CurPriority;
00484             }
00485         }
00486     }
00487     return -1;
00488 }
00489
00490 //-----
00491 const char *KA_Get_Thread_State( int id_ )
00492 {
00493     int i;
00494     for (i = 0; i < u16NumThreads; i++)
00495     {
00496         if (pstThreadInfo[i].bActive)
00497         {
00498             if (pstThreadInfo[i].u8ThreadID == id_)

```

```

00499         {
00500             switch (pstThreadInfo[i].pstThread->u8ThreadState)
00501             {
00502                 case THREAD_STATE_BLOCKED:
00503                     return "Blocked";
00504                 case THREAD_STATE_EXIT:
00505                     return "Exit";
00506                 case THREAD_STATE_READY:
00507                     if (id_ == Mark3KA_GetCurrentThread()->u8ThreadID)
00508                     {
00509                         return "Running";
00510                     }
00511                     return "Ready";
00512                 case THREAD_STATE_STOP:
00513                     return "Stopped";
00514                 default:
00515                     return "unknown";
00516             }
00517         }
00518     }
00519 }
00520 return -1;
00521 }

```

## 4.105 src/kernel\_aware/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

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( ((/(      )\      ( ((/(      | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( )\ )\ / ( )      | -- [ Little ] -----
00006 *      ( ) _ ( )      )\ _ )\ ( ( ) ( ( )      | -- [ AVR ] -----
00007 *      | _ | |      ( ) _ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ V / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _      / _ \ \ \ \ / | _ \      |
00010 *      | _ | | _      / _ \ \ \ \ / | _ \      | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __KA_THREAD_H__
00022 #define __KA_THREAD_H__
00023
00024 #include <stdint.h>
00025
00026 typedef struct
00027 {
00028     uint8_t SPH;
00029     uint8_t SPL;
00030     uint8_t r[32];
00031     uint8_t SREG;
00032     uint16_t PC;
00033 } Mark3_Context_t;
00034
00035 //-----
00036 void KA_Thread_Init( void );
00037
00038 int KA_Get_Thread_Priority(int id_);
00039
00040 const char *KA_Get_Thread_State( int id_ );
00041
00042 #endif

```

## 4.107 src/kernel\_aware/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 KA\_EmitTrace()

```
void KA_EmitTrace (
    KernelAwareCommand_t eCmd_ )
```

KA\_EmitTrace.

Process a kernel trace event and emit the appropriate record into our TLV stream output

#### Parameters

<i>e↔ Cmd↔ _</i>	Type of trace command being emitted.
--------------------------	--------------------------------------

Definition at line 47 of file [ka\\_trace.c](#).

#### 4.107.2.2 KA\_Print()

```
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 KA\_Trace\_Init()

```
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 *      /( )  /( )  ((( ( ) \   \/( )  -- [ Little ] -----
00006 *      ( ) _ ( )   ) \ _ ) \ ( ( ( _ ( )  -- [ AVR ] -----
00007 *      | _ | | |   ( ) _ ( ) \ \ / / | _ \  -- [ Virtual ] -----
00008 *      | _ | | |   / _ \ \ \ / / | _ \  -- [ Runtime ] -----
00009 *      | _ | | |   / _ \ \ \ / / | _ \  |
00010 *                                     | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, 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 ul6File;
00037     uint16_t ul6Line;
00038     uint16_t ul6Code;
00039     uint16_t ul6Arg1;
00040     uint16_t ul6Arg2;
00041 } KernelAwareTrace_t;
00042
00043 //-----
00044 static TLV_t *pstTLV = NULL;
00045
00046 //-----
00047 void KA_EmitTrace( KernelAwareCommand_t eCmd_ )
00048 {
00049     Debug_Symbol_t *pstSymbol = Symbol_Find_Obj_By_Name( "g_stKAData"
00050 );
00051     if (!pstSymbol)
00052     {
00053         return;
00054     }
00055     KernelAwareTrace_t *pstTrace = (KernelAwareTrace_t*)&stCPU.
00056     pstRAM->au8RAM[ pstSymbol->u32StartAddr ];
00057     switch (eCmd_)
00058     {
00059     case KA_COMMAND_TRACE_0:
00060         pstTLV->eTag = KA_COMMAND_TRACE_0;
00061         pstTLV->ul6Len = 6;
00062         break;
00063     case KA_COMMAND_TRACE_1:
00064         pstTLV->eTag = KA_COMMAND_TRACE_1;
00065         pstTLV->ul6Len = 8;
00066         break;
00067     case KA_COMMAND_TRACE_2:
00068         pstTLV->eTag = KA_COMMAND_TRACE_2;
```

```

00068         pstTLV->u16Len = 10;
00069         break;
00070     default:
00071         return;
00072     }
00073     fprintf(stderr, "Trace: %04X, %04X, %04X, %04X, %04x\n", pstTrace->u16File, pstTrace->u16Line,
00074                pstTrace->u16Code, pstTrace->u16Arg1, pstTrace->u16Arg2 );
00075
00076     memcpy( pstTLV->au8Data, pstTrace, pstTLV->u16Len );
00077     TLV_Write( pstTLV );
00078 }
00079
00080 //-----
00081 void KA_Print( void )
00082 {
00083     Debug_Symbol_t *pstSymbol = Symbol_Find_Obj_By_Name( "g_stKADData"
00084 );
00085     if (!pstSymbol)
00086     {
00087         return;
00088     }
00089     uint16_t u16NamePtr = *((uint16_t*)&stCPU.pstRAM->au8RAM[ pstSymbol->
00090 u32StartAddr ]);
00091     const char *szString = (const char*)&stCPU.pstRAM->au8RAM[ u16NamePtr ];
00092     strcpy( pstTLV->au8Data, szString );
00093     fprintf( stderr, "%s", szString );
00094 }
00095
00096 //-----
00097 void KA_Trace_Init(void)
00098 {
00099     pstTLV = TLV_Alloc(64);
00100 }

```

## 4.109 src/kernel\_aware/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 KA\_EmitTrace()

```
void KA_EmitTrace (
    KernelAwareCommand_t eCmd_ )
```

KA\_EmitTrace.

Process a kernel trace event and emit the appropriate record into our TLV stream output

#### Parameters

<i>eCmd_</i>	Type of trace command being emitted.
--------------	--------------------------------------

Definition at line 47 of file [ka\\_trace.c](#).

### 4.109.2.2 KA\_Print()

```
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 KA\_Trace\_Init()

```
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

```

00001  /*****
00002  *      (      (      (      |
00003  *      )\ ) )\ )      (      |
00004  *      ( )/( ( )/(      )\      (      (      ( )/(      | -- [ Funkenstein ] -----
00005  *      /( ) )/( ) ((( ( )\      )\      /( )      | -- [ Little ] -----
00006  *      ( )_ ( )      )\ _ )\      ( ) ( ) ( )      | -- [ AVR ] -----
00007  *      | | _ | |      ( )_ \ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008  *      | _ | | _      / _ \      \ v / | _ /      | -- [ Runtime ] -----
00009  *      | _ | | _ _ | / \ \ \      \ /      | _ | _ |      |
00010  *      | "Yeah, it does Arduino..."
00011  * -----+-----
00012  * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013  * See license.txt for details
00014  *****/
00021  #ifndef __KA_TRACE__
00022  #define __KA_TRACE__
00023
00024
00025  #include "kernel_aware.h"
00026  #include "debug_sym.h"
00027
00028  #include "ka_trace.h"
00029  #include "tlv_file.h"
00030
00031  #include <stdint.h>
00032  #include <stdio.h>
00033  #include <stdlib.h>
00034  #include <string.h>
00035  #include <time.h>
00036
00037  //-----
00046  void KA_EmitTrace( KernelAwareCommand_t eCmd_ );
00047
00048  //-----
00054  void KA_Print( void );
00055
00056  //-----
00062  void KA_Trace_Init( void );
00063
00064  #endif
00065

```

#### 4.111 src/kernel\_aware/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 KernelAware\_Init()

```
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

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/ ( ((/ (      \      | -- [ Funkenstein ] -----
00005 *      /( ) )/( ) )((( ( ) \      \ / ( )      | -- [ Little ] -----
00006 *      ( ) _ ( )      ) \ _      \ ( ( ( ) ( )      | -- [ AVR ] -----
00007 *      | | _ | |      ( ) _ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \      \ V / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _      / _ \      \ V / | _ \      |
00010 *      | _ | | _      / _ \      \ V / | _ \      | "Yeah, it does Arduino..."
00011 *      | _ | | _      / _ \      \ V / | _ \      |
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #include "kernel_aware.h"
00022 #include "debug_sym.h"
00023 #include "write_callout.h"
00024 #include "interrupt_callout.h"
00025
00026 #include "ka_interrupt.h"
00027 #include "ka_profile.h"
00028 #include "ka_thread.h"
00029 #include "ka_trace.h"
00030 #include "ka_graphics.h"
00031 #include "ka_joystick.h"
00032
00033 #include <stdint.h>
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <time.h>
00038
00039
00040 //-----
00041 static bool KA_Command( uint16_t u16Addr_, uint8_t u8Data_ )
00042 {
00043     switch (u8Data_)
00044     {
00045     case KA_COMMAND_PROFILE_INIT:    KA_Command_Profile_Begin();    break;
00046     case KA_COMMAND_PROFILE_STOP:    KA_Command_Profile_Stop();    break;
```

```

00047     case KA_COMMAND_PROFILE_START:    KA_Command_Profile_Start ();        break;
00048     case KA_COMMAND_PROFILE_REPORT:   KA_Command_Profile_Report ();        break;
00049     case KA_COMMAND_TRACE_0:
00050     case KA_COMMAND_TRACE_1:
00051     case KA_COMMAND_TRACE_2:          KA_EmitTrace(u8Data_);              break;
00052     case KA_COMMAND_PRINT:            KA_Print ();                        break;
00053     default:
00054         break;
00055
00056     }
00057     return true;
00058 }
00059
00060 //-----
00061 static bool KA_Set( uint16_t u16Addr_, uint8_t u8Data_ )
00062 {
00063     fprintf(stderr, "ADDR: [%04X], Data: [%02X]\n", u16Addr_, u8Data_ );
00064     stCPU.pstRAM->au8RAM[ u16Addr_ & 0xFFFF ] = 1;
00065     return false;
00066 }
00067
00068 //-----
00069 void KernelAware_Init( void )
00070 {
00071     Debug_Symbol_t *pstSymbol = 0;
00072
00073     // Add a callout for profiling information (present in Mark3 Unit Tests)
00074     pstSymbol = Symbol_Find_Obj_By_Name( "g_u8KACommand" );
00075     if (pstSymbol)
00076     {
00077         // Ensure that we actually have the information we need at a valid address
00078         uint16_t u16CurrPtr = (uint16_t)(pstSymbol->u32StartAddr & 0x0000FFFF);
00079         printf( "found kernel-aware command @ %04X\n", u16CurrPtr );
00080         if (u16CurrPtr)
00081         {
00082             // Add a callback so that when profiling state changes, we do something.
00083             WriteCallout_Add( KA_Command , u16CurrPtr );
00084         }
00085     }
00086     else
00087     {
00088         printf( "Unable to find g_u8KACommand\n" );
00089     }
00090
00091     // Set the kernel's "simulator aware" flag, to let it know to configure itself
00092     // appropriately.
00093
00094     pstSymbol = Symbol_Find_Obj_By_Name( "g_bIsKernelAware" );
00095     if (pstSymbol)
00096     {
00097         fprintf( stderr, "Addr: %4X, Name: %s\n", pstSymbol->u32StartAddr, pstSymbol->
szName );
00098         // Ensure that we actually have the information we need at a valid address
00099         uint16_t u16CurrPtr = (uint16_t)(pstSymbol->u32StartAddr);
00100
00101         if (u16CurrPtr)
00102         {
00103             // Add a callout so that the kernel-aware flag is *always* set.
00104             fprintf( stderr, "Adding writeout\n" );
00105             WriteCallout_Add( KA_Set , u16CurrPtr );
00106             fprintf( stderr, "done\n" );
00107         }
00108     }
00109     else
00110     {
00111         printf( "Unable to find g_bIsKernelAware" );
00112     }
00113
00114
00115     KA_Interrupt_Init();
00116     KA_Thread_Init();
00117     KA_Profile_Init();
00118     KA_Trace_Init();
00119
00120     KA_Graphics_Init();
00121     KA_Joystick_Init();
00122
00123 }
00124 }

```

## 4.113 src/kernel\_aware/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 KernelAware\_Init()

```
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

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ )  )\ )      (      )\ )  |
00004 *      ((/( ((/(      )\      ( ((/(      | -- [ Funkenstein ] -----
00005 *      /( )  /( )  ((( ( ( )\  )\  /( )  | -- [ Little ] -----
00006 *      ( )_ ( )_ )\ _ )\ ( ( ( ( )_ )_ | -- [ AVR ] -----
00007 *      | | _ | | ( )_ \ ( ) \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ / _ \ \ v / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ / _ \ \ \ / | _ \ |
00010 *      | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __KERNEL_AWARE_H__
00022 #define __KERNEL_AWARE_H__
00023
00024 #include "elf_process.h"
00025 #include "debug_sym.h"
00026 #include "avr_cpu.h"
00027
00028 #include <stdint.h>
00029
00030 //-----
00031 typedef enum
00032 {
00033     KA_COMMAND_IDLE = 0,
00034     KA_COMMAND_PROFILE_INIT,
00035     KA_COMMAND_PROFILE_START,
00036     KA_COMMAND_PROFILE_STOP,
00037     KA_COMMAND_PROFILE_REPORT,
00038     KA_COMMAND_EXIT_SIMULATOR,
00039     KA_COMMAND_TRACE_0,
00040     KA_COMMAND_TRACE_1,
00041     KA_COMMAND_TRACE_2,
00042     KA_COMMAND_PRINT
00043 } KernelAwareCommand_t;
00044
00045 //-----
00055 void KernelAware_Init(void);
00056
00057 void KA_Graphics_Init( void ) __attribute__((weak));
00058 void KA_Joystick_Init( void ) __attribute__((weak));
00059 #endif

```

## 4.115 src/kernel\_aware/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

### 4.115.1 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](#).

### 4.115.2 Function Documentation

#### 4.115.2.1 TLV\_Alloc()

```
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

<i>u16Len_</i>	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](#).

## 4.115.2.2 TLV\_Free()

```
void TLV_Free (
    TLV_t * pstTLV_ )
```

TLV\_Free.

Free a previously-allocated TLV object.

## Parameters

<i>pstTLV_</i>	Pointer to a valid, previously-allocated TLV object
----------------	---

Definition at line 61 of file [tlv\\_file.c](#).

## 4.115.2.3 TLV\_Read()

```
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

<i>pstTLV_</i>	Pointer to a valid TLV object, with a buffer large enough to hold the largest data object we may encounter.
<i>pu8Buffer_</i>	Pointer to a buffer containing the contents of the TLV input file.
<i>iIndex_</i>	Byte index at which 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](#).

## 4.115.2.4 TLV\_ReadFinish()

```
void TLV_ReadFinish (
    uint8_t * pu8Buffer_ )
```

TLV\_ReadFinish.

Dispose of the in-ram copy of the TLV read buffer, allocated from TLV\_ReadInit

**Parameters**

<i>pu8↔ Buffer_</i>	Pointer to the previously allocated TLV ram buffer
-------------------------	--

Definition at line 113 of file [tlv\\_file.c](#).

**4.115.2.5 TLV\_ReadInit()**

```
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**

<i>szPath_</i>	Path to the file to open
<i>pu8↔ Buffer_</i>	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](#).

**4.115.2.6 TLV\_Write()**

```
int TLV_Write (
    TLV_t * pstData_ )
```

TLV\_Write.

Write a TLV record to the active file stream.

**Parameters**

<i>pst↔ Data_</i>	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.

#### 4.115.2.7 TLV\_WriteInit()

```
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

sz↔ <i>Path</i>	Name of the TLV output file to create
--------------------	---------------------------------------

Definition at line 36 of file tlv\_file.c.

## 4.116 tlv\_file.c

```

00001  /*****
00002  *      (      (      (      |
00003  *      )\ )  )\ )      )\ )  |
00004  *      ()/( ()/(      )\      ( ( ()/(      | -- [ Funkenstein ] -----
00005  *      /(_) /(_) (((_) ()\      )\ /(_)      | -- [ Little ] -----
00006  *      ( )_ ( )      )\ _ )\ ( ( ( ( )      | -- [ AVR ] -----
00007  *      | _ | _      ( )_ \ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008  *      | _ | _      / _ \      \ v / | _ /      | -- [ Runtime ] -----
00009  *      | _ | _      / _ \      \ v / | _ /      |
00010  *      | _ | _      / _ \      \ v / | _ /      | "Yeah, it does Arduino..."
00011  *      +-----+
00012  * (c) Copyright 2014-17, 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  #include <string.h>
00026  #include <unistd.h>
00027  #include <sys/stat.h>
00028  #include <sys/types.h>
00029
00030  #include "tlv_file.h"
00031
00032  //-----
00033  static FILE *fMyFile = NULL;
00034
00035  //-----
00036  void TLV_WriteInit( const char *szPath_ )
00037  {
00038      if (!fMyFile)
00039      {
00040          fMyFile = fopen( szPath_, "wb" );
00041      }
00042  }
00043
00044  //-----
00045  void TLV_WriteFinish( void )

```

```

00046 {
00047     if (fMyFile)
00048     {
00049         fclose(fMyFile);
00050     }
00051     fMyFile = NULL;
00052 }
00053
00054 //-----
00055 TLV_t *TLV_Alloc( uint16_t ul6Len_ )
00056 {
00057     return (TLV_t*)(malloc(sizeof(TLV_t) + ul6Len_ - 1));
00058 }
00059
00060 //-----
00061 void TLV_Free( TLV_t *pstTLV_ )
00062 {
00063     free( pstTLV_ );
00064 }
00065
00066 //-----
00067 int TLV_Write( TLV_t *pstData_ )
00068 {
00069     if (fMyFile)
00070     {
00071         return fwrite( (void*)pstData_, sizeof(uint8_t), sizeof(TLV_t) + pstData_>
ul6Len - 1, fMyFile );
00072     }
00073     return -1;
00074 }
00075 //-----
00076 int TLV_ReadInit( const char *szPath_, uint8_t **pu8Buffer_ )
00077 {
00078     FILE *fReadFile = fopen( szPath_, "rb" );
00079     struct stat stStat;
00080
00081     if (!fReadFile)
00082     {
00083         fprintf(stderr, "Unable to open tlv for input!\n" );
00084         return 0;
00085     }
00086
00087     stat( szPath_, &stStat );
00088     *pu8Buffer_ = (uint8_t*)malloc( stStat.st_size );
00089     if (!pu8Buffer_)
00090     {
00091         fclose(fReadFile);
00092         fprintf(stderr, "Unable to allocate local tlv read buffer!\n" );
00093         return 0;
00094     }
00095     fread(*pu8Buffer_, 1, stStat.st_size, fReadFile );
00096
00097     fclose(fReadFile);
00098     return stStat.st_size;
00099 }
00100
00101 //-----
00102 int TLV_Read( TLV_t *pstTLV_, uint8_t *pu8Buffer_, int iIndex_)
00103 {
00104     TLV_t *pstStreamTLV = (TLV_t*)&(pu8Buffer_[iIndex_]);
00105     pstTLV_>eTag = pstStreamTLV->eTag;
00106     pstTLV_>ul6Len = pstStreamTLV->ul6Len;
00107     memcpy( pstTLV_>au8Data, pstStreamTLV->au8Data, pstTLV_>
ul6Len );
00108     return (sizeof(TLV_t) + pstTLV_>ul6Len - 1);
00109 }
00110
00111 //-----
00112 void TLV_ReadFinish ( uint8_t *pu8Buffer_ )
00113 {
00114     if (pu8Buffer_)
00115     {
00116         free( pu8Buffer_ );
00117     }
00118 }
00119 }

```

#### 4.117 src/kernel\_aware/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\_)
- int [TLV\\_Write](#) ([TLV\\_t](#) \*pstData\_)  
*TLV\_Write.*
- int [TLV\\_ReadInit](#) (const char \*szPath\_, uint8\_t \*\*pu8Buffer\_)
- int [TLV\\_Read](#) ([TLV\\_t](#) \*pstTLV\_, uint8\_t \*pu8Buffer\_, int iIndex\_)  
*TLV\_Read.*
- void [TLV\\_ReadFinish](#) (uint8\_t \*pu8Buffer\_)
- *TLV\_ReadFinish.*

### 4.117.1 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](#).

### 4.117.2 Enumeration Type Documentation

#### 4.117.2.1 FlavrTag\_t

```
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](#).

### 4.117.3 Function Documentation

#### 4.117.3.1 TLV\_Alloc()

```
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

<i>u16Len_</i>	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](#).

## 4.117.3.2 TLV\_Free()

```
void TLV_Free (
    TLV_t * pstTLV_ )
```

TLV\_Free.

Free a previously-allocated TLV object.

## Parameters

<i>pstTLV_</i>	Pointer to a valid, previously-allocated TLV object
----------------	---

Definition at line 61 of file [tlv\\_file.c](#).

## 4.117.3.3 TLV\_Read()

```
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

<i>pstTLV_</i>	Pointer to a valid TLV object, with a buffer large enough to hold the largest data object we may encounter.
<i>pu8Buffer_</i>	Pointer to a buffer containing the contents of the TLV input file.
<i>iIndex_</i>	Byte index at which 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](#).

## 4.117.3.4 TLV\_ReadFinish()

```
void TLV_ReadFinish (
    uint8_t * pu8Buffer_ )
```

TLV\_ReadFinish.

Dispose of the in-ram copy of the TLV read buffer, allocated from TLV\_ReadInit

**Parameters**

<i>pu8↔ Buffer_</i>	Pointer to the previously allocated TLV ram buffer
-------------------------	--

Definition at line 113 of file [tlv\\_file.c](#).

**4.117.3.5 TLV\_ReadInit()**

```
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**

<i>szPath_</i>	Path to the file to open
<i>pu8↔ Buffer_</i>	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](#).

**4.117.3.6 TLV\_Write()**

```
int TLV_Write (
    TLV_t * pstData_ )
```

TLV\_Write.

Write a TLV record to the active file stream.

**Parameters**

<i>pst↔ Data_</i>	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.

#### 4.117.3.7 TLV\_WriteInit()

```
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

sz↔ <i>Path</i>	Name of the TLV output file to create
--------------------	---------------------------------------

Definition at line 36 of file tlv\_file.c.

## 4.118 tlv\_file.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ()/( ()/( )\      ( ( ()/( | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( )\ )\ /( ) | -- [ Little ] -----
00006 *      ( )_ ( ) )\ _ )\ ( ( ( ( ) | -- [ AVR ] -----
00007 *      | | | |      ( )\ ( )\ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ | / _ \ \ \ v / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ _ | / \ \ \ \ \ \ | _ | _ | |
00010 *
00011 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00022 #ifndef __TLV_FILE_H__
00023 #define __TLV_FILE_H__
00024
00025 #include <stdint.h>
00026 #include <stdio.h>
00027 #include <stdlib.h>
00028 #include <string.h>
00029
00030 //-----
00031 typedef enum
00032 {
00033     TAG_KERNEL_AWARE_INTERRUPT,
00034     TAG_KERNEL_AWARE_CONTEXT_SWITCH,
00035     TAG_KERNEL_AWARE_PRINT,
00036     TAG_KERNEL_AWARE_TRACE_0,
00037     TAG_KERNEL_AWARE_TRACE_1,
00038     TAG_KERNEL_AWARE_TRACE_2,
00039     TAG_KERNEL_AWARE_PROFILE,
00040     TAG_KERNEL_AWARE_THREAD_PROFILE_EPOCH,
00041     TAG_KERNEL_AWARE_THREAD_PROFILE_GLOBAL,
00042     TAG_CODE_PROFILE_FUNCTION_EPOCH,
00043     TAG_CODE_PROFILE_FUNCTION_GLOBAL,
00044     TAG_CODE_COVERAGE_FUNCTION_EPOCH,
00045     TAG_CODE_COVERAGE_FUNCTION_GLOBAL,

```

```

00046     TAG_CODE_COVERAGE_GLOBAL,
00047     TAG_CODE_COVERAGE_ADDRESS,
00048 //---
00049     TAG_COUNT
00050 } FlavrTag_t;
00051
00052 //-----
00053 typedef struct
00054 {
00055     FlavrTag_t eTag;
00056     uint16_t   u16Len;
00057     uint8_t    au8Data[1];
00058 } TLV_t;
00059
00060 //-----
00061 void TLV_WriteInit( const char *szPath_ );
00070
00071 void TLV_WriteFinish( void );
00072 //-----
00082 TLV_t *TLV_Alloc( uint16_t u16Len_ );
00083
00084 //-----
00092 void TLV_Free( TLV_t *pstTLV_ );
00093
00094 //-----
00103 int TLV_Write( TLV_t *pstData_ );
00104
00105 //-----
00119 int TLV_ReadInit( const char *szPath_, uint8_t **pu8Buffer_ );
00120
00121 //-----
00138 int TLV_Read( TLV_t *pstTLV_, uint8_t *pu8Buffer_, int iIndex_ );
00139
00140 //-----
00149 void TLV_ReadFinish( uint8_t *pu8Buffer_ );
00150
00151 #endif

```

## 4.119 src/loader/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.*



### 4.119.1 Detailed Description

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

Definition in file [avr\\_loader.c](#).

### 4.119.2 Function Documentation

#### 4.119.2.1 AVR\_Load\_ELF()

```
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

<i>szFile↵ Path_</i>	Pointer to the elf-file path
--------------------------	------------------------------

##### Returns

true if the elf file load operation succes

Definition at line 142 of file [avr\\_loader.c](#).

#### 4.119.2.2 AVR\_Load\_HEX()

```
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

<i>szFile↵ Path_</i>	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](#).

## 4.120 avr\_loader.c

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( (/(      (      | -- [ Funkenstein ] -----
00005 *      /( ) )/( )\ (( ( )\ )\ /( )\ | -- [ Litle ] -----
00006 *      ( )\ ( )\ )\ )\ ( )\ ( )\ ( )\ | -- [ AVR ] -----
00007 *      | | | | ( )\ ( )\ \ / / | | \ | -- [ Virtual ] -----
00008 *      | | | | \ / \ \ \ / / | | \ | -- [ Runtime ] -----
00009 *      | | | | / / \ \ \ / | | \ |
00010 *      | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #include <stdio.h>
00022 #include <stdlib.h>
00023 #include <string.h>
00024 #include <sys/stat.h>
00025 #include <sys/fcntl.h>
00026
00027 #include "emu_config.h"
00028
00029 #include "avr_cpu.h"
00030 #include "intel_hex.h"
00031
00032 #include "elf_types.h"
00033 #include "elf_process.h"
00034 #include "elf_print.h"
00035
00036 #include "debug_sym.h"
00037
00038 //-----
00039 static void AVR_Copy_Record( HEX_Record_t *pstHex_)
00040 {
00041     uint16_t u16Data;
00042     uint16_t i;
00043     for (i = 0; i < pstHex_->u8ByteCount; i += 2)
00044     {
00045         u16Data = pstHex_->u8Data[i+1];
00046         u16Data <= 8;
00047         u16Data |= pstHex_->u8Data[i];
00048
00049         stCPU.pu16ROM[(pstHex_->u16Address + i) >> 1] = u16Data;
00050     }
00051 }
00052
00053 //-----
00054 bool AVR_Load_HEX( const char *szFilePath_)
00055 {
00056     HEX_Record_t stRecord;
00057     uint32_t u32Addr = 0;
00058     int fd = -1;
00059
00060     if (!szFilePath_)
00061     {
00062         fprintf(stderr, "No programming file specified\n");
00063         return false;
00064     }
00065
00066     fd = open(szFilePath_, O_RDONLY);
00067
00068     if (-1 == fd)
00069     {
00070         fprintf(stderr, "Unable to open file\n");
00071         return false;
00072     }
00073
00074     bool rc = true;
00075
00076     while (rc)
00077     {
00078         rc = HEX_Read_Record(fd, &stRecord);
00079         if (RECORD_EOF == stRecord.u8RecordType)
00080         {
00081             break;
00082         }
00083         if (RECORD_DATA == stRecord.u8RecordType)
00084         {
00085             AVR_Copy_Record(&stRecord);
00086         }
00087     }
00088
00089 cleanup:
00090     close(fd);

```

```

00091     return rc;
00092 }
00093
00094 //-----
00095 static void AVR_Load_ELF_Symbols( const uint8_t *pau8Buffer_ )
00096 {
00097     // Get a pointer to the section header for the symbol table
00098     uint32_t u32Offset = ELF_GetSymbolTableOffset( pau8Buffer_ );
00099     if (u32Offset == 0)
00100     {
00101         printf( "No debug symbol, bailing\n");
00102         return;
00103     }
00104     ElfSectionHeader_t *pstSymHeader = (ElfSectionHeader_t*)(&pau8Buffer_[u32Offset]);
00105
00106     // Get a pointer to the section header for the symbol table's strings
00107     u32Offset = ELF_GetSymbolStringTableOffset( pau8Buffer_ );
00108     if (u32Offset == 0)
00109     {
00110         printf( "No debug symbol strings, bailing\n");
00111         return;
00112     }
00113     ElfSectionHeader_t *pstStrHeader = (ElfSectionHeader_t*)(&pau8Buffer_[u32Offset]);
00114
00115     // Iterate through the symbol table section, printing out the details of each.
00116     uint32_t u32SymOffset = pstSymHeader->u32Offset;
00117     ElfSymbol_t *pstSymbol = (ElfSymbol_t*)(&pau8Buffer_[u32SymOffset]);
00118
00119     while (u32SymOffset < (pstSymHeader->u32Offset + pstSymHeader->u32Size))
00120     {
00121         uint8_t u8Type = pstSymbol->u8Info & 0x0F;
00122         if (u8Type == 2)
00123         {
00124             // Note that elf file uses byte addressing, and we use 16-bit word addressing
00125             Symbol_Add_Func( &pau8Buffer_[pstSymbol->u32Name + pstStrHeader->u32Offset],
00126                             pstSymbol->u32Value >> 1,
00127                             pstSymbol->u32Size >> 1);
00128         }
00129         else if (u8Type == 1)
00130         {
00131             // The elf files use 0x0080XXXX as an offset for dat objects. Mask here
00132             Symbol_Add_Obj( &pau8Buffer_[pstSymbol->u32Name + pstStrHeader->u32Offset],
00133                             pstSymbol->u32Value & 0x0000FFFF,
00134                             pstSymbol->u32Size );
00135         }
00136         u32SymOffset += pstSymHeader->u32EntrySize;
00137         pstSymbol = (ElfSymbol_t*)(&pau8Buffer_[u32SymOffset]);
00138     }
00139 }
00140
00141 //-----
00142 bool AVR_Load_ELF( const char *szFilePath_ )
00143 {
00144     uint8_t *pu8Buffer;
00145
00146     // Load the ELF Binary from into a newly-created local buffer
00147     if (0 != ELF_LoadFromFile(&pu8Buffer, szFilePath_))
00148     {
00149         return false;
00150     }
00151
00152     // Loaded ELF successfully, load program sections into AVR memory.
00153     ElfHeader_t *pstHeader = (ElfHeader_t*)(pu8Buffer);
00154     uint32_t u32Offset = pstHeader->u32PHOffset;
00155     uint32_t u32MaxOffset = pstHeader->u32PHOffset
00156                             + (pstHeader->u16PHNum * pstHeader->u16PHSize);
00157
00158     // Iterate through every program header section in the elf-file
00159     while (u32Offset < u32MaxOffset)
00160     {
00161         ElfProgramHeader_t *pstPHeader = (ElfProgramHeader_t*)(&pu8Buffer[u32Offset]);
00162
00163         // RAM encoded in ELF file using addresses >= 0x00800000
00164         if (pstPHeader->u32PhysicalAddress >= 0x00800000)
00165         {
00166             // Clear range in segment
00167             memset( &(stCPU.pstRAM->au8RAM[pstPHeader->u32PhysicalAddress & 0x0000FFFF]),
00168                     0,
00169                     pstPHeader->u32MemSize );
00170             // Copy program segment from ELF into CPU RAM
00171             memcpy( &(stCPU.pstRAM->au8RAM[pstPHeader->u32PhysicalAddress & 0x0000FFFF]),
00172                     &pu8Buffer[pstPHeader->u32Offset],
00173                     pstPHeader->u32FileSize );
00174         }
00175         else
00176         {
00177             // Clear range in segment

```

```

00178         memset( &(stCPU.pu16ROM[pstPHeader->u32PhysicalAddress >> 1]),
00179                 0,
00180                 pstPHeader->u32MemSize );
00181
00182         // Copy program segment from ELF into CPU Flash
00183         memcpy( &(stCPU.pu16ROM[pstPHeader->u32PhysicalAddress >> 1]),
00184                &pu8Buffer[pstPHeader->u32Offset],
00185                pstPHeader->u32FileSize );
00186     }
00187
00188     // Next Section...
00189     u32Offset += pstHeader->u16PHSize;
00190 }
00191
00192 AVR_Load_ELF_Symbols( pu8Buffer );
00193
00194 free( pu8Buffer );
00195 return true;
00196 }

```

## 4.121 src/loader/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.*

### 4.121.1 Detailed Description

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

Definition in file [avr\\_loader.h](#).

### 4.121.2 Function Documentation

#### 4.121.2.1 AVR\_Load\_ELF()

```

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

<i>szFile</i> ↔ <i>Path_</i>	Pointer to the elf-file path
---------------------------------	------------------------------

## Returns

true if the elf file load operation succes

Definition at line 142 of file [avr\\_loader.c](#).

## 4.121.2.2 AVR\_Load\_HEX()

```
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

<i>szFile</i> ↔ <i>Path_</i>	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](#).

## 4.122 avr\_loader.h

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ( ) / ( ) / (      \      ( ) / (      | -- [ Funkenstein ] -----
00005 *      / ( ) / ( ) / ( ( ( ( ) \      ) \      / ( )      | -- [ Little ] -----
00006 *      ( ) _ ( )      ) \ _      ) \ ( ( ( ( ) ( )      | -- [ AVR ] -----
00007 *      | _ | |      ( ) _ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \      \ \ / / | _ \      | -- [ Runtime ] -----
00009 *      | _ | | _ _      / _ \      \ \ / / | _ \      |
00010 *      | _ | | _ _ _      / _ \      \ \ / / | _ \      | "Yeah, it does Arduino..."
00011 *      +-----+
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __AVR_LOADER_H_
00022 #define __AVR_LOADER_H_
00023
00024 #include <stdint.h>
00025 #include "avr_cpu.h"
00026
00027 //-----
00035 bool AVR_Load_HEX( const char *szFilePath_);
00036
00037 //-----
00046 bool AVR_Load_ELF( const char *szFilePath_);
00047 #endif
```

## 4.123 src/loader/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.*

### 4.123.1 Detailed Description

Functions used to process ELF Binaries.

Definition in file [elf\\_process.c](#).

### 4.123.2 Function Documentation

#### 4.123.2.1 [ELF\\_GetHeaderStringTableOffset\(\)](#)

```
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

<i>pau8↵ Buffer_</i>	- 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](#).

## 4.123.2.2 ELF\_GetSymbolStringTableOffset()

```
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

<i>pau8↵ Buffer_</i>	- 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](#).

## 4.123.2.3 ELF\_GetSymbolTableOffset()

```
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

<i>pau8↵ Buffer_</i>	- 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](#).

**4.123.2.4 ELF\_LoadFromFile()**

```
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**

<i>ppau8Buffer_</i>	- Byte-array pointer, which will point to a newly-allocated buffer on successful read (or NULL) on error.
<i>szPath_</i>	- File path to load

**Returns**

0 on success, -1 on error.

Definition at line 104 of file [elf\\_process.c](#).

**4.124 elf\_process.c**

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ )  )\ )      (      |
00004 *      ((/( ((/(      \      | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( ) \   | -- [ Little ] -----
00006 *      ( ) _ ( )      ) \ _ \ (( ( ( ) ( )      | -- [ AVR ] -----
00007 *      | _ | |      ( ) _ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ / / | _ \      | -- [ Runtime ] -----
00009 *      | _ | | _ _ / _ \ \ \ / / | _ \      |
00010 *      | _ | | _ _ / _ \ \ \ / / | _ \      | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #include "elf_process.h"
00022 #include "elf_types.h"
00023
00024 #include <stdint.h>
00025 #include <stdio.h>
00026 #include <stdlib.h>
00027 #include <sys/types.h>
00028 #include <sys/stat.h>
00029 #include <unistd.h>
00030
00031 #define DEBUG_PRINT(...)
00032
00033 //-----
00034 uint32_t ELF_GetHeaderStringTableOffset( const uint8_t *pau8Buffer_ )
00035 {
```



```

00036     ElfHeader_t *pstHeader = (ElfHeader_t*)pau8Buffer_;
00037
00038     ElfSectionHeader_t *pstStringTable =
00039         (ElfSectionHeader_t*)(&pau8Buffer_[pstHeader->u32SHOffset + (pstHeader->u16SHSize * pstHeader->
00040             u16SHIndex)]);
00041     return pstStringTable->u32Offset;
00042 }
00043
00044 //-----
00045 uint32_t ELF_GetSymbolStringTableOffset( const uint8_t *pau8Buffer_ )
00046 {
00047     uint32_t u32Offset;
00048     uint16_t u16SHCount;
00049
00050     ElfHeader_t *pstHeader = (ElfHeader_t*)pau8Buffer_;
00051     uint32_t u32StringOffset = ELF_GetHeaderStringTableOffset( pau8Buffer_ );
00052
00053     u32Offset = pstHeader->u32SHOffset;
00054     u16SHCount = pstHeader->u16SHNum;
00055
00056     while (u16SHCount)
00057     {
00058         ElfSectionHeader_t *pstSHeader = (ElfSectionHeader_t*)(&pau8Buffer_[u32Offset]);
00059         if (
00060             (ELF_SECTION_TYPE_STRTAB == pstSHeader->u32Type) &&
00061             (0 == strcmp( ".strtab", &pau8Buffer_[u32StringOffset + pstSHeader->u32Name]))
00062         )
00063         {
00064             return u32Offset;
00065         }
00066     }
00067     //--
00068     u16SHCount--;
00069     u32Offset += pstHeader->u16SHSize;
00070 }
00071
00072 return 0;
00073 }
00074
00075 //-----
00076 uint32_t ELF_GetSymbolTableOffset( const uint8_t *pau8Buffer_ )
00077 {
00078     uint32_t u32Offset;
00079     uint16_t u16SHCount;
00080
00081     ElfHeader_t *pstHeader = (ElfHeader_t*)pau8Buffer_;
00082
00083     u32Offset = pstHeader->u32SHOffset;
00084     u16SHCount = pstHeader->u16SHNum;
00085
00086     while (u16SHCount)
00087     {
00088         ElfSectionHeader_t *pstSHeader = (ElfSectionHeader_t*)(&pau8Buffer_[u32Offset]);
00089         if (ELF_SECTION_TYPE_SYMTAB == pstSHeader->u32Type)
00090         {
00091             return u32Offset;
00092         }
00093     }
00094     //--
00095     u16SHCount--;
00096     u32Offset += pstHeader->u16SHSize;
00097 }
00098
00099 return 0;
00100 }
00101
00102 //-----
00103 int ELF_LoadFromFile( uint8_t **ppau8Buffer_, const char *szPath_ )
00104 {
00105     size_t file_size;
00106     FILE *my_file;
00107
00108     my_file = fopen( szPath_, "rb" );
00109     if (NULL == my_file)
00110     {
00111         DEBUG_PRINT( "Unable to read file @ %s\n", szPath_ );
00112         return -1;
00113     }
00114     fseek(my_file, 0, SEEK_END);
00115     file_size = ftell(my_file);
00116     fseek(my_file, 0, SEEK_SET);
00117
00118     uint8_t *bufptr = (uint8_t*)malloc(file_size);
00119     *ppau8Buffer_ = bufptr;
00120
00121

```

```

00122     if (!bufptr)
00123     {
00124         DEBUG_PRINT( "Unable to malloc elf file buffer\n" );
00125         fclose( my_file );
00126         return -1;
00127     }
00128
00129     size_t bytes_read = 0;
00130     while (bytes_read < file_size)
00131     {
00132         size_t iter_read = fread( bufptr, 1, 1, my_file );
00133         if( iter_read == 0 )
00134         {
00135             DEBUG_PRINT( "%d read total\n", bytes_read );
00136             break;
00137         }
00138         bytes_read += iter_read;
00139         bufptr += iter_read;
00140     }
00141
00142     DEBUG_PRINT( "Success reading %d bytes\n", file_size );
00143     fclose( my_file );
00144     return 0;
00145 }

```

## 4.125 src/loader/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.*

### 4.125.1 Detailed Description

Functions used to process ELF Binaries.

Definition in file [elf\\_process.h](#).

### 4.125.2 Function Documentation

#### 4.125.2.1 ELF\_GetHeaderStringTableOffset()

```

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

<i>pau8↔ Buffer_</i>	- 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](#).

## 4.125.2.2 ELF\_GetSymbolStringTableOffset()

```
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

<i>pau8↔ Buffer_</i>	- 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](#).

## 4.125.2.3 ELF\_GetSymbolTableOffset()

```
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

<i>pau8↔ Buffer_</i>	- 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](#).

**4.125.2.4 ELF\_LoadFromFile()**

```
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**

<i>ppau8Buffer_</i>	- Byte-array pointer, which will point to a newly-allocated buffer on successful read (or NULL) on error.
<i>szPath_</i>	- File path to load

**Returns**

0 on success, -1 on error.

Definition at line 104 of file [elf\\_process.c](#).

**4.126 elf\_process.h**

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ )  )\ )  (      )\ )  |
00004 *      ((/ ( ((/ (      )\      ( ( ((/ (      | -- [ Funkenstein ] -----
00005 *      /( )  /( )  ((( ( )\ )\  /( )      | -- [ Little ] -----
00006 *      ( ) _ ( )      ) _ )\ ( ( ) ( )      | -- [ AVR ] -----
00007 *      | _ | |      ( ) _ ( ) \ \ / / | _ \  | -- [ Virtual ] -----
00008 *      | _ | | _ _ / _ \ \ \ / / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _ _ / _ \ \ \ / / | _ \      |
00010 *                                          | "Yeah, it does Arduino..."
00011 * -----|-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __ELF_PROCESS_H__
00022 #define __ELF_PROCESS_H__
00023
00024 #include "elf_types.h"
00025 #include <stdint.h>
00026
00027 //-----
00037 uint32_t ELF_GetHeaderStringTableOffset( const uint8_t *pau8Buffer_ );
00038
00039 //-----
00049 uint32_t ELF_GetSymbolStringTableOffset( const uint8_t *pau8Buffer_ );
00050
00051 //-----
00061 uint32_t ELF_GetSymbolTableOffset( const uint8_t *pau8Buffer_ );
00062
00063 //-----
00075 int ELF_LoadFromFile( uint8_t **ppau8Buffer_, const char *szPath_ );
00076
00077 #endif //__ELF_PROCESS_H__
```

## 4.127 src/loader/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.*

### 4.127.1 Detailed Description

Module for decoding Intel hex formatted programming files.

Definition in file [intel\\_hex.c](#).

### 4.127.2 Function Documentation

#### 4.127.2.1 HEX\_Print\_Record()

```
void HEX_Print_Record (
    HEX\_Record\_t * stRecord_ )
```

*HEX\_Print\_Record.*

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

## Parameters

<i>stRecord</i>	Pointer to a valid, initialized hex record
-----------------	--

Definition at line 33 of file [intel\\_hex.c](#).

## 4.127.2.2 HEX\_Read\_Record()

```
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

<i>fd_</i>	[in] Open file handle corresponding to the hex file
<i>stRecord</i>	[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](#).

## 4.128 intel\_hex.c

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/ ( ((/ (      \      ( ((/ (      | -- [ Funkenstein ] -----
00005 *      / ( ) / ( ) ((( ( ) \      / ( )      | -- [ Little ] -----
00006 *      ( ) _ ( )      ) \ _ \      ( ( ) ( ( )      | -- [ AVR ] -----
00007 *      | _ | |      ( ) _ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ / / | _ \      | -- [ Runtime ] -----
00009 *      | _ | | _ _      / _ \ \ \ / / | _ \      |
00010 *      | _ | | _ _ _      / _ \ \ \ / / | _ \      | "Yeah, it does Arduino..."
00011 *      | _ | | _ _ _ _      / _ \ \ \ / / | _ \      |
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #include <stdio.h>
00022 #include <stdlib.h>
00023 #include <string.h>
00024 #include <stdint.h>
00025 #include <sys/stat.h>
00026 #include <sys/fcntl.h>
00027
00028 #include "emu_config.h"
00029
```

```

00030 #include "intel_hex.h"
00031
00032 //-----
00033 void HEX_Print_Record( HEX_Record_t *stRecord_ )
00034 {
00035     printf( "Line: %d\n"
00036            "ByteCount: %d\n"
00037            "RecordType: %d\n"
00038            "Address: %X\n"
00039            "Data:",
00040            stRecord_>u32Line,
00041            stRecord_>u8ByteCount,
00042            stRecord_>u8RecordType,
00043            stRecord_>u16Address );
00044     int i;
00045     for (i = 0; i < stRecord_>u8ByteCount; i++)
00046     {
00047         printf( " %02X", stRecord_>u8Data[i]);
00048     }
00049     printf( "\n" );
00050 }
00051
00052 //-----
00053 static bool HEX_Read_Header( int fd_ )
00054 {
00055     ssize_t bytes_read;
00056     char acBuf[2] = {0};
00057
00058     bytes_read = read(fd_, acBuf, 1);
00059     if (1 != bytes_read)
00060     {
00061         return false;
00062     }
00063     if (':' == acBuf[0])
00064     {
00065         return true;
00066     }
00067     return false;
00068 }
00069
00070 //-----
00071 static bool HEX_Next_Line( int fd_, HEX_Record_t *stRecord_ )
00072 {
00073     ssize_t bytes_read;
00074     char acBuf[2] = {0};
00075
00076     stRecord_>u32Line++;
00077     do
00078     {
00079         bytes_read = read(fd_, acBuf, 1);
00080         if (1 != bytes_read)
00081         {
00082             return false;
00083         }
00084     } while(acBuf[0] != '\n');
00085     return true;
00086 }
00087
00088 //-----
00089 static bool HEX_Read_Record_Type( int fd_, HEX_Record_t *stRecord_ )
00090 {
00091     ssize_t bytes_read;
00092     uint32_t u32Hex;
00093     char acBuf[3] = {0};
00094
00095     bytes_read = read(fd_, acBuf, 2);
00096     if (2 != bytes_read)
00097     {
00098         return false;
00099     }
00100     sscanf(acBuf, "%02X", &u32Hex);
00101     stRecord_>u8RecordType = (uint8_t)u32Hex;
00102
00103     if (stRecord_>u8RecordType >= RECORD_TYPE_MAX)
00104     {
00105         return false;
00106     }
00107     return true;
00108 }
00109
00110 //-----
00111 static bool HEX_Read_Byte_Count( int fd_, HEX_Record_t *stRecord_ )
00112 {
00113     ssize_t bytes_read;
00114     uint32_t u32Hex;

```

```

00117     char acBuf[3] = {0};
00118
00119     bytes_read = read(fd_, acBuf, 2);
00120     if (2 != bytes_read)
00121     {
00122         return false;
00123     }
00124     sscanf(acBuf, "%02X", &u32Hex);
00125     stRecord->u8ByteCount = (uint8_t)u32Hex;
00126
00127     return true;
00128 }
00129
00130 //-----
00131 static bool HEX_Read_Address( int fd_, HEX_Record_t *stRecord_ )
00132 {
00133     ssize_t bytes_read;
00134     uint32_t u32Hex;
00135     char acBuf[5] = {0};
00136
00137     bytes_read = read(fd_, acBuf, 4);
00138     if (4 != bytes_read)
00139     {
00140         return false;
00141     }
00142     sscanf(acBuf, "%04X", &u32Hex);
00143     stRecord->u16Address = (uint16_t)u32Hex;
00144
00145     return true;
00146 }
00147
00148 //-----
00149 static bool HEX_Read_Data( int fd_, HEX_Record_t *stRecord_ )
00150 {
00151     ssize_t bytes_read;
00152     uint32_t u32Hex;
00153     char acBuf[MAX_HEX_DATA_BYTES * 2] = {0};
00154
00155     int i;
00156     for (i = 0; i < stRecord->u8ByteCount; i++)
00157     {
00158         // printf("i:%d\n", i);
00159         bytes_read = read(fd_, acBuf, 2);
00160         if (2 != bytes_read)
00161         {
00162             return false;
00163         }
00164         sscanf(acBuf, "%02X", &u32Hex);
00165         stRecord->u8Data[i] = (uint8_t)u32Hex;
00166     }
00167
00168     return true;
00169 }
00170
00171 //-----
00172 static bool HEX_Read_Checksum( int fd_, HEX_Record_t *stRecord_ )
00173 {
00174     ssize_t bytes_read;
00175     uint32_t u32Hex;
00176     char acBuf[3] = {0,0,0};
00177
00178     bytes_read = read(fd_, acBuf, 2);
00179     if (2 != bytes_read)
00180     {
00181         return false;
00182     }
00183     sscanf(acBuf, "%02X", &u32Hex);
00184     stRecord->u8Checksum = (uint8_t)u32Hex;
00185
00186     return true;
00187 }
00188
00189 //-----
00190 static bool HEX_Line_Validate( HEX_Record_t *stRecord_ )
00191 {
00192     // Calculate the CRC for the fields in the struct and compare
00193     // against the value read from file...
00194     uint8_t u8CRC = 0;
00195     u8CRC += (uint8_t)(stRecord->u16Address >> 8);
00196     u8CRC += (uint8_t)(stRecord->u16Address & 0x00FF);
00197     u8CRC += stRecord->u8ByteCount;
00198     u8CRC += stRecord->u8RecordType;
00199
00200     uint8_t i;
00201     for (i = 0; i < stRecord->u8ByteCount; i++)
00202     {
00203         u8CRC += stRecord->u8Data[i];

```



```

00204     }
00205
00206     u8CRC = (~u8CRC) + 1;    // Spec says to take the 2's complement
00207     if (u8CRC != stRecord->u8Checksum)
00208     {
00209         return false;
00210     }
00211
00212     return true;
00213 }
00214
00215 //-----
00216 bool HEX_Read_Record( int fd_, HEX_Record_t *stRecord_ )
00217 {
00218     bool rc = true;
00219     if (rc)
00220     {
00221         rc = HEX_Read_Header(fd_);
00222     }
00223     if (rc)
00224     {
00225         rc = HEX_Read_Byte_Count(fd_, stRecord_);
00226     }
00227     if (rc)
00228     {
00229         rc = HEX_Read_Address(fd_, stRecord_);
00230     }
00231     if (rc)
00232     {
00233         rc = HEX_Read_Record_Type(fd_, stRecord_);
00234     }
00235     if (rc)
00236     {
00237         rc = HEX_Read_Data(fd_, stRecord_);
00238     }
00239     if (rc)
00240     {
00241         rc = HEX_Read_Checksum(fd_, stRecord_);
00242     }
00243     if (rc)
00244     {
00245         rc = HEX_Line_Validate(stRecord_);
00246     }
00247
00248     HEX_Next_Line(fd_, stRecord_);
00249     return rc;
00250 }

```

## 4.129 src/loader/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.*

### 4.129.1 Detailed Description

Module for decoding Intel hex formatted programming files.

Definition in file [intel\\_hex.h](#).

### 4.129.2 Function Documentation

#### 4.129.2.1 HEX\_Print\_Record()

```
void HEX_Print_Record (  
    HEX\_Record\_t * stRecord_ )
```

*HEX\_Print\_Record.*

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

#### Parameters

<i>stRecord_</i>	Pointer to a valid, initialized hex record
—	

Definition at line 33 of file [intel\\_hex.c](#).

#### 4.129.2.2 HEX\_Read\_Record()

```
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

<i>fd</i>	[in] Open file handle corresponding to the hex file
<i>st</i> <i>Record</i>	[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](#).

## 4.130 intel\_hex.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( ()/(      \      | -- [ Funkenstein ] -----
00005 *      /( ) )/( ) )((( ( ) \      \      / ( )      | -- [ Little ] -----
00006 *      ( ) _ | ( )      ) \ _      \ ( ) ( ( ) ( )      | -- [ AVR ] -----
00007 *      | | _ | |      ( ) _ ( ) \ \ / / | _      \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \      \ V / | _      /      | -- [ Runtime ] -----
00009 *      | _ | | _      / _ \      \ V / | _      /      |
00010 *      | _ | | _      / _ \      \ V / | _      /      | "Yeah, it does Arduino..."
00011 *      | _ | | _      / _ \      \ V / | _      /      |
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __INTEL_HEX_H__
00022 #define __INTEL_HEX_H__
00023
00024 #include <stdint.h>
00025 #include <stdbool.h>
00026
00027 //-----
00028 // Load a hex file into the ROM section of a virtual AVR.
00029 #define MAX_HEX_DATA_BYTES      (255) // max data bytes per line in a record
00030
00031 //-----
00032 // Record types in the HEX specification
00033 #define RECORD_DATA      (0)
00034 #define RECORD_EOF      (1)
00035 #define RECORD_EXTENDED_SEGMENT      (2)
00036 #define RECORD_START_SEGMENT      (3)
00037 #define RECORD_EXTENDED_LINEAR      (4)
00038 #define RECORD_START_LINEAR      (5)
00039
00040 //-----
00041 #define RECORD_TYPE_MAX      (5)
00042
00043 //-----
00044 // For reference, this is the line format for an intel hex record.
00045 // :WWXXYYYYzz.....zzCC
00046 // Where : = the ":" start code
00047 // WW = the byte count in the data field
00048 // XX = the record type
00049 // YYYY = record address
00050 // zz = data bytes
00051 // CC = 2's complement checksum of all fields, excluding start code and checksum
00052
00053 //-----
00057 typedef struct
00058 {
00059     uint8_t      u8ByteCount;
00060     uint8_t      u8RecordType;
00061     uint16_t      u16Address;
00062     uint8_t      u8Data[MAX_HEX_DATA_BYTES];
00063     uint8_t      u8Checksum;
00064     uint32_t      u32Line;
00065 } HEX_Record_t;
00066
00067 //-----

```

```

00075 void HEX_Print_Record( HEX_Record_t *stRecord_ );
00076
00077 //-----
00090 bool HEX_Read_Record( int fd_, HEX_Record_t *stRecord_ );
00091
00092 #endif

```

## 4.131 src/peripheral/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**

### 4.131.1 Detailed Description

Interfaces for creating AVR peripheral plugins.

Definition in file [avr\\_peripheral.h](#).

## 4.132 avr\_peripheral.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( (/( ( \      ( ( (/( | -- [ Funkenstein ] -----
00005 *      /( ) )/( ) ((( ( ) \ ) \ /( ) | -- [ Little ] -----
00006 *      ( ) _ ( ) ) \ _ \ ( ( ( ( ) | -- [ AVR ] -----
00007 *      | _ | | ( ) _ ( ) \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ / _ \ \ V / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ / _ \ \ V / | _ \ |
00010 *      | "Yeah, it does Arduino..."
00011 *      -----
00012 *      (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 *      See license.txt for details
00014 *      *****/
00021 #ifndef __AVR_PERIPHERAL_H__
00022 #define __AVR_PERIPHERAL_H__
00023
00024 #include <stdint.h>
00025
00026 //-----
00027 // Peripheral callout functions - used to implement arbitrary peripherals
00028 // which are able to intercept/react to read/write operations to specific

```

```

00029 // I/O addresses.
00030 //-----
00031
00032 typedef void (*PeriphInit) (void *context_ );
00033 typedef void (*PeriphRead) (void *context_, uint8_t ucAddr_, uint8_t *pucValue_ );
00034 typedef void (*PeriphWrite)(void *context_, uint8_t ucAddr_, uint8_t ucValue_ );
00035 typedef void (*PeriphClock)(void *context_ );
00036
00037 //-----
00038 typedef void (*InterruptAck)( uint8_t ucVector_);
00039
00040 //-----
00041 typedef struct AVRPeripheral
00042 {
00043     PeriphInit          pfInit;
00044     PeriphRead          pfRead;
00045     PeriphWrite         pfWrite;
00046     PeriphClock         pfClock;
00047
00048     void                *pvContext;
00049
00050     uint8_t             u8AddrStart;
00051     uint8_t             u8AddrEnd;
00052 } AVRPeripheral;
00053
00054 #endif /*__AVR_PERIPHERAL_H__

```

## 4.133 src/peripheral/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\_ADCSR**
- **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

#### 4.133.1 Detailed Description

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

Definition in file [avr\\_periphregs.h](#).

#### 4.134 avr\_periphregs.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( (/(      (      | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( ) \ ) \ / ( ) ) | -- [ Little ] -----
00006 *      ( ) _ ( )      ) \ _ ) \ ( ( ( ( ) | -- [ AVR ] -----
00007 *      | _ | | |      ( ) _ ( ) \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ |      / _ \ \ \ V / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ | / _ \ \ \ \ / | _ \ |
00010 *      | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00022 #ifndef __AVR_PERIPHREGS_H__
00023 #define __AVR_PERIPHREGS_H__

```

```

00024
00025 #include <stdint.h>
00026
00027 //-----
00028 // UART/USART register struct definitions.
00029 //-----
00030 typedef struct __attribute__((packed))
00031 {
00032     union __attribute__((packed))
00033     {
00034         uint8_t r;
00035         struct __attribute__((packed))
00036         {
00037             unsigned int MPCM0 : 1;
00038             unsigned int U2X0 : 1;
00039             unsigned int UPE0 : 1;
00040             unsigned int DOR0 : 1;
00041             unsigned int FE0 : 1;
00042             unsigned int UDRE0 : 1;
00043             unsigned int TXC0 : 1;
00044             unsigned int RXC0 : 1;
00045         };
00046     };
00047 } AVR_UCSR0A;
00048
00049 //-----
00050 typedef struct __attribute__((packed))
00051 {
00052     union __attribute__((packed))
00053     {
00054         uint8_t r;
00055         struct __attribute__((packed))
00056         {
00057             unsigned int TXB80 : 1;
00058             unsigned int RXB80 : 1;
00059             unsigned int UCSZ02 : 1;
00060             unsigned int TXEN0 : 1;
00061             unsigned int RXEN0 : 1;
00062             unsigned int UDRIE0 : 1;
00063             unsigned int TXCIE0 : 1;
00064             unsigned int RXCIE0 : 1;
00065         };
00066     };
00067 } AVR_UCSR0B;
00068
00069 //-----
00070 typedef struct __attribute__((packed))
00071 {
00072     union __attribute__((packed))
00073     {
00074         uint8_t r;
00075         struct __attribute__((packed))
00076         {
00077             unsigned int UCPOL0 : 1;
00078             unsigned int UCPHA0 : 1;
00079             unsigned int UDORD0 : 1;
00080             unsigned int USBS0 : 1;
00081             unsigned int UPM00 : 1;
00082             unsigned int UPM01 : 1;
00083             unsigned int UMSEL00 : 1;
00084             unsigned int UMSEL01 : 1;
00085         };
00086     };
00087 } AVR_UCSR0C;
00088
00089 //-----
00090 // TWI interface register struct definitions
00091 //-----
00092 typedef struct __attribute__((packed))
00093 {
00094     union __attribute__((packed))
00095     {
00096         uint8_t r;
00097         struct __attribute__((packed))
00098         {
00099             unsigned int reserved : 1;
00100             unsigned int TWAM0 : 1;
00101             unsigned int TWAM1 : 1;
00102             unsigned int TWAM2 : 1;
00103             unsigned int TWAM3 : 1;
00104             unsigned int TWAM4 : 1;
00105             unsigned int TWAM5 : 1;
00106             unsigned int TWAM6 : 1;
00107         };
00108     };
00109 } AVR_TWAMR;
00110

```

```

00111 //-----
00112 typedef struct __attribute__((packed))
00113 {
00114     union __attribute__((packed))
00115     {
00116         uint8_t r;
00117         struct __attribute__((packed))
00118         {
00119             unsigned int TWIE : 1;
00120             unsigned int reserved : 1;
00121             unsigned int TWEN : 1;
00122             unsigned int TWWC : 1;
00123             unsigned int TWSTO : 1;
00124             unsigned int TWSTA : 1;
00125             unsigned int TWEA : 1;
00126             unsigned int TWINT : 1;
00127         };
00128     };
00129 } AVR_TWCR;
00130
00131 //-----
00132 typedef struct __attribute__((packed))
00133 {
00134     union __attribute__((packed))
00135     {
00136         uint8_t r;
00137         struct __attribute__((packed))
00138         {
00139             unsigned int TWGCE : 1;
00140             unsigned int TWA0 : 1;
00141             unsigned int TWA1 : 1;
00142             unsigned int TWA2 : 1;
00143             unsigned int TWA3 : 1;
00144             unsigned int TWA4 : 1;
00145             unsigned int TWA5 : 1;
00146             unsigned int TWA6 : 1;
00147         };
00148     };
00149 } AVR_TWAR;
00150
00151 //-----
00152 typedef struct __attribute__((packed))
00153 {
00154     union __attribute__((packed))
00155     {
00156         uint8_t r;
00157         struct __attribute__((packed))
00158         {
00159             unsigned int TWPS0 : 1;
00160             unsigned int TWPS1 : 1;
00161             unsigned int reserved : 1;
00162             unsigned int TWPS3 : 1;
00163             unsigned int TWPS4 : 1;
00164             unsigned int TWPS5 : 1;
00165             unsigned int TWPS6 : 1;
00166             unsigned int TWPS7 : 1;
00167         };
00168     };
00169 } AVR_TWSR;
00170
00171 //-----
00172 // Timer 2 register struct __attribute__((packed)) definitins.
00173 //-----
00174 typedef struct __attribute__((packed))
00175 {
00176     union __attribute__((packed))
00177     {
00178         uint8_t r;
00179         struct __attribute__((packed))
00180         {
00181             unsigned int TCR2BUB : 1;
00182             unsigned int TCR2AUB : 1;
00183             unsigned int OCR2BUB : 1;
00184             unsigned int OCR2AUB : 1;
00185             unsigned int TCN2UB : 1;
00186             unsigned int AS2 : 1;
00187             unsigned int EXCLK : 1;
00188             unsigned int reserved : 1;
00189         };
00190     };
00191 } AVR_ASSR;
00192
00193 //-----
00194 typedef struct __attribute__((packed))
00195 {
00196     union __attribute__((packed))
00197     {

```



```

00198     uint8_t r;
00199     struct __attribute__((packed))
00200     {
00201         unsigned int CS20 : 1;
00202         unsigned int CS21 : 1;
00203         unsigned int CS22 : 1;
00204         unsigned int WGM22 : 1;
00205         unsigned int reserved : 2;
00206         unsigned int FOC2B : 1;
00207         unsigned int FOC2A : 1;
00208     };
00209 };
00210 } AVR_TCCR2B;
00211
00212 //-----
00213 typedef struct __attribute__((packed))
00214 {
00215     union __attribute__((packed))
00216     {
00217         uint8_t r;
00218         struct __attribute__((packed))
00219         {
00220             unsigned int WGM20 : 1;
00221             unsigned int WGM21 : 1;
00222             unsigned int reserved : 2;
00223             unsigned int COM2B0 : 1;
00224             unsigned int COM2B1 : 1;
00225             unsigned int COM2A0 : 1;
00226             unsigned int COM2A1 : 1;
00227         };
00228     };
00229 } AVR_TCCR2A;
00230
00231 //-----
00232 // Timer 1 Register struct __attribute__((packed)) definitions
00233 //-----
00234
00235 typedef struct __attribute__((packed))
00236 {
00237     union __attribute__((packed))
00238     {
00239         uint8_t r;
00240         struct __attribute__((packed))
00241         {
00242             unsigned int WGM10 : 1;
00243             unsigned int WGM11 : 1;
00244             unsigned int reserved : 2;
00245             unsigned int COM1B0 : 1;
00246             unsigned int COM1B1 : 1;
00247             unsigned int COM1A0 : 1;
00248             unsigned int COM1A1 : 1;
00249         };
00250     };
00251 } AVR_TCCR1A;
00252
00253 //-----
00254 typedef struct __attribute__((packed))
00255 {
00256     union __attribute__((packed))
00257     {
00258         uint8_t r;
00259         struct __attribute__((packed))
00260         {
00261             unsigned int CS10 : 1;
00262             unsigned int CS11 : 1;
00263             unsigned int CS12 : 1;
00264             unsigned int WGM12 : 1;
00265             unsigned int WGM13 : 1;
00266             unsigned int reserved : 1;
00267             unsigned int ICES1 : 1;
00268             unsigned int ICNC1 : 1;
00269         };
00270     };
00271 } AVR_TCCR1B;
00272
00273 //-----
00274 typedef struct __attribute__((packed))
00275 {
00276     union __attribute__((packed))
00277     {
00278         uint8_t r;
00279         struct __attribute__((packed))
00280         {
00281             unsigned int reserved : 6;
00282             unsigned int FOC1B : 1;
00283             unsigned int FOC1A : 1;
00284         };
00285     };

```

```

00285     };
00286 } AVR_TCCR1C;
00287
00288 //-----
00289 // A2D converter register definitions
00290 //-----
00291 typedef struct __attribute__((packed))
00292 {
00293     union __attribute__((packed))
00294     {
00295         uint8_t r;
00296         struct __attribute__((packed))
00297         {
00298             unsigned int AIN0D : 1;
00299             unsigned int AIN1D : 1;
00300             unsigned int reserved : 6;
00301         };
00302     };
00303 } AVR_DIDR1;
00304
00305 //-----
00306 typedef struct __attribute__((packed))
00307 {
00308     union __attribute__((packed))
00309     {
00310         uint8_t r;
00311         struct __attribute__((packed))
00312         {
00313             unsigned int ADC0D : 1;
00314             unsigned int ADC1D : 1;
00315             unsigned int ADC2D : 1;
00316             unsigned int ADC3D : 1;
00317             unsigned int ADC4D : 1;
00318             unsigned int ADC5D : 1;
00319             unsigned int reserved : 2;
00320         };
00321     };
00322 } AVR_DIDR0;
00323
00324 //-----
00325 typedef struct __attribute__((packed))
00326 {
00327     union __attribute__((packed))
00328     {
00329         uint8_t r;
00330         struct __attribute__((packed))
00331         {
00332             unsigned int MUX0 : 1;
00333             unsigned int MUX1 : 1;
00334             unsigned int MUX2 : 1;
00335             unsigned int MUX3 : 1;
00336             unsigned int reserved : 1;
00337             unsigned int ADLAR : 1;
00338             unsigned int REFS0 : 1;
00339             unsigned int REFS1 : 1;
00340         };
00341     };
00342 } AVR_ADMUX;
00343
00344 //-----
00345 typedef struct __attribute__((packed))
00346 {
00347     union __attribute__((packed))
00348     {
00349         uint8_t r;
00350         struct __attribute__((packed))
00351         {
00352             unsigned int ADPS0 : 1;
00353             unsigned int ADPS1 : 1;
00354             unsigned int ADPS2 : 1;
00355             unsigned int ADIE : 1;
00356             unsigned int ADIF : 1;
00357             unsigned int ADATE : 1;
00358             unsigned int ADSC : 1;
00359             unsigned int ADEN : 1;
00360         };
00361     };
00362 } AVR_ADCSRA;
00363
00364 //-----
00365 typedef struct __attribute__((packed))
00366 {
00367     union __attribute__((packed))
00368     {
00369         uint8_t r;
00370         struct __attribute__((packed))
00371         {

```

```

00372         unsigned int ADTS0      : 1;
00373         unsigned int ADTS1      : 1;
00374         unsigned int ADTS2      : 1;
00375         unsigned int reserved   : 3;
00376         unsigned int ACMD       : 1;
00377         unsigned int reserved_  : 1;
00378     };
00379 };
00380 } AVR_ADCSRB;
00381
00382 //-----
00383 // Timer interrupt mask registers.
00384 //-----
00385 typedef struct __attribute__((__packed__))
00386 {
00387     union __attribute__((__packed__))
00388     {
00389         uint8_t r;
00390         struct __attribute__((__packed__))
00391         {
00392             unsigned int TOIE2      : 1;
00393             unsigned int OCIE2A     : 1;
00394             unsigned int OCIE2B     : 1;
00395             unsigned int reserved   : 5;
00396         };
00397     };
00398 } AVR_TIMSK2;
00399
00400 //-----
00401 typedef struct __attribute__((__packed__))
00402 {
00403     union __attribute__((__packed__))
00404     {
00405         uint8_t r;
00406         struct __attribute__((__packed__))
00407         {
00408             unsigned int TOIE1      : 1;
00409             unsigned int OCIE1A     : 1;
00410             unsigned int OCIE1B     : 1;
00411             unsigned int reserved   : 2;
00412             unsigned int ICIE1      : 1;
00413             unsigned int reserved_  : 2;
00414         };
00415     };
00416 } AVR_TIMSK1;
00417
00418 //-----
00419 typedef struct __attribute__((__packed__))
00420 {
00421     union __attribute__((__packed__))
00422     {
00423         uint8_t r;
00424         struct __attribute__((__packed__))
00425         {
00426             unsigned int TOIE0      : 1;
00427             unsigned int OCIE0A     : 1;
00428             unsigned int OCIE0B     : 1;
00429             unsigned int reserved   : 5;
00430         };
00431     };
00432 } AVR_TIMSK0;
00433
00434 //-----
00435 // Pin change interrupt mask bit definitions
00436 //-----
00437 typedef struct __attribute__((__packed__))
00438 {
00439     union __attribute__((__packed__))
00440     {
00441         uint8_t r;
00442         struct __attribute__((__packed__))
00443         {
00444             unsigned int PCINT16 : 1;
00445             unsigned int PCINT17 : 1;
00446             unsigned int PCINT18 : 1;
00447             unsigned int PCINT19 : 1;
00448             unsigned int PCINT20 : 1;
00449             unsigned int PCINT21 : 1;
00450             unsigned int PCINT22 : 1;
00451             unsigned int PCINT23 : 1;
00452         };
00453     };
00454 } AVR_PCMSK2;
00455
00456 //-----
00457 typedef struct __attribute__((__packed__))
00458 {

```

```

00459     union __attribute__((packed))
00460     {
00461         uint8_t r;
00462         struct __attribute__((packed))
00463         {
00464             unsigned int PCINT8 : 1;
00465             unsigned int PCINT9 : 1;
00466             unsigned int PCINT10 : 1;
00467             unsigned int PCINT11 : 1;
00468             unsigned int PCINT12 : 1;
00469             unsigned int PCINT13 : 1;
00470             unsigned int PCINT14 : 1;
00471             unsigned int PCINT15 : 1;
00472         };
00473     };
00474 } AVR_PCMSK1;
00475
00476 //-----
00477 typedef struct __attribute__((packed))
00478 {
00479     union __attribute__((packed))
00480     {
00481         uint8_t r;
00482         struct __attribute__((packed))
00483         {
00484             unsigned int PCINT0 : 1;
00485             unsigned int PCINT1 : 1;
00486             unsigned int PCINT2 : 1;
00487             unsigned int PCINT3 : 1;
00488             unsigned int PCINT4 : 1;
00489             unsigned int PCINT5 : 1;
00490             unsigned int PCINT6 : 1;
00491             unsigned int PCINT7 : 1;
00492         };
00493     };
00494 } AVR_PCMSK0;
00495
00496 //-----
00497 typedef struct __attribute__((packed))
00498 {
00499     union __attribute__((packed))
00500     {
00501         uint8_t r;
00502         struct __attribute__((packed))
00503         {
00504             unsigned int PCIE0 : 1;
00505             unsigned int PCIE1 : 1;
00506             unsigned int PCIE2 : 1;
00507             unsigned int reserved : 5;
00508         };
00509     };
00510 } AVR_PCICR;
00511
00512 //-----
00513 typedef struct __attribute__((packed))
00514 {
00515     union __attribute__((packed))
00516     {
00517         uint8_t r;
00518         struct __attribute__((packed))
00519         {
00520             unsigned int ISC00 : 1;
00521             unsigned int ISC01 : 1;
00522             unsigned int ISC10 : 1;
00523             unsigned int ISC11 : 1;
00524             unsigned int ISC20 : 1;
00525             unsigned int ISC21 : 1;
00526             unsigned int reserved : 2;
00527         };
00528     };
00529 } AVR_EICRA;
00530
00531 //-----
00532 typedef struct __attribute__((packed))
00533 {
00534     union __attribute__((packed))
00535     {
00536         uint8_t r;
00537         struct __attribute__((packed))
00538         {
00539             unsigned int PRADC : 1;
00540             unsigned int PRUSART0 : 1;
00541             unsigned int PRSPI : 1;
00542             unsigned int PRTIM1 : 1;
00543             unsigned int reserved : 1;
00544             unsigned int PRTIM0 : 1;
00545             unsigned int PRTIM2 : 1;

```

```

00546         unsigned int PRTWI : 1;
00547     };
00548 };
00549 } AVR_PRR;
00550
00551 //-----
00552 typedef struct __attribute__((packed))
00553 {
00554     union __attribute__((packed))
00555     {
00556         uint8_t r;
00557         struct __attribute__((packed))
00558         {
00559             unsigned int CLKPS0 : 1;
00560             unsigned int CLKPS1 : 1;
00561             unsigned int CLKPS2 : 1;
00562             unsigned int CLKPS3 : 1;
00563             unsigned int reserved : 3;
00564             unsigned int CLKPCE : 1;
00565         };
00566     };
00567 } AVR_CLKPR;
00568
00569 //-----
00570 typedef struct __attribute__((packed))
00571 {
00572     union __attribute__((packed))
00573     {
00574         uint8_t r;
00575         struct __attribute__((packed))
00576         {
00577             unsigned int WDP0 : 1;
00578             unsigned int WDP1 : 1;
00579             unsigned int WDP2 : 1;
00580             unsigned int WDE : 1;
00581             unsigned int WDCE : 1;
00582             unsigned int WDP3 : 1;
00583             unsigned int WDIE : 1;
00584             unsigned int WDIF : 1;
00585         };
00586     };
00587 } AVR_WDTCSR;
00588
00589 //-----
00590 typedef struct __attribute__((packed))
00591 {
00592     union __attribute__((packed))
00593     {
00594         uint8_t r;
00595         struct __attribute__((packed))
00596         {
00597             unsigned int C : 1;
00598             unsigned int Z : 1;
00599             unsigned int N : 1;
00600             unsigned int V : 1;
00601             unsigned int S : 1;
00602             unsigned int H : 1;
00603             unsigned int T : 1;
00604             unsigned int I : 1;
00605         };
00606     };
00607 } AVR_SREG;
00608
00609 //-----
00610 typedef struct __attribute__((packed))
00611 {
00612     union __attribute__((packed))
00613     {
00614         uint8_t r;
00615         struct __attribute__((packed))
00616         {
00617             unsigned int SP0 : 1;
00618             unsigned int SP1 : 1;
00619             unsigned int SP2 : 1;
00620             unsigned int SP3 : 1;
00621             unsigned int SP4 : 1;
00622             unsigned int SP5 : 1;
00623             unsigned int SP6 : 1;
00624             unsigned int SP7 : 1;
00625         };
00626     };
00627 } AVR_SPL;
00628
00629 //-----
00630 typedef struct __attribute__((packed))
00631 {
00632     union __attribute__((packed))

```

```

00633     {
00634         uint8_t r;
00635         struct __attribute__((packed))
00636         {
00637             unsigned int SP8      : 1;
00638             unsigned int SP9      : 1;
00639             unsigned int SP10     : 1;
00640             unsigned int reserved : 5;
00641         };
00642     };
00643 } AVR_SPH;
00644
00645 //-----
00646 typedef struct __attribute__((packed))
00647 {
00648     union __attribute__((packed))
00649     {
00650         uint8_t r;
00651         struct __attribute__((packed))
00652         {
00653             unsigned int SELFPRGEN : 1;
00654             unsigned int PGERS      : 1;
00655             unsigned int PGWRT      : 1;
00656             unsigned int BLBSET     : 1;
00657             unsigned int RWWSRE     : 1;
00658             unsigned int RWWSB      : 1;
00659             unsigned int SPMIE      : 1;
00660         };
00661     };
00662 } AVR_SPMCSR;
00663
00664 //-----
00665 typedef struct __attribute__((packed))
00666 {
00667     union __attribute__((packed))
00668     {
00669         uint8_t r;
00670         struct __attribute__((packed))
00671         {
00672             unsigned int IVCE       : 1;
00673             unsigned int IVSEL      : 1;
00674             unsigned int reserved   : 2;
00675             unsigned int PUD        : 1;
00676             unsigned int BODSE      : 1;
00677             unsigned int BODS       : 1;
00678             unsigned int reserved_  : 1;
00679         };
00680     };
00681 } AVR_MCUCR;
00682
00683 //-----
00684 typedef struct __attribute__((packed))
00685 {
00686     union __attribute__((packed))
00687     {
00688         uint8_t r;
00689         struct __attribute__((packed))
00690         {
00691             unsigned int PORF       : 1;
00692             unsigned int EXTRF      : 1;
00693             unsigned int BORF       : 1;
00694             unsigned int WDRF       : 1;
00695             unsigned int reserved   : 4;
00696         };
00697     };
00698 } AVR_MCUSR;
00699
00700 //-----
00701 typedef struct __attribute__((packed))
00702 {
00703     union __attribute__((packed))
00704     {
00705         uint8_t r;
00706         struct __attribute__((packed))
00707         {
00708             unsigned int SE         : 1;
00709             unsigned int SM0        : 1;
00710             unsigned int SM1        : 1;
00711             unsigned int SM2        : 1;
00712             unsigned int reserved   : 4;
00713         };
00714     };
00715 } AVR_SMCR;
00716
00717 //-----
00718 typedef struct __attribute__((packed))
00719 {

```

```

00720     union __attribute__ ((__packed__))
00721     {
00722         uint8_t r;
00723         struct __attribute__ ((__packed__))
00724         {
00725             unsigned int ACIS0 : 1;
00726             unsigned int ACIS1 : 1;
00727             unsigned int ACIC : 1;
00728             unsigned int ACIE : 1;
00729             unsigned int ACI : 1;
00730             unsigned int AC0 : 1;
00731             unsigned int ACBG : 1;
00732             unsigned int ACD : 1;
00733         };
00734     };
00735 } AVR_ACSR;
00736
00737 //-----
00738 typedef struct __attribute__ ((__packed__))
00739 {
00740     union __attribute__ ((__packed__))
00741     {
00742         uint8_t r;
00743         struct __attribute__ ((__packed__))
00744         {
00745             unsigned int SPRO : 1;
00746             unsigned int SPR1 : 1;
00747             unsigned int CPHA : 1;
00748             unsigned int CPOL : 1;
00749             unsigned int MSTR : 1;
00750             unsigned int DORD : 1;
00751             unsigned int SPE : 1;
00752             unsigned int SPIE : 1;
00753         };
00754     };
00755 } AVR_SPCR;
00756
00757 //-----
00758 typedef struct __attribute__ ((__packed__))
00759 {
00760     union __attribute__ ((__packed__))
00761     {
00762         uint8_t r;
00763         struct __attribute__ ((__packed__))
00764         {
00765             unsigned int SPI2X : 1;
00766             unsigned int reserved : 5;
00767             unsigned int WCOL : 1;
00768             unsigned int SPIF : 1;
00769         };
00770     };
00771 } AVR_SPSR;
00772
00773 //-----
00774 typedef struct __attribute__ ((__packed__))
00775 {
00776     union __attribute__ ((__packed__))
00777     {
00778         uint8_t r;
00779         struct __attribute__ ((__packed__))
00780         {
00781             unsigned int PSRSYNC : 1;
00782             unsigned int PSRASY : 1;
00783             unsigned int reserved : 5;
00784             unsigned int TSM : 1;
00785         };
00786     };
00787 } AVR_GTCCR;
00788
00789 //-----
00790 typedef struct __attribute__ ((__packed__))
00791 {
00792     union __attribute__ ((__packed__))
00793     {
00794         uint8_t r;
00795         struct __attribute__ ((__packed__))
00796         {
00797             unsigned int WGM00 : 1;
00798             unsigned int WGM01 : 1;
00799             unsigned int reserved : 2;
00800             unsigned int COM0B0 : 1;
00801             unsigned int COM0B1 : 1;
00802             unsigned int COM0A0 : 1;
00803             unsigned int COM0A1 : 1;
00804         };
00805     };
00806 } AVR_TCCR0A;

```

```

00807
00808 //-----
00809 typedef struct __attribute__((packed))
00810 {
00811     union __attribute__((packed))
00812     {
00813         uint8_t r;
00814         struct __attribute__((packed))
00815         {
00816             unsigned int CS00 : 1;
00817             unsigned int CS01 : 1;
00818             unsigned int CS02 : 1;
00819             unsigned int WGM02 : 1;
00820             unsigned int reserved : 2;
00821             unsigned int FOC0B : 1;
00822             unsigned int FOC0A : 1;
00823         };
00824     };
00825 } AVR_TCCRB;
00826
00827 //-----
00828 typedef struct __attribute__((packed))
00829 {
00830     union __attribute__((packed))
00831     {
00832         uint8_t r;
00833         struct __attribute__((packed))
00834         {
00835             unsigned int EERE : 1;
00836             unsigned int EEPE : 1;
00837             unsigned int EEMPE : 1;
00838             unsigned int EERIE : 1;
00839             unsigned int EEPM0 : 1;
00840             unsigned int EEPM1 : 1;
00841             unsigned int reserved : 2;
00842         };
00843     };
00844 } AVR_EECR;
00845
00846 //-----
00847 // External interrupt flag register definitions
00848 //-----
00849 typedef struct __attribute__((packed))
00850 {
00851     union __attribute__((packed))
00852     {
00853         uint8_t r;
00854         struct __attribute__((packed))
00855         {
00856             unsigned int INTF0 : 1;
00857             unsigned int INTF1 : 1;
00858             unsigned int INTF2 : 1;
00859             unsigned int reserved : 5;
00860         };
00861     };
00862 } AVR_EIFR;
00863
00864 //-----
00865 // External interrupt mask register definitions
00866 //-----
00867 typedef struct __attribute__((packed))
00868 {
00869     union __attribute__((packed))
00870     {
00871         uint8_t r;
00872         struct __attribute__((packed))
00873         {
00874             unsigned int INTO : 1;
00875             unsigned int INT1 : 1;
00876             unsigned int INT2 : 1;
00877             unsigned int reserved : 5;
00878         };
00879     };
00880 } AVR_EIMSK;
00881
00882 //-----
00883 // Pin (GPIO) register definitions
00884 //-----
00885 typedef struct __attribute__((packed))
00886 {
00887     union __attribute__((packed))
00888     {
00889         uint8_t r;
00890         struct __attribute__((packed))
00891         {
00892             unsigned int PIN0 : 1;
00893             unsigned int PIN1 : 1;

```



```

00894         unsigned int PIN2 : 1;
00895         unsigned int PIN3 : 1;
00896         unsigned int PIN4 : 1;
00897         unsigned int PIN5 : 1;
00898         unsigned int PIN6 : 1;
00899         unsigned int PIN7 : 1;
00900     };
00901 };
00902 } AVR_PIN;
00903
00904 //-----
00905 // Data-direction register (GPIO) definitions
00906 //-----
00907 typedef struct __attribute__((packed))
00908 {
00909     union __attribute__((packed))
00910     {
00911         uint8_t r;
00912         struct __attribute__((packed))
00913         {
00914             unsigned int DDR0 : 1;
00915             unsigned int DDR1 : 1;
00916             unsigned int DDR2 : 1;
00917             unsigned int DDR3 : 1;
00918             unsigned int DDR4 : 1;
00919             unsigned int DDR5 : 1;
00920             unsigned int DDR6 : 1;
00921             unsigned int DDR7 : 1;
00922         };
00923     };
00924 } AVR_DDR;
00925
00926 //-----
00927 // Port (GPIO) register definitions
00928 //-----
00929 typedef struct __attribute__((packed))
00930 {
00931     union __attribute__((packed))
00932     {
00933         uint8_t r;
00934         struct __attribute__((packed))
00935         {
00936             unsigned int PORT0 : 1;
00937             unsigned int PORT1 : 1;
00938             unsigned int PORT2 : 1;
00939             unsigned int PORT3 : 1;
00940             unsigned int PORT4 : 1;
00941             unsigned int PORT5 : 1;
00942             unsigned int PORT6 : 1;
00943             unsigned int PORT7 : 1;
00944         };
00945     };
00946 } AVR_PORT;
00947
00948
00949 //-----
00950 // Timer interrupt flag register struct __attribute__((packed)) definitions
00951 //-----
00952 typedef struct __attribute__((packed))
00953 {
00954     union __attribute__((packed))
00955     {
00956         uint8_t r;
00957         struct __attribute__((packed))
00958         {
00959             unsigned int TOV0 : 1;
00960             unsigned int OCF0A : 1;
00961             unsigned int OCF0B : 1;
00962             unsigned int reserved : 5;
00963         };
00964     };
00965 } AVR_TIFR0;
00966
00967 //-----
00968 typedef struct __attribute__((packed))
00969 {
00970     union __attribute__((packed))
00971     {
00972         uint8_t r;
00973         struct __attribute__((packed))
00974         {
00975             unsigned int TOV1 : 1;
00976             unsigned int OCF1A : 1;
00977             unsigned int OCF1B : 1;
00978             unsigned int reserved : 2;
00979             unsigned int ICF1 : 1;
00980             unsigned int reserved_ : 2;

```

```

00981     };
00982     };
00983 } AVR_TIFR1;
00984
00985 //-----
00986 typedef struct __attribute__((packed))
00987 {
00988     union __attribute__((packed))
00989     {
00990         uint8_t r;
00991         struct __attribute__((packed))
00992         {
00993             unsigned int TOV2      : 1;
00994             unsigned int OCF2A     : 1;
00995             unsigned int OCF2B     : 1;
00996             unsigned int reserved : 5;
00997         };
00998     };
00999 } AVR_TIFR2;
01000
01001 //-----
01002 // Pin-change interrupt flag bits
01003 //-----
01004 typedef struct __attribute__((packed))
01005 {
01006     union __attribute__((packed))
01007     {
01008         uint8_t r;
01009         struct __attribute__((packed))
01010         {
01011             unsigned int PCIF0      : 1;
01012             unsigned int PCIF1      : 1;
01013             unsigned int PCIF2      : 1;
01014             unsigned int reserved : 5;
01015         };
01016     };
01017 } AVR_PCIFR;
01018
01019 #endif // __AVR_PERIPHRGS_H__

```

## 4.135 src/peripheral/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](#) **EEPROM\_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**

### 4.135.1 Detailed Description

AVR atmega EEPROM plugin.

Definition in file [mega\\_eeprom.c](#).

### 4.135.2 Enumeration Type Documentation

#### 4.135.2.1 EEPROM\_Mode\_t

enum [EEPROM\\_Mode\\_t](#)

#### Enumerator

<a href="#">EEPROM_MODE_ATOMIC</a>	Atomic Clear/Write operation.
<a href="#">EEPROM_MODE_ERASE</a>	Erase only.
<a href="#">EEPROM_MODE_WRITE</a>	Write only.

Definition at line 50 of file [mega\\_eeprom.c](#).

#### 4.135.2.2 EEPROM\_State\_t

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](#).

### 4.135.3 Function Documentation

#### 4.135.3.1 EEPROM\_Write()

```
static void EEPROM_Write (  
    void * context_,  
    uint8_t ucAddr_,  
    uint8_t ucValue_ ) [static]
```

! ToDo - Fix the times to use RC-oscillator times, not CPU-clock times.

Definition at line 183 of file [mega\\_eeprom.c](#).

### 4.135.4 Variable Documentation

#### 4.135.4.1 stEEPROM

[AVRPeripheral](#) stEEPROM

##### Initial value:

```
=  
{  
    EEPROM_Init,  
    EEPROM_Read,  
    EEPROM_Write,  
    EEPROM_Clock,  
    0,  
    0x3F,  
    0x3F  
}
```

Definition at line 310 of file [mega\\_eeprom.c](#).

## 4.136 mega\_eeprom.c

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( (/( (      (      | -- [ Funkenstein ] -----
00005 *      /( ) )/( )((( ( )\ )\ /( ) | -- [ Little ] -----
00006 *      ( )_ ( )      )\ _ )\ ( ( ( ) | -- [ AVR ] -----
00007 *      | _ | _ |      ( )\ ( )\ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ |      / _ \ \ \ / / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ | / _ \ \ \ \ / | _ \ |
00010 *      | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #include "mega_eeprom.h"
00022
00023 #include "avr_cpu.h"
00024
00025 #include <stdint.h>
00026 #include <stdbool.h>
00027 #include <stdlib.h>
00028 #include <stdio.h>
00029 #include <string.h>
00030
00031 #if 1
00032 #define DEBUG_PRINT(...)
00033 #else
00034 #define DEBUG_PRINT printf
00035 #endif
00036
00037 //-----
00038 typedef enum
00039 {
00040     EEPROM_STATE_IDLE = 0,
00041     EEPROM_STATE_WRITE_ENABLE,
00042     EEPROM_STATE_READ,
00043     EEPROM_STATE_WRITE,
00044     //--
00045     EEPROM_STATES
00046 } EEPROM_State_t;
00047
00048 //-----
00049 //-----
00050 typedef enum
00051 {
00052     EEPROM_MODE_ATOMIC = 0,
00053     EEPROM_MODE_ERASE,
00054     EEPROM_MODE_WRITE,
00055     //----
00056     EEPROM_MODES
00057 } EEPROM_Mode_t;
00058
00059 //-----
00060 static EEPROM_State_t eState = EEPROM_STATE_IDLE;
00061 static uint32_t u32CountDown = 0;
00062
00063 //-----
00064 static void EEARH_Write( uint8_t u8Addr_ )
00065 {
00066     stCPU.pstRAM->stRegisters.EEARH = (u8Addr_ & 0x03);
00067 }
00068
00069 //-----
00070 static void EEARL_Write( uint8_t u8Addr_ )
00071 {
00072     stCPU.pstRAM->stRegisters.EEARL = u8Addr_;
00073 }
00074
00075 //-----
00076 static uint16_t EEAR_Read( void )
00077 {
00078     uint16_t u16Addr;
00079     u16Addr = ((uint16_t) (stCPU.pstRAM->stRegisters.EEARH) << 8) |
00080             (uint16_t) (stCPU.pstRAM->stRegisters.EEARL);
00081     return u16Addr;
00082 }
00083
00084 //-----
00085 static void EEPE_Clear(void)
00086 {
00087     stCPU.pstRAM->stRegisters.EECP.EEPE = 0;
00088 }
00089
00090 //-----

```

```

00091 static void EEPE_Set(void)
00092 {
00093     stCPU.pstRAM->stRegisters.EECP.EEPE = 1;
00094 }
00095
00096 //-----
00097 static bool EEPE_Read(void)
00098 {
00099     return (stCPU.pstRAM->stRegisters.EECP.EEPE == 1);
00100 }
00101 //-----
00102 static void EERE_Clear(void)
00103 {
00104     stCPU.pstRAM->stRegisters.EECP.EERE = 0;
00105 }
00106
00107 //-----
00108 static void EERE_Set(void)
00109 {
00110     stCPU.pstRAM->stRegisters.EECP.EERE = 1;
00111 }
00112
00113 //-----
00114 static bool EERE_Read(void)
00115 {
00116     return (stCPU.pstRAM->stRegisters.EECP.EERE == 1);
00117 }
00118 //-----
00119 static void EEMPE_Clear(void)
00120 {
00121     stCPU.pstRAM->stRegisters.EECP.EEMPE = 0;
00122 }
00123
00124 //-----
00125 static void EEMPE_Set(void)
00126 {
00127     stCPU.pstRAM->stRegisters.EECP.EEMPE = 1;
00128 }
00129
00130 //-----
00131 static bool EEMPE_Read(void)
00132 {
00133     return (stCPU.pstRAM->stRegisters.EECP.EEMPE == 1);
00134 }
00135
00136 //-----
00137 static void EERIE_Clear(void)
00138 {
00139     stCPU.pstRAM->stRegisters.EECP.EERIE = 0;
00140 }
00141
00142 //-----
00143 static void EERIE_Set(void)
00144 {
00145     stCPU.pstRAM->stRegisters.EECP.EERIE = 1;
00146 }
00147
00148 //-----
00149 static bool EERIE_Read(void)
00150 {
00151     return (stCPU.pstRAM->stRegisters.EECP.EERIE == 1);
00152 }
00153
00154 //-----
00155 static EEPROM_Mode_t EEPM_Read(void)
00156 {
00157     EEPROM_Mode_t eRet;
00158     eRet = (EEPROM_Mode_t) (stCPU.pstRAM->stRegisters.EECP.r & (0x30)) >> 4;
00159     return eRet;
00160 }
00161
00162 //-----
00163 static uint8_t EEDR_Read(void)
00164 {
00165     return stCPU.pstRAM->stRegisters.EEDR;
00166 }
00167
00168 //-----
00169 static void EEPROM_Init(void *context_)
00170 {
00171     eState = EEPROM_STATE_IDLE;
00172     u32CountDown = 0;
00173 }
00174
00175 //-----
00176 static void EEPROM_Read(void *context_, uint8_t ucAddr_, uint8_t *pucValue_)
00177 {

```

```

00178     DEBUG_PRINT( "EEPROM Read %2x\n", stCPU.pstRAM->stRegisters.EECR.r );
00179     *pucValue_ = stCPU.pstRAM->stRegisters.EECR.r;
00180 }
00181
00182 //-----
00183 static void EEPROM_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
00184 {
00185     // We're only interested in the EECR register. If we really want to be
00186     // 100% CPU-accurate, we'd take into account a ton of additional logic for
00187     // other peripherals (CPU SPM registers, etc.), but that's a lot of code
00188     // when pretty much everyone is going to be using the app note or the AVR
00189     // libc implementation, which is very much "sunny case" code. In short,
00190     // this will handle incorrectly-implemented code incorrectly.
00191
00192     stCPU.pstRAM->stRegisters.EECR.r |= (ucValue_ & 0x3F);
00193
00194     switch (eState)
00195     {
00196     case EEPROM_STATE_IDLE:
00197     {
00198         if ((ucValue_ & 0x01) == 0x01) // Read
00199         {
00200             // When the data is read, the data is available in the next instruction
00201             // but the CPU is halted for 4 cycles before it's executed.
00202             DEBUG_PRINT( "EEPROM Read\n" );
00203             eState = EEPROM_STATE_READ;
00204             u32CountDown = 4;
00205
00206             stCPU.u16ExtraCycles += u32CountDown;
00207             stCPU.u64CycleCount += u32CountDown;
00208
00209             // Read data at EEPROM address to EEPROM data register
00210             stCPU.pstRAM->stRegisters.EEDR = stCPU.pu8EEPROM[ EEAR_Read() ];
00211         }
00212         else if ((ucValue_ & 0x04) == 0x04) // Program Enable
00213         {
00214             // Must initiate a write within 4 cycles of enabling the EEPROM write bit
00215             DEBUG_PRINT( "EEPROM Write Enable\n" );
00216             eState = EEPROM_STATE_WRITE_ENABLE;
00217             u32CountDown = 4;
00218         }
00219     }
00220     break;
00221
00222     case EEPROM_STATE_WRITE_ENABLE:
00223     {
00224         if ((ucValue_ & 0x02) == 0x02) // Value has EEPE
00225         {
00226             eState = EEPROM_STATE_WRITE;
00227             DEBUG_PRINT( "EEPROM Write\n" );
00228             switch ( EEPM_Read() )
00229             {
00230             case EEPROM_MODE_ATOMIC:
00231             {
00232                 stCPU.pu8EEPROM[ EEAR_Read() ] = EEDR_Read();
00233                 u32CountDown = 48000;
00234             }
00235             break;
00236             case EEPROM_MODE_WRITE:
00237             {
00238                 // EEPROM works by setting individual bits -- once a bit is set, it must be
00239                 // cleared before it can be reset.
00240                 stCPU.pu8EEPROM[ EEAR_Read() ] |= EEDR_Read();
00241                 u32CountDown = 25000;
00242             }
00243             break;
00244             case EEPROM_MODE_ERASE:
00245             {
00246                 // EEPROM is 0 when cleared
00247                 stCPU.pu8EEPROM[ EEAR_Read() ] = 0x00;
00248                 u32CountDown = 25000;
00249             }
00250             break;
00251             default:
00252                 break;
00253             }
00254         }
00255     }
00256     break;
00257     default:
00258         break;
00259     }
00260 }
00261 }
00262
00263 //-----
00264 static void EEPROM_Clock(void *context_)
00265 {

```

```

00266
00267     if (u32CountDown)
00268     {
00269         // DEBUG_PRINT( "EEPROM Clock %d\n", u32CountDown );
00270
00271         u32CountDown--;
00272         if (!u32CountDown)
00273         {
00274             // We're only interested in the EECR register.
00275             switch (eState)
00276             {
00277                 case EEPROM_STATE_WRITE:
00278                 {
00279                     EEPE_Clear();
00280                     EERE_Clear();
00281                     EEMPE_Clear();
00282
00283                     eState = EEPROM_STATE_IDLE;
00284                 }
00285                 break;
00286                 case EEPROM_STATE_READ:
00287                 {
00288                     EEPE_Clear();
00289                     EERE_Clear();
00290                     EEMPE_Clear();
00291
00292                     eState = EEPROM_STATE_IDLE;
00293                 }
00294                 break;
00295                 case EEPROM_STATE_WRITE_ENABLE:
00296                 {
00297                     EEMPE_Clear();
00298                     EERE_Clear();
00299                     eState = EEPROM_STATE_IDLE;
00300                 }
00301                 break;
00302                 default:
00303                     break;
00304             }
00305         }
00306     }
00307 }
00308
00309 //-----
00310 AVRPeripheral stEEPROM =
00311 {
00312     EEPROM_Init,
00313     EEPROM_Read,
00314     EEPROM_Write,
00315     EEPROM_Clock,
00316     0,
00317     0x3F,
00318     0x3F
00319 };
00320

```

## 4.137 src/peripheral/mega\_eeprom.h File Reference

AVR atmega EEPROM plugin.

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

### Variables

- [AVRPeripheral](#) **stEEPROM**

#### 4.137.1 Detailed Description

AVR atmega EEPROM plugin.

Definition in file [mega\\_eeprom.h](#).



## 4.138 mega\_eeprom.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ )  )\ )      (      |
00004 *      ((/( ((/(      )\      | -- [ Funkenstein ] -----
00005 *      /( )  /( )  ((( ( )\  )\  / ( )  | -- [ Little ] -----
00006 *      ( ) _ ( )  )\ _ )\  ( ( ) ( ) ( )  | -- [ AVR ] -----
00007 *      | | _ | |  ( ) _ ( ) \ \ / / | _ \  | -- [ Virtual ] -----
00008 *      | _ | | _  / _ \ \ \ v / | _ /  | -- [ Runtime ] -----
00009 *      | _ | | _  / _ \ \ \ \ / | _ \  |
00010 *                                     | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __MEGA_EEPROM_H__
00022 #define __MEGA_EEPROM_H__
00023
00024 #include "avr_peripheral.h"
00025
00026
00027 extern AVRPeripheral stEEPROM;
00028
00029 #endif // __MEGA_EEPROM_H__

```

## 4.139 src/peripheral/mega\_eint.c File Reference

ATMega External Interrupt Implementation.

```

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

```

### Macros

- #define **DEBUG\_PRINT**(...)

### Enumerations

- enum **InterruptSense\_t** { **INT\_SENSE\_LOW** = 0, **INT\_SENSE\_CHANGE**, **INT\_SENSE\_FALL**, **INT\_SENSE\_RISE**, **INT\_SENSE\_BOTH** }

### 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 [InterruptSense\\_t](#) **eINT2Sense**
- static uint8\_t **ucLastINT0**
- static uint8\_t **ucLastINT1**
- static uint8\_t **ucLastINT2**
- [AVRPeripheral](#) **stEINT\_a**
- [AVRPeripheral](#) **stEINT\_b**

### 4.139.1 Detailed Description

ATMega External Interrupt Implementation.

Definition in file [mega\\_eint.c](#).

### 4.139.2 Enumeration Type Documentation

#### 4.139.2.1 InterruptSense\_t

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 36 of file [mega\\_eint.c](#).

### 4.139.3 Function Documentation

#### 4.139.3.1 EINT\_Clock()

```
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 201 of file [mega\\_eint.c](#).

#### 4.139.4 Variable Documentation

##### 4.139.4.1 stEINT\_a

`AVRPeripheral` stEINT\_a

##### Initial value:

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

Definition at line 363 of file [mega\\_eint.c](#).

##### 4.139.4.2 stEINT\_b

`AVRPeripheral` stEINT\_b

##### Initial value:

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

Definition at line 375 of file [mega\\_eint.c](#).

## 4.140 mega\_eint.c

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( ((/(      (      | -- [ Funkenstein ] -----
00005 *      /(_ ) /(_ ) ((((_ )\ )\ /(_ ) | -- [ Little ] -----
00006 *      (_ )_ | (_ )      )\ _ )\ (( _ ) | -- [ AVR ] -----
00007 *      | | _ | |      (_ )\(_ )\ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | | _      / _ \ \ \ / / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ | /_ / \_ \ \_ / | _ | \ |
00010 *      | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #include <stdio.h>
00022 #include <stdlib.h>
00023 #include <string.h>
00024 #include "avr_cpu.h"
00025 #include "avr_peripheral.h"
00026 #include "avr_periphregs.h"
00027 #include "avr_interrupt.h"
00028
00029 #if 1
00030 #define DEBUG_PRINT(...)
00031 #else
00032 #define DEBUG_PRINT printf
00033 #endif
00034
00035 //-----
00036 typedef enum
00037 {
00038     INT_SENSE_LOW = 0,
00039     INT_SENSE_CHANGE,
00040     INT_SENSE_FALL,
00041     INT_SENSE_RISE
00042 } InterruptSense_t;
00043
00044 //-----
00045 static InterruptSense_t eINT0Sense;
00046 static InterruptSense_t eINT1Sense;
00047 static InterruptSense_t eINT2Sense;
00048
00049 static uint8_t ucLastINT0;
00050 static uint8_t ucLastINT1;
00051 static uint8_t ucLastINT2;
00052
00053 //-----
00054 static void EINT_AckInt( uint8_t ucVector_ );
00055
00056 //-----
00057 static void EINT_Init(void *context_ )
00058 {
00059     DEBUG_PRINT("EINT INIT\n");
00060     eINT0Sense = INT_SENSE_LOW;
00061     eINT1Sense = INT_SENSE_LOW;
00062     eINT2Sense = INT_SENSE_LOW;
00063
00064     ucLastINT0 = 0;
00065     ucLastINT1 = 0;
00066     ucLastINT2 = 0;
00067
00068     // Register interrupt callback functions
00069     CPU_RegisterInterruptCallback(EINT_AckInt, stCPU.pstVectorMap->INT0);
00070     CPU_RegisterInterruptCallback(EINT_AckInt, stCPU.pstVectorMap->INT1);
00071     CPU_RegisterInterruptCallback(EINT_AckInt, stCPU.pstVectorMap->INT2);
00072 }
00073
00074 //-----
00075 static void EINT_Read(void *context_, uint8_t ucAddr_, uint8_t *pucValue_ )
00076 {
00077     *pucValue_ = stCPU.pstRAM->au8RAM[ucAddr_];
00078 }
00079
00080 //-----
00081 static void EICRA_Write( uint8_t ucValue_ )
00082 {
00083     DEBUG_PRINT("EICRA Clock\n");
00084     stCPU.pstRAM->stRegisters.EICRA.r = ucValue_;
00085
00086     // Change local interrupt sense value.
00087     if ((stCPU.pstRAM->stRegisters.EICRA.ISC00 == 0) &&
00088         (stCPU.pstRAM->stRegisters.EICRA.ISC01 == 0))
00089     {
00090         DEBUG_PRINT("I0-low\n");

```

```

00091     eINT0Sense = INT_SENSE_LOW;
00092 }
00093 else if ((stCPU.pstRAM->stRegisters.EICRA.ISC00 == 1) &&
00094         (stCPU.pstRAM->stRegisters.EICRA.ISC01 == 0))
00095 {
00096     DEBUG_PRINT("I0-change\n");
00097     eINT0Sense = INT_SENSE_CHANGE;
00098 }
00099 else if ((stCPU.pstRAM->stRegisters.EICRA.ISC00 == 0) &&
00100         (stCPU.pstRAM->stRegisters.EICRA.ISC01 == 1))
00101 {
00102     DEBUG_PRINT("I0-fall\n");
00103     eINT0Sense = INT_SENSE_FALL;
00104 }
00105 else if ((stCPU.pstRAM->stRegisters.EICRA.ISC00 == 1) &&
00106         (stCPU.pstRAM->stRegisters.EICRA.ISC01 == 1))
00107 {
00108     DEBUG_PRINT("I0-risel\n");
00109     eINT0Sense = INT_SENSE_RISE;
00110 }
00111
00112 if ((stCPU.pstRAM->stRegisters.EICRA.ISC10 == 0) &&
00113     (stCPU.pstRAM->stRegisters.EICRA.ISC11 == 0))
00114 {
00115     eINT1Sense = INT_SENSE_LOW;
00116 }
00117 else if ((stCPU.pstRAM->stRegisters.EICRA.ISC10 == 1) &&
00118         (stCPU.pstRAM->stRegisters.EICRA.ISC11 == 0))
00119 {
00120     eINT1Sense = INT_SENSE_CHANGE;
00121 }
00122 else if ((stCPU.pstRAM->stRegisters.EICRA.ISC10 == 0) &&
00123         (stCPU.pstRAM->stRegisters.EICRA.ISC11 == 1))
00124 {
00125     eINT1Sense = INT_SENSE_FALL;
00126 }
00127 else if ((stCPU.pstRAM->stRegisters.EICRA.ISC10 == 1) &&
00128         (stCPU.pstRAM->stRegisters.EICRA.ISC11 == 1))
00129 {
00130     eINT1Sense = INT_SENSE_RISE;
00131 }
00132
00133 if ((stCPU.pstRAM->stRegisters.EICRA.ISC20 == 0) &&
00134     (stCPU.pstRAM->stRegisters.EICRA.ISC21 == 0))
00135 {
00136     eINT2Sense = INT_SENSE_LOW;
00137 }
00138 else if ((stCPU.pstRAM->stRegisters.EICRA.ISC20 == 1) &&
00139         (stCPU.pstRAM->stRegisters.EICRA.ISC21 == 0))
00140 {
00141     eINT2Sense = INT_SENSE_CHANGE;
00142 }
00143 else if ((stCPU.pstRAM->stRegisters.EICRA.ISC20 == 0) &&
00144         (stCPU.pstRAM->stRegisters.EICRA.ISC21 == 1))
00145 {
00146     eINT2Sense = INT_SENSE_FALL;
00147 }
00148 else if ((stCPU.pstRAM->stRegisters.EICRA.ISC20 == 1) &&
00149         (stCPU.pstRAM->stRegisters.EICRA.ISC21 == 1))
00150 {
00151     eINT2Sense = INT_SENSE_RISE;
00152 }
00153
00154 DEBUG_PRINT("IntSense0,1,2: %d, %d, %d\n", eINT0Sense, eINT1Sense, eINT2Sense);
00155 DEBUG_PRINT("EICRA: %d, ISC00 : %d, ISC01 : %d, ISC10: %d, ISC11: %d, ISC20: %d, ISC21: %d\n",
00156             stCPU.pstRAM->stRegisters.EICRA.r,
00157             stCPU.pstRAM->stRegisters.EICRA.ISC00,
00158             stCPU.pstRAM->stRegisters.EICRA.ISC01,
00159             stCPU.pstRAM->stRegisters.EICRA.ISC10,
00160             stCPU.pstRAM->stRegisters.EICRA.ISC11,
00161             stCPU.pstRAM->stRegisters.EICRA.ISC20,
00162             stCPU.pstRAM->stRegisters.EICRA.ISC21
00163         );
00164 }
00165
00166 //-----
00167 static void EIFR_Write( uint8_t ucValue_ )
00168 {
00169     DEBUG_PRINT("EIFR Clock\n");
00170     stCPU.pstRAM->stRegisters(EIFR.r = ucValue_;
00171 }
00172
00173 //-----
00174 static void EIMSK_Write( uint8_t ucValue_ )
00175 {
00176     DEBUG_PRINT("EIMSK Write\n");

```

```

00177     stCPU.pstRAM->stRegisters.EIMSK.r = ucValue_;
00178 }
00179
00180 //-----
00181 static void EINT_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
00182 {
00183     DEBUG_PRINT("EINT Write\n");
00184     switch (ucAddr_)
00185     {
00186     case 0x69: // EICRA
00187         EICRA_Write(ucValue_);
00188         break;
00189     case 0x3C: // EIFR
00190         EIFR_Write(ucValue_);
00191         break;
00192     case 0x3D: // EIMSK
00193         EIMSK_Write(ucValue_);
00194         break;
00195     default:
00196         break;
00197     }
00198 }
00199
00200 //-----
00201 static void EINT_Clock(void *context_ )
00202 {
00203     // Check to see if interrupts are enabled. If so, check to see if the
00204     // interrupt mask is set, and then finally - whether or not an interrupt
00205     // condition has occurred based on the interrupt sense mode.
00206     bool bSetINT0 = false;
00207     bool bSetINT1 = false;
00208     bool bSetINT2 = false;
00209
00210     if (stCPU.pstRAM->stRegisters.EIMSK.INT0 == 1)
00211     {
00212         switch (eINT0Sense)
00213         {
00214         case INT_SENSE_LOW:
00215             if (stCPU.pstRAM->stRegisters.PORTD.PORT2 == 0)
00216             {
00217                 DEBUG_PRINT(" SET INT0\n");
00218                 bSetINT0 = true;
00219             }
00220             break;
00221         case INT_SENSE_CHANGE:
00222             if (stCPU.pstRAM->stRegisters.PORTD.PORT2 != ucLastINT0)
00223             {
00224                 DEBUG_PRINT(" SET INT0\n");
00225                 bSetINT0 = true;
00226             }
00227             break;
00228         case INT_SENSE_FALL:
00229             if ((stCPU.pstRAM->stRegisters.PORTD.PORT2 == 0) && (ucLastINT0 == 1))
00230             {
00231                 DEBUG_PRINT(" SET INT0\n");
00232                 bSetINT0 = true;
00233             }
00234             break;
00235         case INT_SENSE_RISE:
00236             if ((stCPU.pstRAM->stRegisters.PORTD.PORT2 == 1) && (ucLastINT0 == 0))
00237             {
00238                 DEBUG_PRINT(" SET INT0\n");
00239                 bSetINT0 = true;
00240             }
00241             break;
00242         }
00243     }
00244
00245     if (stCPU.pstRAM->stRegisters.EIMSK.INT1 == 1)
00246     {
00247         switch (eINT0Sense)
00248         {
00249         case INT_SENSE_LOW:
00250             if (stCPU.pstRAM->stRegisters.PORTD.PORT3 == 0)
00251             {
00252                 DEBUG_PRINT(" SET INT1\n");
00253                 bSetINT1 = true;
00254             }
00255             break;
00256         case INT_SENSE_CHANGE:
00257             if (stCPU.pstRAM->stRegisters.PORTD.PORT3 != ucLastINT1)
00258             {
00259                 DEBUG_PRINT(" SET INT1\n");
00260                 bSetINT1 = true;
00261             }
00262             break;
00263         case INT_SENSE_FALL:
00264             if ((stCPU.pstRAM->stRegisters.PORTD.PORT3 == 0) && (ucLastINT1 == 1))
00265             {
00266                 DEBUG_PRINT(" SET INT1\n");
00267                 bSetINT1 = true;
00268             }
00269             break;
00270         case INT_SENSE_RISE:
00271             if ((stCPU.pstRAM->stRegisters.PORTD.PORT3 == 1) && (ucLastINT1 == 0))
00272             {
00273                 DEBUG_PRINT(" SET INT1\n");
00274                 bSetINT1 = true;
00275             }
00276             break;
00277         }
00278     }
00279 }

```

```

00267         if ((stCPU.pstRAM->stRegisters.PORTD.PORT3 == 0) && (ucLastINT1 == 1))
00268         {
00269             DEBUG_PRINT(" SET INT1\n");
00270             bSetINT1 = true;
00271         }
00272         break;
00273     case INT_SENSE_RISE:
00274         if ((stCPU.pstRAM->stRegisters.PORTD.PORT3 == 1) && (ucLastINT1 == 0))
00275         {
00276             DEBUG_PRINT(" SET INT1\n");
00277             bSetINT1 = true;
00278         }
00279         break;
00280     }
00281 }
00282 if (stCPU.pstRAM->stRegisters.EIMSK.INT2 == 1)
00283 {
00284     switch (eINT2Sense)
00285     {
00286     case INT_SENSE_LOW:
00287         if (stCPU.pstRAM->stRegisters.PORTB.PORT2 == 0)
00288         {
00289             DEBUG_PRINT(" SET INT2\n");
00290             bSetINT2 = true;
00291         }
00292         break;
00293     case INT_SENSE_CHANGE:
00294         if (stCPU.pstRAM->stRegisters.PORTB.PORT2 != ucLastINT2)
00295         {
00296             DEBUG_PRINT(" SET INT2\n");
00297             bSetINT2 = true;
00298         }
00299         break;
00300     case INT_SENSE_FALL:
00301         if ((stCPU.pstRAM->stRegisters.PORTB.PORT2 == 0) && (ucLastINT2 == 1))
00302         {
00303             DEBUG_PRINT(" SET INT2\n");
00304             bSetINT2 = true;
00305         }
00306         break;
00307     case INT_SENSE_RISE:
00308         if ((stCPU.pstRAM->stRegisters.PORTB.PORT2 == 1) && (ucLastINT2 == 0))
00309         {
00310             DEBUG_PRINT(" SET INT2\n");
00311             bSetINT2 = true;
00312         }
00313         break;
00314     }
00315 }
00316 // Trigger interrupts where necessary
00317 if (bSetINT0)
00318 {
00319     stCPU.pstRAM->stRegisters.EIFR.INTF0 = 1;
00320     AVR_InterruptCandidate(stCPU.pstVectorMap->INT0);
00321 }
00322 if (bSetINT1)
00323 {
00324     stCPU.pstRAM->stRegisters.EIFR.INTF1 = 1;
00325     AVR_InterruptCandidate(stCPU.pstVectorMap->INT1);
00326 }
00327 if (bSetINT2)
00328 {
00329     stCPU.pstRAM->stRegisters.EIFR.INTF2 = 1;
00330     AVR_InterruptCandidate(stCPU.pstVectorMap->INT2);
00331 }
00332 // Update locally-cached copy of previous INT0/INT1 pin status.
00333 ucLastINT0 = stCPU.pstRAM->stRegisters.PORTD.PORT2;
00334 ucLastINT1 = stCPU.pstRAM->stRegisters.PORTD.PORT3;
00335 ucLastINT2 = stCPU.pstRAM->stRegisters.PORTB.PORT2;
00336 }
00337 }
00338
00339 //-----
00340 static void EINT_AckInt( uint8_t ucVector_)
00341 {
00342     DEBUG_PRINT("EINT ACK INT\n");
00343     // We automatically clear the INTx flag as soon as the interrupt
00344     // is acknowledged.
00345     if (ucVector_ == stCPU.pstVectorMap->INT0)
00346     {
00347         DEBUG_PRINT("INT0!\n");
00348         stCPU.pstRAM->stRegisters.EIFR.INTF0 = 0;
00349     }
00350     else if (ucVector_ == stCPU.pstVectorMap->INT1)
00351     {
00352         DEBUG_PRINT("INT1!\n");
00353         stCPU.pstRAM->stRegisters.EIFR.INTF1 = 0;

```

```

00354     }
00355     else if (ucVector_ == stCPU.pstVectorMap->INT2)
00356     {
00357         DEBUG_PRINT("INT2!\n");
00358         stCPU.pstRAM->stRegisters.EIFR.INTF2 = 0;
00359     }
00360 }
00361
00362 //-----
00363 AVRPeripheral stEINT_a =
00364 {
00365     EINT_Init,
00366     EINT_Read,
00367     EINT_Write,
00368     EINT_Clock,
00369     NULL,
00370     0x69,
00371     0x69
00372 };
00373
00374 //-----
00375 AVRPeripheral stEINT_b =
00376 {
00377     NULL,
00378     EINT_Read,
00379     EINT_Write,
00380     NULL,
00381     NULL,
00382     0x3C,
00383     0x3D
00384 };

```

## 4.141 src/peripheral/mega\_eint.h File Reference

ATMega External Interrupt Implementation.

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

### Variables

- AVRPeripheral **stEINT\_a**
- AVRPeripheral **stEINT\_b**

### 4.141.1 Detailed Description

ATMega External Interrupt Implementation.

Definition in file [mega\\_eint.h](#).

## 4.142 mega\_eint.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ )  )\ )  (      )\ ) |
00004 *      ((/ ( ((/ (      \      ( ((/ ( | -- [ Funkenstein ] -----
00005 *      / ( ) / ( ) ) (( ( ( ) \      / ( ) | -- [ Little ] -----
00006 *      ( ) _ ( )      )\ _ )\ ( ( ( ( ) ( ) | -- [ AVR ] -----
00007 *      | _ | |      ( ) _ ( ) \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ / _ \ \ V / | _ / | -- [ Runtime ] -----
00009 *      | _ | | _ | / / \ \ \ / | _ \ |
00010 *      |                                     | "Yeah, it does Arduino..."
00011 *      +-----+
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved

```



```

00013  *      See license.txt for details
00014  *****/
00021  #ifndef __MEGA_EINT_H__
00022  #define __MEGA_EINT_H__
00023
00024  #include "avr_peripheral.h"
00025
00026  extern AVRPeripheral stEINT_a;
00027  extern AVRPeripheral stEINT_b;
00028
00029  #endif //__MEGA_EINT_H__

```

## 4.143 src/peripheral/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**

### 4.143.1 Detailed Description

ATMega 16-bit timer implementation.

Definition in file [mega\\_timer16.c](#).

### 4.143.2 Enumeration Type Documentation

#### 4.143.2.1 ClockSource\_t

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 [42](#) of file [mega\\_timer16.c](#).

### 4.143.3 Function Documentation

#### 4.143.3.1 Timer16\_Clock()

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

! ToDo - Handle external timer generated events.

Definition at line [452](#) of file [mega\\_timer16.c](#).

### 4.143.4 Variable Documentation

#### 4.143.4.1 stTimer16

[AVRPeripheral](#) [stTimer16](#)

**Initial value:**

```
=  
{  
    Timer16_Init,  
    Timer16_Read,  
    Timer16_Write,  
    Timer16\_Clock,  
    0,  
    0x80,  
    0x8B  
}
```

Definition at line [584](#) of file [mega\\_timer16.c](#).

## 4.143.4.2 stTimer16a

AVRPeripheral stTimer16a

## Initial value:

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

Definition at line 596 of file [mega\\_timer16.c](#).

## 4.143.4.3 stTimer16b

AVRPeripheral stTimer16b

## Initial value:

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

Definition at line 608 of file [mega\\_timer16.c](#).

## 4.144 mega\_timer16.c

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( ((/(      \      | -- [ Funkenstein ] -----
00005 *      /(\) /(\) ((((\) ()\  | -- [ Little ] -----
00006 *      (\)_| (\) )\ _ )\ ((\ ((\ (\) | -- [ AVR ] -----
00007 *      | _ | | _ (\)_\ (\) \ / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ / _ \ \ / | _ \ | -- [ Runtime ] -----
00009 *      | _ | | _ | _ \ \ / | _ \ |
00010 *      | _ | | _ | _ \ \ / | _ \ |
00011 *      | _ | | _ | _ \ \ / | _ \ | "Yeah, it does Arduino..."
00012 *      | _ | | _ | _ \ \ / | _ \ |
00013 *      | _ | | _ | _ \ \ / | _ \ |
00014 *      | _ | | _ | _ \ \ / | _ \ |
00015 *      | _ | | _ | _ \ \ / | _ \ |
00016 *      | _ | | _ | _ \ \ / | _ \ |
00017 *      | _ | | _ | _ \ \ / | _ \ |
00018 *      | _ | | _ | _ \ \ / | _ \ |
00019 *      | _ | | _ | _ \ \ / | _ \ |
00020 *      | _ | | _ | _ \ \ / | _ \ |
00021 #include <stdio.h>
00022 #include <stdlib.h>
00023 #include <string.h>
00024 #include "avr_cpu.h"
00025 #include "avr_peripheral.h"
00026 #include "avr_periphregs.h"
00027 #include "avr_interrupt.h"
00028
00029 #if 1
```

```

00030 #define DEBUG_PRINT(...)
00031 #else
00032 #define DEBUG_PRINT printf
00033 #endif
00034
00035 //-----
00039 //-----
00040
00041 //-----
00042 typedef enum
00043 {
00044     CLK_SRC_OFF,
00045     CLK_SRC_DIV_1,
00046     CLK_SRC_DIV_8,
00047     CLK_SRC_DIV_64,
00048     CLK_SRC_DIV_256,
00049     CLK_SRC_DIV_1024,
00050     CLK_SRC_T1_FALL,
00051     CLK_SRC_T1_RISE
00052 } ClockSource_t;
00053
00054 //-----
00055 typedef enum
00056 {
00057     WGM_NORMAL,
00058     WGM_PWM_PC_8BIT,
00059     WGM_PWM_PC_9BIT,
00060     WGM_PWM_PC_10BIT,
00061     WGM CTC_OCR,
00062     WGM_PWM_8BIT,
00063     WGM_PWM_9BIT,
00064     WGM_PWM_10BIT,
00065     WGM_PWM_PC_FC_ICR,
00066     WGM_PWM_PC_FC_OCR,
00067     WGM_PWM_PC_ICR,
00068     WGM_PWM_PC_OCR,
00069     WGM CTC_ICR,
00070     WGM_RESERVED,
00071     WGM_FAST_PWM_ICR,
00072     WGM_FAST_PWM_OCR
00073 } WaveformGeneratorMode_t;
00074
00075 //-----
00076 typedef enum
00077 {
00078     COM_NORMAL,           // OCA1/B disconnected
00079     COM_TOGGLE_MATCH,     // Toggle on match
00080     COM_CLEAR_MATCH,
00081     COM_SET_MATCH
00082 } CompareOutputMode_t;
00083
00084 //-----
00085 static uint16_t u16DivCycles = 0;
00086 static uint16_t u16DivRemain = 0;
00087 static ClockSource_t eClockSource = CLK_SRC_OFF;
00088 static WaveformGeneratorMode_t eWGM = WGM_NORMAL;
00089 static CompareOutputMode_t eCOM1A = COM_NORMAL;
00090 static CompareOutputMode_t eCOM1B = COM_NORMAL;
00091
00092 //-----
00093 static uint8_t u8Temp; // The 8-bit temporary register used in 16-bit register accesses
00094 static uint16_t u8Count; // Internal 16-bit count register
00095
00096 //-----
00097 static void TCNT1_Increment()
00098 {
00099     uint16_t u16NewVal = 0;
00100
00101     u16NewVal = (stCPU.pstRAM->stRegisters.TCNT1H << 8) |
00102                stCPU.pstRAM->stRegisters.TCNT1L;
00103
00104     u16NewVal++;
00105     stCPU.pstRAM->stRegisters.TCNT1L = (u16NewVal & 0x00FF);
00106     stCPU.pstRAM->stRegisters.TCNT1H = (u16NewVal >> 8);
00107 }
00108
00109 //-----
00110 static uint16_t TCNT1_Read()
00111 {
00112     uint16_t u16Ret = 0;
00113
00114     u16Ret = (stCPU.pstRAM->stRegisters.TCNT1H << 8) |
00115             stCPU.pstRAM->stRegisters.TCNT1L;
00116     return u16Ret;
00117 }
00118
00119 //-----

```

```

00120 static void TCNT1_Clear()
00121 {
00122     stCPU.pstRAM->stRegisters.TCNT1H = 0;
00123     stCPU.pstRAM->stRegisters.TCNT1L = 0;
00124 }
00125
00126 //-----
00127 static uint16_t OCR1A_Read()
00128 {
00129     uint16_t u16Ret = 0;
00130
00131     u16Ret = (stCPU.pstRAM->stRegisters.OCR1AH << 8 ) |
00132             stCPU.pstRAM->stRegisters.OCR1AL;
00133     return u16Ret;
00134 }
00135
00136 //-----
00137 static uint16_t OCR1B_Read()
00138 {
00139     uint16_t u16Ret = 0;
00140
00141     u16Ret = (stCPU.pstRAM->stRegisters.OCR1BH << 8 ) |
00142             stCPU.pstRAM->stRegisters.OCR1BL;
00143     return u16Ret;
00144 }
00145
00146 //-----
00147 static uint16_t ICR1_Read()
00148 {
00149     uint16_t u16Ret = 0;
00150
00151     u16Ret = (stCPU.pstRAM->stRegisters.ICR1H << 8 ) |
00152             stCPU.pstRAM->stRegisters.ICR1L;
00153     return u16Ret;
00154 }
00155
00156 //-----
00157 static bool Timer16_Is_TOIE1_Enabled()
00158 {
00159     return (stCPU.pstRAM->stRegisters.TIMSK1.TOIE1 == 1);
00160 }
00161
00162 //-----
00163 static bool Timer16_Is_OCIE1A_Enabled()
00164 {
00165     return (stCPU.pstRAM->stRegisters.TIMSK1_OCIE1A == 1);
00166 }
00167
00168 //-----
00169 static bool Timer16_Is_OCIE1B_Enabled()
00170 {
00171     return (stCPU.pstRAM->stRegisters.TIMSK1_OCIE1B == 1);
00172 }
00173
00174 //-----
00175 static bool Timer16_Is_ICIE1_Enabled()
00176 {
00177     return (stCPU.pstRAM->stRegisters.TIMSK1_ICIE1 == 1);
00178 }
00179
00180 //-----
00181 static void OV1_Ack( uint8_t ucVector_)
00182 {
00183     stCPU.pstRAM->stRegisters.TIFR1.TOV1 = 0;
00184 }
00185
00186 //-----
00187 static void IC1_Ack( uint8_t ucVector_)
00188 {
00189     stCPU.pstRAM->stRegisters.TIFR1.ICF1 = 0;
00190 }
00191
00192 //-----
00193 static void COMPlA_Ack( uint8_t ucVector_)
00194 {
00195     static uint64_t lastcycles = 0;
00196     // printf("COMPlA - Ack'd: %d delta\n", stCPU.u64CycleCount - lastcycles);
00197     lastcycles = stCPU.u64CycleCount;
00198
00199     stCPU.pstRAM->stRegisters.TIFR1.OCF1A = 0;
00200 }
00201
00202 //-----
00203 static void COMPlB_Ack( uint8_t ucVector_)
00204 {
00205     stCPU.pstRAM->stRegisters.TIFR1.OCF1B = 0;
00206 }

```

```

00207
00208 //-----
00209 static void Timer16_Init(void *context_)
00210 {
00211     DEBUG_PRINT(stderr, "Timer16 Init\n");
00212
00213     CPU_RegisterInterruptCallback( OV1_Ack, stCPU.pstVectorMap->TIMER1_OVF);
00214     CPU_RegisterInterruptCallback( IC1_Ack, stCPU.pstVectorMap->TIMER1_CAPT);
00215     CPU_RegisterInterruptCallback( COMP1A_Ack, stCPU.pstVectorMap->
TIMER1_COMPA);
00216     CPU_RegisterInterruptCallback( COMP1B_Ack, stCPU.pstVectorMap->
TIMER1_COMPB);
00217 }
00218
00219 //-----
00220 static void Timer16_Read(void *context_, uint8_t ucAddr_, uint8_t *pucValue_)
00221 {
00222     DEBUG_PRINT(stderr, "Timer16 Read: 0x%02x\n", ucAddr_);
00223     *pucValue_ = stCPU.pstRAM->au8RAM[ ucAddr_ ];
00224 }
00225
00226 //-----
00227 static void TCCR1A_Write( uint8_t ucAddr_, uint8_t ucValue_)
00228 {
00229     // Update the waveform generator mode (WGM11:10) bits.
00230     uint8_t u8WGMBits = ucValue_ & 0x03; // WGM11 and 10 are in bits 0,1
00231     uint8_t u8WGMTemp = (uint8_t)eWGM;
00232     u8WGMTemp &= ~(0x03);
00233     u8WGMTemp |= u8WGMBits;
00234     eWGM = (WaveformGeneratorMode_t)u8WGMTemp;
00235
00236     // Update the memory-mapped register.
00237     stCPU.pstRAM->stRegisters.TCCR1A.r = ucValue_ & 0xF3;
00238 }
00239
00240 //-----
00241 static void TCCR1B_Write( uint8_t ucAddr_, uint8_t ucValue_)
00242 {
00243     // Update the waveform generator mode (WGM13:12) bits.
00244     uint8_t u8WGMBits = (ucValue_ >> 1) & 0x0C; // WGM13 and 12 are in register bits 3,4
00245     uint8_t u8WGMTemp = (uint8_t)eWGM;
00246     u8WGMTemp &= ~(0x0C);
00247     u8WGMTemp |= u8WGMBits;
00248     eWGM = (WaveformGeneratorMode_t)u8WGMTemp;
00249
00250     // Update the clock-select bits
00251     uint8_t u8ClockSource = ucValue_ & 0x07; // clock select is last 3 bits in reg
00252     eClockSource = (ClockSource_t)u8ClockSource;
00253     switch (eClockSource)
00254     {
00255     case CLK_SRC_DIV_1:
00256         u16DivCycles = 1;
00257         break;
00258     case CLK_SRC_DIV_8:
00259         u16DivCycles = 8;
00260         break;
00261     case CLK_SRC_DIV_64:
00262         u16DivCycles = 64;
00263         break;
00264     case CLK_SRC_DIV_256:
00265         u16DivCycles = 256;
00266         break;
00267     case CLK_SRC_DIV_1024:
00268         u16DivCycles = 1024;
00269         break;
00270     default:
00271         u16DivCycles = 0;
00272         break;
00273     }
00274
00275     // Update the memory-mapped register.
00276     stCPU.pstRAM->stRegisters.TCCR1B.r = ucValue_ & 0xDF; // Bit 5 is read-only
00277 }
00278
00279 //-----
00280 static void TCCR1C_Write( uint8_t ucAddr_, uint8_t ucValue_)
00281 {
00282     stCPU.pstRAM->stRegisters.TCCR1C.r = ucValue_;
00283 }
00284
00285 //-----
00286 static void TCNT1L_Write( uint8_t ucAddr_, uint8_t ucValue_)
00287 {
00288     // Writing the low-word forces the high-word to be stored from the internal
00289     // temp register... which is why the high byte must be written first.
00290     stCPU.pstRAM->stRegisters.TCNT1L = ucValue_;
00291     stCPU.pstRAM->stRegisters.TCNT1H = u8Temp;

```

```

00292 }
00293 //-----
00294 static void TCNT1H_Write( uint8_t ucAddr_, uint8_t ucValue_)
00295 {
00296     u8Temp = ucValue_;
00297 }
00298 //-----
00299 static void ICR1L_Write( uint8_t ucAddr_, uint8_t ucValue_)
00300 {
00301     // Writing the low-word forces the high-word to be stored from the internal
00302     // temp register... which is why the high byte must be written first.
00303     stCPU.pstRAM->stRegisters.ICR1L = ucValue_;
00304     stCPU.pstRAM->stRegisters.ICR1H = u8Temp;
00305 }
00306 //-----
00307 static void ICR1H_Write( uint8_t ucAddr_, uint8_t ucValue_)
00308 {
00309     u8Temp = ucValue_;
00310 }
00311 //-----
00312 static void OCR1AL_Write( uint8_t ucAddr_, uint8_t ucValue_)
00313 {
00314     // Writing the low-word forces the high-word to be stored from the internal
00315     // temp register... which is why the high byte must be written first.
00316     stCPU.pstRAM->stRegisters.OCR1AL = ucValue_;
00317     stCPU.pstRAM->stRegisters.OCR1AH = u8Temp;
00318 }
00319 //-----
00320 static void OCR1AH_Write( uint8_t ucAddr_, uint8_t ucValue_)
00321 {
00322     u8Temp = ucValue_;
00323 }
00324 //-----
00325 static void OCR1BL_Write( uint8_t ucAddr_, uint8_t ucValue_)
00326 {
00327     // Writing the low-word forces the high-word to be stored from the internal
00328     // temp register... which is why the high byte must be written first.
00329     stCPU.pstRAM->stRegisters.OCR1BL = ucValue_;
00330     stCPU.pstRAM->stRegisters.OCR1BH = u8Temp;
00331 }
00332 //-----
00333 static void OCR1BH_Write( uint8_t ucAddr_, uint8_t ucValue_)
00334 {
00335     u8Temp = ucValue_;
00336 }
00337 //-----
00338 static void Timer16_IntFlagUpdate(void)
00339 {
00340     if (stCPU.pstRAM->stRegisters.TIMSK1.TOIE1 == 1)
00341     {
00342         if (stCPU.pstRAM->stRegisters.TIFR1.TOV1 == 1)
00343         {
00344             DEBUG_PRINT(" TOV1 Interrupt Candidate\n" );
00345             AVR_InterruptCandidate(stCPU.pstVectorMap->TIMER1_OVF);
00346         }
00347         else
00348         {
00349             AVR_ClearCandidate(stCPU.pstVectorMap->TIMER1_OVF);
00350         }
00351     }
00352     if (stCPU.pstRAM->stRegisters.TIMSK1.OCF1A == 1)
00353     {
00354         if (stCPU.pstRAM->stRegisters.TIFR1.OCF1A == 1)
00355         {
00356             DEBUG_PRINT(" OCF1A Interrupt Candidate\n" );
00357             AVR_InterruptCandidate(stCPU.pstVectorMap->TIMER1_COMPA);
00358         }
00359         else
00360         {
00361             AVR_ClearCandidate(stCPU.pstVectorMap->TIMER1_COMPA);
00362         }
00363     }
00364     if (stCPU.pstRAM->stRegisters.TIMSK1.OCF1B == 1)
00365     {
00366         if (stCPU.pstRAM->stRegisters.TIFR1.OCF1B == 1)
00367         {
00368             DEBUG_PRINT(" OCF1B Interrupt Candidate\n" );
00369             AVR_InterruptCandidate(stCPU.pstVectorMap->TIMER1_COMPB);
00370         }
00371         else
00372     }

```



```

00379     {
00380         AVR_ClearCandidate(stCPU.pstVectorMap->TIMER1_COMPB);
00381     }
00382 }
00383
00384 if (stCPU.pstRAM->stRegisters.TIMSK1.ICIE1 == 1)
00385 {
00386     if (stCPU.pstRAM->stRegisters.TIFR1.ICF1 == 1)
00387     {
00388         DEBUG_PRINT(" ICF1 Interrupt Candidate\n" );
00389         AVR_InterruptCandidate(stCPU.pstVectorMap->TIMER1_CAPT);
00390     }
00391     else
00392     {
00393         AVR_ClearCandidate(stCPU.pstVectorMap->TIMER1_CAPT);
00394     }
00395 }
00396 }
00397
00398 //-----
00399 // TIFR & TMSK
00400 static void Timer16b_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_)
00401 {
00402     stCPU.pstRAM->au8RAM[ucAddr_] = ucValue_;
00403     Timer16_IntFlagUpdate();
00404 }
00405
00406 //-----
00407 static void Timer16_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_)
00408 {
00409     switch (ucAddr_)
00410     {
00411         case 0x80: //TCCR1A
00412             TCCR1A_Write(ucAddr_, ucValue_);
00413             break;
00414         case 0x81: //TCCR1B
00415             TCCR1B_Write(ucAddr_, ucValue_);
00416             break;
00417         case 0x82: //TCCR1C
00418             TCCR1C_Write(ucAddr_, ucValue_);
00419             break;
00420         case 0x83: // Reserved
00421             break;
00422         case 0x84: // TCNT1L
00423             TCNT1L_Write(ucAddr_, ucValue_);
00424             break;
00425         case 0x85: // TCNT1H
00426             TCNT1H_Write(ucAddr_, ucValue_);
00427             break;
00428         case 0x86: // ICR1L
00429             ICR1L_Write(ucAddr_, ucValue_);
00430             break;
00431         case 0x87: // ICR1H
00432             ICR1H_Write(ucAddr_, ucValue_);
00433             break;
00434         case 0x88: // OCR1AL
00435             OCR1AL_Write(ucAddr_, ucValue_);
00436             break;
00437         case 0x89: // OCR1AH
00438             OCR1AH_Write(ucAddr_, ucValue_);
00439             break;
00440         case 0x8A: // OCR1BL
00441             OCR1BL_Write(ucAddr_, ucValue_);
00442             break;
00443         case 0x8B: // OCR1BH
00444             OCR1BH_Write(ucAddr_, ucValue_);
00445             break;
00446         default:
00447             break;
00448     }
00449 }
00450
00451 //-----
00452 static void Timer16_Clock(void *context_)
00453 {
00454     if (eClockSource == CLK_SRC_OFF)
00455     {
00456         return;
00457     }
00458
00459     // Handle clock division logic
00460     bool bUpdateTimer = false;
00461     switch (eClockSource)
00462     {
00463         case CLK_SRC_DIV_1:
00464         case CLK_SRC_DIV_8:
00465         case CLK_SRC_DIV_64:

```

```

00466     case CLK_SRC_DIV_256:
00467     case CLK_SRC_DIV_1024:
00468     {
00469         // Decrement the clock-divide value
00470         if (ul6DivRemain)
00471         {
00472             //DEBUG_PRINT(" %d ticks remain\n", ul6DivRemain);
00473             ul6DivRemain--;
00474         }
00475
00476         if (!ul6DivRemain)
00477         {
00478             // clock-divider count hits zero, reset and trigger an update.
00479             //DEBUG_PRINT(" expire and reset\n");
00480             if (ul6DivCycles)
00481             {
00482                 ul6DivRemain = ul6DivCycles;
00483                 bUpdateTimer = true;
00484             }
00485         }
00486     }
00487     break;
00488 default:
00489     break;
00490 }
00491
00492
00493
00494 if (bUpdateTimer)
00495 {
00496     // Handle event flags on timer updates
00497     bool bOVF = false;
00498     bool bCTCA = false;
00499     bool bCTCB = false;
00500     bool bICR = false;
00501     bool bIntr = false;
00502
00503     //DEBUG_PRINT(" WGM Mode %d\n", eWGM );
00504     switch (eWGM)
00505     {
00506     case WGM_NORMAL:
00507     {
00508         DEBUG_PRINT(" Update Normal\n");
00509         TCNT1_Increment();
00510         if (TCNT1_Read() == 0)
00511         {
00512             bOVF = true;
00513         }
00514     }
00515     break;
00516     case WGM CTC_OCR:
00517     {
00518         DEBUG_PRINT(" Update CTC\n");
00519         TCNT1_Increment();
00520         if (TCNT1_Read() == 0)
00521         {
00522             bOVF = true;
00523         }
00524         else
00525         {
00526             bool bClearTCNT1 = false;
00527             if (TCNT1_Read() == OCR1A_Read())
00528             {
00529                 DEBUG_PRINT(" CTC1A Match\n" );
00530                 bCTCA = true;
00531                 bClearTCNT1 = true;
00532             }
00533             if (TCNT1_Read() == ICR1_Read())
00534             {
00535                 DEBUG_PRINT(" ICR1 Match\n" );
00536                 bICR = true;
00537                 bClearTCNT1 = true;
00538             }
00539             if (bClearTCNT1)
00540             {
00541                 TCNT1_Clear();
00542             }
00543         }
00544     }
00545     break;
00546 default:
00547     break;
00548 }
00549
00550 // Set interrupt flags if an appropriate transition has taken place
00551 if (bOVF)
00552 {
00553     DEBUG_PRINT(" TOV1 Set\n" );

```

```

00554         stCPU.pstRAM->stRegisters.TIFR1.TOV1 = 1;
00555         bIntr = true;
00556     }
00557     if (bCTCA)
00558     {
00559         DEBUG_PRINT(" OCF1A Set\n" );
00560         stCPU.pstRAM->stRegisters.TIFR1.OCF1A = 1;
00561         bIntr = true;
00562     }
00563     if (bCTCB)
00564     {
00565         DEBUG_PRINT(" OCF1B Set\n" );
00566         stCPU.pstRAM->stRegisters.TIFR1.OCF1B = 1;
00567         bIntr = true;
00568     }
00569     if (bICR)
00570     {
00571         DEBUG_PRINT(" ICF1 Set\n" );
00572         stCPU.pstRAM->stRegisters.TIFR1.ICF1 = 1;
00573         bIntr = true;
00574     }
00575
00576     if (bIntr)
00577     {
00578         Timer16_IntFlagUpdate();
00579     }
00580 }
00581 }
00582
00583 //-----
00584 AVRPeripheral stTimer16 =
00585 {
00586     Timer16_Init,
00587     Timer16_Read,
00588     Timer16_Write,
00589     Timer16_Clock,
00590     0,
00591     0x80,
00592     0x8B
00593 };
00594
00595 //-----
00596 AVRPeripheral stTimer16a =
00597 {
00598     0,
00599     Timer16_Read,
00600     Timer16b_Write,
00601     0,
00602     0,
00603     0x36,
00604     0x36
00605 };
00606
00607 //-----
00608 AVRPeripheral stTimer16b =
00609 {
00610     0,
00611     Timer16_Read,
00612     Timer16b_Write,
00613     0,
00614     0,
00615     0x6F,
00616     0x6F
00617 };

```

## 4.145 src/peripheral/mega\_timer16.h File Reference

ATMega 16-bit timer implementation.

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

### Variables

- AVRPeripheral stTimer16
- AVRPeripheral stTimer16a
- AVRPeripheral stTimer16b

### 4.145.1 Detailed Description

ATMega 16-bit timer implementation.

Definition in file [mega\\_timer16.h](#).

## 4.146 mega\_timer16.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( ((/(      \      | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( ) \      | -- [ Little ] -----
00006 *      ( ) _ ( )      ) \ ) \ ( ( ( ) _ ( )      | -- [ AVR ] -----
00007 *      | _ | |      ( ) _ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ v / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _ _ / _ \ \ \ \ / | _ \      |
00010 *      | _ | | _ _ _ / _ \ \ \ \ / | _ \      | "Yeah, it does Arduino..."
00011 * -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __MEGA_TIMER16_H__
00022 #define __MEGA_TIMER16_H__
00023
00024 #include "avr_peripheral.h"
00025
00026 extern AVRPeripheral stTimer16;
00027 extern AVRPeripheral stTimer16a;
00028 extern AVRPeripheral stTimer16b;
00029
00030 #endif //__MEGA_EINT_H__

```

## 4.147 src/peripheral/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**

#### 4.147.1 Detailed Description

ATMega 8-bit timer implementation.

Definition in file [mega\\_timer8.c](#).

#### 4.147.2 Enumeration Type Documentation

##### 4.147.2.1 ClockSource\_t

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 [42](#) of file [mega\\_timer8.c](#).

#### 4.147.3 Function Documentation

##### 4.147.3.1 Timer8\_Clock()

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

! ToDo - Handle external timer generated events.

Definition at line [319](#) of file [mega\\_timer8.c](#).

#### 4.147.4 Variable Documentation

##### 4.147.4.1 stTimer8

[AVRPeripheral](#) [stTimer8](#)

**Initial value:**

```
=  
{  
    Timer8_Init,  
    Timer8_Read,  
    Timer8_Write,  
    Timer8\_Clock,  
    0,  
    0x44,  
    0x48  
}
```

Definition at line [432](#) of file [mega\\_timer8.c](#).

## 4.147.4.2 stTimer8a

AVRPeripheral stTimer8a

Initial value:

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

Definition at line 445 of file mega\_timer8.c.

## 4.147.4.3 stTimer8b

AVRPeripheral stTimer8b

Initial value:

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

Definition at line 457 of file mega\_timer8.c.

## 4.148 mega\_timer8.c

```
00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      )\ )      |
00004 *      ((/( ()/(      \      (      ((/(      | -- [ Funkenstein ] -----
00005 *      /( ) )/( ) )((( ( )\      )\      / ( )      | -- [ Little ] -----
00006 *      ( ) _ ( )      )\ _ )\      ( ( ) ( ( ) ( )      | -- [ AVR ] -----
00007 *      | _ | | _      ( ) _ ( ) \ \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ \ / / | _ \      | -- [ Runtime ] -----
00009 *      | _ | | _      / _ \ \ \ / / | _ \      |
00010 *      | _ | | _      / _ \ \ \ / / | _ \      | "Yeah, it does Arduino..."
00011 *      -----+-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #include <stdio.h>
00022 #include <stdlib.h>
00023 #include <string.h>
00024 #include "avr_cpu.h"
00025 #include "avr_peripheral.h"
00026 #include "avr_periphregs.h"
00027 #include "avr_interrupt.h"
00028
00029 #if 1
```

```

00030 #define DEBUG_PRINT(...)
00031 #else
00032 #define DEBUG_PRINT printf
00033 #endif
00034
00035 //-----
00039 //-----
00040
00041 //-----
00042 typedef enum
00043 {
00044     CLK_SRC_OFF,
00045     CLK_SRC_DIV_1,
00046     CLK_SRC_DIV_8,
00047     CLK_SRC_DIV_64,
00048     CLK_SRC_DIV_256,
00049     CLK_SRC_DIV_1024,
00050     CLK_SRC_T1_FALL,
00051     CLK_SRC_T1_RISE
00052 } ClockSource_t;
00053
00054 //-----
00055 typedef enum
00056 {
00057     WGM_NORMAL,
00058     WGM_PWM_PC_FF,
00059     WGM CTC_OCR,
00060     WGM_FAST_PWM_FF,
00061     WGM_RESERVED_1, // Not a valid mode
00062     WGM_PWM_PC_OCR,
00063     WGM_RESERVED_2, // Not a valid mode
00064     WGM_FAST_PWM_OCR
00065 } WaveformGeneratorMode_t;
00066
00067 //-----
00068 typedef enum
00069 {
00070     COM_NORMAL, // OCA
00071     COM_TOGGLE_MATCH, // Toggle on match
00072     COM_CLEAR_MATCH,
00073     COM_SET_MATCH
00074 } CompareOutputMode_t;
00075
00076 //-----
00077 static uint16_t u16DivCycles = 0;
00078 static uint16_t u16DivRemain = 0;
00079 static ClockSource_t eClockSource = CLK_SRC_OFF;
00080 static WaveformGeneratorMode_t eWGM = WGM_NORMAL;
00081 static CompareOutputMode_t eCOM1A = COM_NORMAL;
00082 static CompareOutputMode_t eCOM1B = COM_NORMAL;
00083
00084 //-----
00085 static uint8_t u8Temp; // The 8-bit temporary register used in 16-bit register accesses
00086 static uint16_t u8Count; // Internal 16-bit count register
00087
00088 //-----
00089 static void TCNT0_Increment()
00090 {
00091     stCPU.pstRAM->stRegisters.TCNT0++;
00092 }
00093
00094 //-----
00095 static uint8_t TCNT0_Read()
00096 {
00097     return stCPU.pstRAM->stRegisters.TCNT0;
00098 }
00099
00100 //-----
00101 static void TCNT0_Clear()
00102 {
00103     stCPU.pstRAM->stRegisters.TCNT0 = 0;
00104 }
00105
00106 //-----
00107 static uint8_t OCR0A_Read()
00108 {
00109     return stCPU.pstRAM->stRegisters.OCR0A;
00110 }
00111
00112 //-----
00113 static uint8_t OCR0B_Read()
00114 {
00115     return stCPU.pstRAM->stRegisters.OCR0B;
00116 }
00117
00118 //-----
00119 static bool Timer8_Is_TOIE0_Enabled()

```



```

00120 {
00121     return (stCPU.pstRAM->stRegisters.TIMSK0.TOIE0 == 1);
00122 }
00123
00124 //-----
00125 static bool Timer8_Is_OCIE0A_Enabled()
00126 {
00127     return (stCPU.pstRAM->stRegisters.TIMSK0_OCIE0A == 1);
00128 }
00129
00130 //-----
00131 static bool Timer8_Is_OCIE1B_Enabled()
00132 {
00133     return (stCPU.pstRAM->stRegisters.TIMSK0_OCIE0B == 1);
00134 }
00135
00136 //-----
00137 static void OV0_Ack( uint8_t ucVector_)
00138 {
00139     static uint64_t lastcycles = 0;
00140     stCPU.pstRAM->stRegisters.TIFR0.TOV0 = 0;
00141     // printf("OV0 - Ack'd: %d delta\n", stCPU.u64CycleCount - lastcycles);
00142     lastcycles = stCPU.u64CycleCount;
00143 }
00144
00145 //-----
00146 static void COMP0A_Ack( uint8_t ucVector_)
00147 {
00148     stCPU.pstRAM->stRegisters.TIFR0.OCF0A = 0;
00149 }
00150
00151 //-----
00152 static void COMP0B_Ack( uint8_t ucVector_)
00153 {
00154     stCPU.pstRAM->stRegisters.TIFR0.OCF0B = 0;
00155 }
00156
00157 //-----
00158 static void Timer8_Init(void *context_)
00159 {
00160     DEBUG_PRINT( "Timer8 Init\n");
00161     CPU_RegisterInterruptCallback( OV0_Ack, stCPU.pstVectorMap->TIMER0_OVF);
00162     CPU_RegisterInterruptCallback( COMP0A_Ack, stCPU.pstVectorMap->
TIMER0_COMPA);
00163     CPU_RegisterInterruptCallback( COMP0B_Ack, stCPU.pstVectorMap->
TIMER0_COMPB);
00164 }
00165
00166 //-----
00167 static void Timer8_Read(void *context_, uint8_t ucAddr_, uint8_t *pucValue_)
00168 {
00169     DEBUG_PRINT( "Timer8 Read: 0x%02x\n", ucAddr_);
00170     *pucValue_ = stCPU.pstRAM->au8RAM[ ucAddr_ ];
00171 }
00172
00173 //-----
00174 static void TCCR0A_Write( uint8_t ucAddr_, uint8_t ucValue_)
00175 {
00176     // Update the waveform generator mode (WGM1:0) bits.
00177     uint8_t u8WGMBits = ucValue_ & 0x03; // WGM1 and 0 are in bits 0,1
00178     uint8_t u8WGMTemp = (uint8_t)eWGM;
00179     u8WGMTemp &= ~(0x03);
00180     u8WGMTemp |= u8WGMBits;
00181     eWGM = (WaveformGeneratorMode_t)u8WGMTemp;
00182
00183     // Update the memory-mapped register.
00184     stCPU.pstRAM->stRegisters.TCCR0A.r = ucValue_ & 0xF3;
00185 }
00186
00187 //-----
00188 static void TCCR0B_Write( uint8_t ucAddr_, uint8_t ucValue_)
00189 {
00190     // Update the waveform generator mode (WGM2) bit
00191     uint8_t u8WGMBits = (ucValue_ >> 1) & 0x04; // WGM2 is in bit 3 of the register
00192     uint8_t u8WGMTemp = (uint8_t)eWGM;
00193     u8WGMTemp &= ~(0x04);
00194     u8WGMTemp |= u8WGMBits;
00195     eWGM = (WaveformGeneratorMode_t)u8WGMTemp;
00196
00197     // Update the clock-select bits
00198     uint8_t u8ClockSource = ucValue_ & 0x07; // clock select is last 3 bits in reg
00199     eClockSource = (ClockSource_t)u8ClockSource;
00200     switch (eClockSource)
00201     {
00202     case CLK_SRC_DIV_1:
00203         u16DivCycles = 1;
00204         break;

```

```

00205     case CLK_SRC_DIV_8:
00206         u16DivCycles = 8;
00207         break;
00208     case CLK_SRC_DIV_64:
00209         u16DivCycles = 64;
00210         break;
00211     case CLK_SRC_DIV_256:
00212         u16DivCycles = 256;
00213         break;
00214     case CLK_SRC_DIV_1024:
00215         u16DivCycles = 1024;
00216         break;
00217     default:
00218         u16DivCycles = 0;
00219         break;
00220 }
00221 DEBUG_PRINT(" ClockSource = %d, %d cycles\n", eClockSource, u16DivCycles);
00222 // Update the memory-mapped register.
00223 stCPU.pstRAM->stRegisters.TCCR0B.r = ucValue_ & 0xCF; // Bit 5&6 are read-only
00224 }
00225
00226 //-----
00227 static void TCNT0_Write( uint8_t ucAddr_, uint8_t ucValue_)
00228 {
00229     stCPU.pstRAM->stRegisters.TCNT0 = ucValue_;
00230 }
00231
00232 //-----
00233 static void OCR0A_Write( uint8_t ucAddr_, uint8_t ucValue_)
00234 {
00235     stCPU.pstRAM->stRegisters.OCR0A = ucValue_;
00236 }
00237
00238 //-----
00239 static void OCR0B_Write( uint8_t ucAddr_, uint8_t ucValue_)
00240 {
00241     stCPU.pstRAM->stRegisters.OCR0B = ucValue_;
00242 }
00243
00244 //-----
00245 static void Timer8_IntFlagUpdate(void)
00246 {
00247     if (stCPU.pstRAM->stRegisters.TIMSK0.TOIE0 == 1)
00248     {
00249         if (stCPU.pstRAM->stRegisters.TIFR0.TOV0 == 1)
00250         {
00251             DEBUG_PRINT(" TOV0 Interrupt Candidate\n" );
00252             AVR_InterruptCandidate(stCPU.pstVectorMap->TIMER0_OVF);
00253         }
00254         else
00255         {
00256             AVR_ClearCandidate(stCPU.pstVectorMap->TIMER0_OVF);
00257         }
00258     }
00259     if (stCPU.pstRAM->stRegisters.TIMSK0.OCF0A == 1)
00260     {
00261         if (stCPU.pstRAM->stRegisters.TIFR0.OCF0A == 1)
00262         {
00263             DEBUG_PRINT(" OCF0A Interrupt Candidate\n" );
00264             AVR_InterruptCandidate(stCPU.pstVectorMap->TIMER0_COMPA);
00265         }
00266         else
00267         {
00268             AVR_ClearCandidate(stCPU.pstVectorMap->TIMER0_COMPA);
00269         }
00270     }
00271     if (stCPU.pstRAM->stRegisters.TIMSK0.OCF0B == 1)
00272     {
00273         if (stCPU.pstRAM->stRegisters.TIFR0.OCF0B == 1)
00274         {
00275             DEBUG_PRINT(" OCF0B Interrupt Candidate\n" );
00276             AVR_InterruptCandidate(stCPU.pstVectorMap->TIMER0_COMPB);
00277         }
00278         else
00279         {
00280             AVR_ClearCandidate(stCPU.pstVectorMap->TIMER0_COMPB);
00281         }
00282     }
00283 }
00284
00285 //-----
00286 static void Timer8b_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_)
00287 {
00288     stCPU.pstRAM->au8RAM[ucAddr_] = ucValue_;
00289     Timer8_IntFlagUpdate();
00290 }
00291

```

```

00292 //-----
00293 static void Timer8_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
00294 {
00295     DEBUG_PRINT("Timer8_Write: %d=%d\n", ucAddr_, ucValue_);
00296     switch (ucAddr_)
00297     {
00298     case 0x44: //TCCR1A
00299         TCCR0A_Write(ucAddr_, ucValue_);
00300         break;
00301     case 0x45: //TCCR1B
00302         TCCR0B_Write(ucAddr_, ucValue_);
00303         break;
00304     case 0x46: // TCNT0
00305         TCNT0_Write(ucAddr_, ucValue_);
00306         break;
00307     case 0x47: // OCR0A
00308         OCR0A_Write(ucAddr_, ucValue_);
00309         break;
00310     case 0x48: // OCR0B
00311         OCR0B_Write(ucAddr_, ucValue_);
00312         break;
00313     default:
00314         break;
00315     }
00316 }
00317 //-----
00318 static void Timer8_Clock(void *context_ )
00319 {
00320     if (eClockSource == CLK_SRC_OFF)
00321     {
00322         return;
00323     }
00324     // Handle clock division logic
00325     bool bUpdateTimer = false;
00326     switch (eClockSource)
00327     {
00328     case CLK_SRC_DIV_1:
00329     case CLK_SRC_DIV_8:
00330     case CLK_SRC_DIV_64:
00331     case CLK_SRC_DIV_256:
00332     case CLK_SRC_DIV_1024:
00333     {
00334         // Decrement the clock-divide value
00335         if (ul6DivRemain)
00336         {
00337             //DEBUG_PRINT(" %d ticks remain\n", ul6DivRemain);
00338             ul6DivRemain--;
00339         }
00340         if (!ul6DivRemain)
00341         {
00342             // clock-divider count hits zero, reset and trigger an update.
00343             DEBUG_PRINT(" expire and reset\n");
00344             if (ul6DivCycles)
00345             {
00346                 ul6DivRemain = ul6DivCycles;
00347                 bUpdateTimer = true;
00348             }
00349         }
00350     }
00351     break;
00352     default:
00353     break;
00354 }
00355 if (bUpdateTimer)
00356 {
00357     // Handle event flags on timer updates
00358     bool bOVF = false;
00359     bool bCTCA = false;
00360     bool bCTCB = false;
00361     bool bIntr = false;
00362     switch (eWGM)
00363     {
00364     case WGM_NORMAL:
00365     {
00366         DEBUG_PRINT(" Update Normal, TCNT = %d\n", TCNT0_Read());
00367         TCNT0_Increment();
00368         if (TCNT0_Read() == 0)
00369         {
00370             bOVF = true;
00371         }
00372     }
00373     }
00374 }

```

```

00380         break;
00381     case WGM_CTC_OCR:
00382     {
00383         DEBUG_PRINT(" Update CTC\n");
00384         TCNT0_Increment();
00385         if (TCNT0_Read() == 0)
00386         {
00387             bOVF = true;
00388         }
00389         else
00390         {
00391             if (TCNT0_Read() == OCR0A_Read())
00392             {
00393                 DEBUG_PRINT(" CTC0A Match\n" );
00394                 bCTCA = true;
00395                 TCNT0_Clear();
00396             }
00397         }
00398     }
00399     break;
00400 default:
00401     break;
00402 }
00403
00404 // Set interrupt flags if an appropriate transition has taken place
00405 if (bOVF)
00406 {
00407     DEBUG_PRINT(" TOV0 Set\n" );
00408     stCPU.pstRAM->stRegisters.TIFR0.TOV0 = 1;
00409     bIntr = true;
00410 }
00411 if (bCTCA)
00412 {
00413     DEBUG_PRINT(" OCF0A Set\n" );
00414     stCPU.pstRAM->stRegisters.TIFR0.OCF0A = 1;
00415     bIntr = true;
00416 }
00417 if (bCTCB)
00418 {
00419     DEBUG_PRINT(" OCF0B Set\n" );
00420     stCPU.pstRAM->stRegisters.TIFR0.OCF0B = 1;
00421     bIntr = true;
00422 }
00423
00424 if (bIntr)
00425 {
00426     Timer8_IntFlagUpdate();
00427 }
00428 }
00429 }
00430
00431 //-----
00432 AVRPeripheral stTimer8 =
00433 {
00434     Timer8_Init,
00435     Timer8_Read,
00436     Timer8_Write,
00437     Timer8_Clock,
00438     0,
00439     0x44,
00440     0x48
00441 };
00442
00443
00444 //-----
00445 AVRPeripheral stTimer8a =
00446 {
00447     0,
00448     Timer8_Read,
00449     Timer8b_Write,
00450     0,
00451     0,
00452     0x35,
00453     0x35
00454 };
00455
00456 //-----
00457 AVRPeripheral stTimer8b =
00458 {
00459     0,
00460     Timer8_Read,
00461     Timer8b_Write,
00462     0,
00463     0,
00464     0x6E,
00465     0x6E
00466 };

```

## 4.149 src/peripheral/mega\_timer8.h File Reference

ATMega 8-bit timer implementation.

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

### Variables

- AVRPeripheral stTimer8
- AVRPeripheral stTimer8a
- AVRPeripheral stTimer8b

### 4.149.1 Detailed Description

ATMega 8-bit timer implementation.

Definition in file [mega\\_timer8.h](#).

## 4.150 mega\_timer8.h

```
00001 /*****
00002 *      (      (      (      (      (      (      (      (      (      (      (
00003 *      )\ )  )\ )  )\ )  )\ )  )\ )  )\ )  )\ )  )\ )  )\ )  )\ )  )\ )  )\ )
00004 *      ((/( ((/( ((/( ((/( ((/( ((/( ((/( ((/( ((/( ((/( ((/( ((/( ((/( ((/( ((/(
00005 *      /( )  /( )  /( )  /( )  /( )  /( )  /( )  /( )  /( )  /( )  /( )  /( )  /( )
00006 *      ( )_ | ( ) )\ _ )\ _ )\ _ )\ _ )\ _ )\ _ )\ _ )\ _ )\ _ )\ _ )\ _ )\ _ )\ _
00007 *      | _ | | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _
00008 *      | _ | | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _
00009 *      | _ | | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _
00010 *      | _ | | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _
00011 *      | _ | | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _
00012 *      (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 *      See license.txt for details
00014 *****/
00021 #ifndef __MEGA_TIMER8_H__
00022 #define __MEGA_TIMER8_H__
00023
00024 #include "avr_peripheral.h"
00025
00026 extern AVRPeripheral stTimer8;
00027 extern AVRPeripheral stTimer8a;
00028 extern AVRPeripheral stTimer8b;
00029
00030 #endif //__MEGA_EINT_H__
```

## 4.151 src/peripheral/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"
#include "options.h"
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

## Macros

- `#define DEBUG\_PRINT(...)`

*Plugin must interface with the following registers:*

## Functions

- static void **UART\_BeginServer** (void)
- 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 **use\_uart\_socket** = false
- static int **listener\_socket** = 0
- static int **uart\_socket** = 0
- 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**

### 4.151.1 Detailed Description

Implements an atmega UART plugin.

Definition in file [mega\\_uart.c](#).

### 4.151.2 Macro Definition Documentation

#### 4.151.2.1 DEBUG\_PRINT

```
#define DEBUG_PRINT(  
    ... )
```

Plugin must interface with the following registers:

UDRn UCSRnA UCSRnB UCSRnC UBBRnL UBBRnH

Definition at line 43 of file [mega\\_uart.c](#).

### 4.151.3 Variable Documentation

#### 4.151.3.1 stUART

[AVRPeripheral](#) stUART

**Initial value:**

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

Definition at line 641 of file [mega\\_uart.c](#).

## 4.152 mega\_uart.c

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      (      |
00004 *      ((/( (/(      )\      | -- [ Funkenstein ] -----
00005 *      /( ) / ( ) ((( ( )\      | -- [ Litle ] -----
00006 *      ( ) _ ( )      )\ _ )\ ( ( ( )      | -- [ AVR ] -----
00007 *      | _ | |      ( ) _ ( ) \ / / | _ \      | -- [ Virtual ] -----
00008 *      | _ | | _      / _ \ \ / / | _ /      | -- [ Runtime ] -----
00009 *      | _ | | _ _ | / _ \ \ / \ | _ \      |
00010 *      | "Yeah, it does Arduino..."
00011 * -----
00012 * (c) Copyright 2014-17, 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 #include "options.h"
00041
00042 #if 1
00043 #define DEBUG_PRINT(...)
00044 #else
00045 #define DEBUG_PRINT printf
00046 #endif
00047
00048 //-----
00049 static bool use_uart_socket = false;
00050
00051 #if _WIN32
00052 #include <io.h>
00053 #include <WinSock2.h>
00054 #include <WS2tcpip.h>
00055
00056 static SOCKET listener_socket = INVALID_SOCKET;
00057 static SOCKET uart_socket = INVALID_SOCKET;
00058
00059 #pragma comment(lib, "Ws2_32.lib")
00060 static WSADATA ws;
00061
00062 //-----
00063 static void UART_BeginServer(void)
00064 {
00065     int err;
00066
00067     struct addrinfo *localaddr = 0;
00068     struct addrinfo hints = { 0 };
00069
00070     do
00071     {
00072         // Initialize winsock prior to use.
00073         err = WSStartup(MAKEWORD(2,2), &ws);
00074         if (0 != err)
00075         {
00076             DEBUG_PRINT(stderr, "Error initializing winsock - bailing\n");
00077             break;
00078         }
00079
00080         // Figure out what address to use for our server, specifying we want TCP/IP
00081         hints.ai_family = AF_INET;
00082         hints.ai_protocol = IPPROTO_TCP;
00083         hints.ai_socktype = SOCK_STREAM;
00084         hints.ai_flags = AI_PASSIVE;
00085
00086         const char *portnum = Options_GetByName("--uart");
00087         if (!portnum)
00088         {
00089             portnum = "4444";
00090         }
00091
00092         err = getaddrinfo(NULL, portnum, &hints, &localaddr);
00093         if (0 != err)
00094         {
00095             DEBUG_PRINT(stderr, "Error getting address info - bailing\n");
00096             break;
00097         }
00098
00099         // Create a socket to listen for UART connections
00100         listener_socket = socket(localaddr->ai_family, localaddr->ai_socktype, localaddr->ai_protocol);
00101         if (INVALID_SOCKET == listener_socket)
00102         {

```



```

00103         DEBUG_PRINT(stderr, "Error creating socket - bailing\n" );
00104         err = -1;
00105         break;
00106     }
00107
00108     // Setup the TCP listening socket
00109     if (SOCKET_ERROR == bind(listener_socket, localaddr->ai_addr, (int)localaddr->ai_addrlen))
00110     {
00111         DEBUG_PRINT(stderr, "Error on socket bind - bailing\n");
00112         err = -1;
00113         break;
00114     }
00115
00116     if (SOCKET_ERROR == listen(listener_socket, SOMAXCONN))
00117     {
00118         DEBUG_PRINT(stderr, "Error on socket listen - bailing\n");
00119         err = -1;
00120         break;
00121     }
00122
00123     printf("[Waiting for incoming conneciton on port %s]\n", portnum);
00124     uart_socket = accept(listener_socket, NULL, NULL);
00125     if (INVALID_SOCKET == uart_socket)
00126     {
00127         DEBUG_PRINT(stderr, "Error on socket accept - bailing\n");
00128         err = -1;
00129         break;
00130     }
00131
00132     unsigned long mode = 1;
00133     int rc = ioctlsocket(uart_socket, FIONBIO, &mode);
00134     if (NO_ERROR != rc) {
00135         DEBUG_PRINT(stderr, "Error setting non-blocking\n");
00136         err = -1;
00137         break;
00138     }
00139
00140     } while(0);
00141
00142     if (localaddr)
00143     {
00144         freeaddrinfo(localaddr);
00145     }
00146
00147     if (0 != err)
00148     {
00149         if (INVALID_SOCKET != listener_socket)
00150         {
00151             closesocket(listener_socket);
00152         }
00153         if (INVALID_SOCKET != uart_socket)
00154         {
00155             closesocket(uart_socket);
00156         }
00157         WSACleanup();
00158         exit(-1);
00159     }
00160
00161     printf("[UART Connected!]\n");
00162 }
00163 #else
00164 #include <fcntl.h>
00165 #include <sys/types.h>
00166 #include <sys/socket.h>
00167 #include <netinet/in.h>
00168
00169 static int  listener_socket = 0;
00170 static int  uart_socket     = 0;
00171
00172 //-----
00173 static void UART_BeginServer(void)
00174 {
00175     fprintf(stderr, "[Initializing UART socket]");
00176
00177     const char *port_string = Options_GetByName("--uart");
00178     if (!port_string)
00179     {
00180         port_string = "4444";
00181     }
00182     int portnum = atoi(port_string);
00183
00184     listener_socket = socket(AF_INET, SOCK_STREAM, 0);
00185     if (listener_socket <= 0)
00186     {
00187         fprintf(stderr, "Error creating socket on port %s, bailing\n", port_string );
00188         exit(-1);
00189     }

```

```

00190
00191     struct sockaddr_in serv_addr = { 0 };
00192     struct sockaddr_in cli_addr = { 0 };
00193
00194     serv_addr.sin_family = AF_INET;
00195     serv_addr.sin_addr.s_addr = INADDR_ANY;
00196     serv_addr.sin_port = htons(portnum);
00197
00198     if (bind(listener_socket, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) < 0)
00199     {
00200         fprintf(stderr, "Error binding socket -- bailing\n");
00201         exit(-1);
00202     }
00203
00204     listen(listener_socket, 1);
00205
00206     int cliilen = sizeof(cli_addr);
00207     uart_socket = accept(listener_socket, (struct sockaddr *)&cli_addr, &cliilen);
00208     printf("[Waiting for incoming conneciton on port %d]\n", portnum);
00209     if (uart_socket < 0)
00210     {
00211         fprintf(stderr, "Error on accept -- bailing\n");
00212         exit(-1);
00213     }
00214
00215     int flags;
00216     flags = fcntl(uart_socket, F_GETFL, 0);
00217     fcntl(uart_socket, F_SETFL, flags | O_NONBLOCK);
00218
00219     printf( "[UART Connected!]" );
00220 }
00221
00222 #endif
00223 //-----
00224 static bool bUDR_Empty = true;
00225 static bool bTSR_Empty = true;
00226
00227 static uint8_t RXB = 0; // receive buffer
00228 static uint8_t TXB = 0; // transmit buffer
00229 static uint8_t TSR = 0; // transmit shift register.
00230 static uint8_t RSR = 0; // receive shift register.
00231
00232 static uint32_t u32BaudTicks = 0;
00233 static uint32_t u32TxTicksRemaining = 0;
00234 static uint32_t u32RxTicksRemaining = 0;
00235
00236 //-----
00237 static void Echo_Tx()
00238 {
00239     if (use_uart_socket) {
00240         if (send(uart_socket, &TSR, 1, 0) <= 0) {
00241             exit(-1);
00242         }
00243     } else {
00244         printf("%c", TSR);
00245     }
00246 }
00247
00248 //-----
00249 static void Echo_Rx()
00250 {
00251     if (use_uart_socket) {
00252         if (send(uart_socket, &RSR, 1, 0) <= 0) {
00253             exit(-1);
00254         }
00255     } else {
00256         printf("%c", RSR);
00257     }
00258 }
00259
00260 //-----
00261 static bool UART_IsRxEnabled( void )
00262 {
00263     //DEBUG_PRINT( "RxEnabled\n");
00264     return (stCPU.pstRAM->stRegisters.UCSR0B.RXEN0 == 1);
00265 }
00266
00267 //-----
00268 static bool UART_IsTxEnabled( void )
00269 {
00270     //DEBUG_PRINT( "TxEnabled\n");
00271     return (stCPU.pstRAM->stRegisters.UCSR0B.TXEN0 == 1);
00272 }
00273
00274 //-----
00275 static bool UART_IsTxIntEnabled( void )
00276 {

```

```

00277     return (stCPU.pstRAM->stRegisters.UCSR0B.TXCIE0 == 1);
00278 }
00279
00280 //-----
00281 static bool UART_IsDREIntEnabled( void )
00282 {
00283     return (stCPU.pstRAM->stRegisters.UCSR0B.UDRIE0 == 1);
00284 }
00285
00286 //-----
00287 static bool UART_IsRxIntEnabled( void )
00288 {
00289     return (stCPU.pstRAM->stRegisters.UCSR0B.RXCIE0 == 1);
00290 }
00291
00292 //-----
00293 static bool UART_IsDoubleSpeed()
00294 {
00295     return (stCPU.pstRAM->stRegisters.UCSR0A.U2X0 == 1);
00296 }
00297
00298 //-----
00299 static void UART_SetDoubleSpeed()
00300 {
00301     stCPU.pstRAM->stRegisters.UCSR0A.U2X0 = 1;
00302 }
00303
00304 //-----
00305 static void UART_SetEmpty( void )
00306 {
00307     stCPU.pstRAM->stRegisters.UCSR0A.UDRE0 = 1;
00308 }
00309
00310 //-----
00311 static void UART_ClearEmpty( void )
00312 {
00313     stCPU.pstRAM->stRegisters.UCSR0A.UDRE0 = 0;
00314 }
00315
00316 //-----
00317 static bool UART_IsEmpty( void )
00318 {
00319     return (stCPU.pstRAM->stRegisters.UCSR0A.UDRE0 == 1);
00320 }
00321
00322 //-----
00323 static bool UART_IsTxComplete( void )
00324 {
00325     return (stCPU.pstRAM->stRegisters.UCSR0A.TXC0 == 1);
00326 }
00327
00328 //-----
00329 static void UART_TxComplete( void )
00330 {
00331     stCPU.pstRAM->stRegisters.UCSR0A.TXC0 = 1;
00332 }
00333
00334 //-----
00335 static bool UART_IsRxComplete( void )
00336 {
00337     return (stCPU.pstRAM->stRegisters.UCSR0A.RXC0 == 1);
00338 }
00339
00340 //-----
00341 static void UART_RxComplete( void )
00342 {
00343     stCPU.pstRAM->stRegisters.UCSR0A.RXC0 = 1;
00344 }
00345
00346 //-----
00347 static void TXC0_Callback( uint8_t ucVector_ )
00348 {
00349     // On TX Complete interrupt, automatically clear the TXC0 flag.
00350     stCPU.pstRAM->stRegisters.UCSR0A.TXC0 = 0;
00351 }
00352
00353 //-----
00354 static void UART_Init(void *context_ )
00355 {
00356     DEBUG_PRINT("UART Init\n");
00357     stCPU.pstRAM->stRegisters.UCSR0A.UDRE0 = 1;
00358
00359     CPU_RegisterInterruptCallback(TXC0_Callback, stCPU.pstVectorMap->USART0_TX
); // TX Complete
00360
00361     if (Options_GetByName("--uart")) {
00362         use_uart_socket = true;

```

```

00363     UART_BeginServer();
00364 }
00365 }
00366
00367 //-----
00368 static void UART_Read(void *context_, uint8_t ucAddr_, uint8_t *pucValue_)
00369 {
00370     DEBUG_PRINT( "UART Read: 0x%02x == 0x%02X\n", ucAddr_, stCPU.pstRAM->au8RAM[ ucAddr_ ] );
00371     DEBUG_PRINT( "ADDR=%08X\n", stCPU.u32PC );
00372     *pucValue_ = stCPU.pstRAM->au8RAM[ ucAddr_ ];
00373     switch (ucAddr_)
00374     {
00375         case 0xC6: // UDR0
00376             stCPU.pstRAM->stRegisters.UCSR0A.RXC0 = 0;
00377             break;
00378         default:
00379             break;
00380     }
00381 }
00382
00383 //-----
00384 static void UART_WriteBaudReg()
00385 {
00386     DEBUG_PRINT( "WriteBaud\n" );
00387     uint16_t u16Baud = (uint16_t)(stCPU.pstRAM->stRegisters.UBRR0L) |
00388                       ((uint16_t)(stCPU.pstRAM->stRegisters.UBRR0H) << 8);
00389
00390     u32BaudTicks = u16Baud;
00391 }
00392
00393 //-----
00394 static void UART_WriteDataReg()
00395 {
00396     DEBUG_PRINT( "UART Write UDR...\n" );
00397     DEBUG_PRINT( "ADDR=%08X\n", stCPU.u32PC );
00398     if (UART_IsTxEnabled())
00399     {
00400         DEBUG_PRINT( "Enabled...\n" );
00401         // Only set the baud timer if the UART is idle
00402         if (!u32TxTicksRemaining)
00403         {
00404             u32TxTicksRemaining = u32BaudTicks;
00405             if (UART_IsDoubleSpeed())
00406             {
00407                 u32TxTicksRemaining >>= 1;
00408             }
00409         }
00410
00411         // If the shift register is empty, load it immediately
00412         if (bTSR_Empty)
00413         {
00414             TSR = stCPU.pstRAM->stRegisters.UDR0;
00415             TXB = 0;
00416             bTSR_Empty = false;
00417             bUDR_Empty = true;
00418             UART_SetEmpty();
00419
00420             if (UART_IsDREIntEnabled())
00421             {
00422                 DEBUG_PRINT( "DRE Interrupt\n" );
00423                 AVR_InterruptCandidate( stCPU.pstVectorMap->USART0_UDRE );
00424             }
00425         }
00426         else
00427         {
00428             TXB = stCPU.pstRAM->stRegisters.UDR0;
00429             bTSR_Empty = false;
00430             bUDR_Empty = false;
00431             UART_ClearEmpty();
00432         }
00433     }
00434     else
00435     {
00436         DEBUG_PRINT( "Disabled...\n" );
00437     }
00438 }
00439
00440 //-----
00441 static void UART_WriteUCSR0A( uint8_t u8Value_)
00442 {
00443     DEBUG_PRINT( "UART Write UCSR0A...\n" );
00444     uint8_t u8Reg = stCPU.pstRAM->stRegisters.UCSR0A.r;
00445     if (u8Value_ & 0x40) // TXC was set explicitly -- clear it in the SR.
00446     {
00447         u8Reg &= ~0x40;
00448     }
00449     u8Reg &= ~(0xBC);

```

```

00450
00451     stCPU.pstRAM->stRegisters.UCSR0A.r |= u8Reg;
00452 }
00453
00454 //-----
00455 static void UART_UpdateInterruptFlags(void)
00456 {
00457     //DEBUG_PRINT("Check UART Interrupts\n");
00458     if (UART_IsTxIntEnabled())
00459     {
00460         if (UART_IsTxComplete())
00461         {
00462             DEBUG_PRINT("Enable TXC Interrupt\n");
00463             AVR_InterruptCandidate( stCPU.pstVectorMap->USART0_TX );
00464         }
00465         else
00466         {
00467             DEBUG_PRINT("Clear TXC Interrupt\n");
00468             AVR_ClearCandidate( stCPU.pstVectorMap->USART0_TX );
00469         }
00470     }
00471     if (UART_IsDREIntEnabled())
00472     {
00473         if ( UART_IsEmpty() )
00474         {
00475             DEBUG_PRINT("Enable DRE Interrupt\n");
00476             AVR_InterruptCandidate( stCPU.pstVectorMap->USART0_UDRE );
00477         }
00478         else
00479         {
00480             DEBUG_PRINT("Clear DRE Interrupt\n");
00481             AVR_ClearCandidate( stCPU.pstVectorMap->USART0_UDRE );
00482         }
00483     }
00484     if (UART_IsRxIntEnabled())
00485     {
00486         if (UART_IsRxComplete())
00487         {
00488             DEBUG_PRINT("Enable RXC Interrupt\n");
00489             AVR_InterruptCandidate( stCPU.pstVectorMap->USART0_RX );
00490         }
00491         else
00492         {
00493             DEBUG_PRINT("Clear RXC Interrupt\n");
00494             AVR_ClearCandidate( stCPU.pstVectorMap->USART0_RX );
00495         }
00496     }
00497 }
00498
00499 //-----
00500 static void UART_WriteUCSR0B( uint8_t u8Value_ )
00501 {
00502     DEBUG_PRINT("Write UCSR0B = %02x\n", u8Value_);
00503     stCPU.pstRAM->stRegisters.UCSR0B.r = u8Value_;
00504     UART_UpdateInterruptFlags();
00505 }
00506
00507 //-----
00508 static void UART_WriteUCSR0C( uint8_t u8Value_ )
00509 {
00510     DEBUG_PRINT("Write UCSR0C\n");
00511     stCPU.pstRAM->stRegisters.UCSR0C.r == u8Value_;
00512 }
00513
00514 //-----
00515 static void UART_Write(void *context_, uint8_t ucAddr_, uint8_t ucValue_ )
00516 {
00517     DEBUG_PRINT("UART Write: %2X=%2X\n", ucAddr_, ucValue_ );
00518     DEBUG_PRINT("ADDR=%08X\n", stCPU.u32PC);
00519     switch (ucAddr_)
00520     {
00521     case 0xC0: //UCSR0A
00522         UART_WriteUCSR0A( ucValue_ );
00523         break;
00524     case 0xC1: //UCSR0B
00525         UART_WriteUCSR0B( ucValue_ );
00526         break;
00527     case 0xC2: //UCSR0C
00528         UART_WriteUCSR0C( ucValue_ );
00529         break;
00530     case 0xC3: // NA.
00531         break;
00532     case 0xC4: //UBRR0L
00533     case 0xC5: //UBRR0H
00534         DEBUG_PRINT("Write UBRR0x\n");
00535         stCPU.pstRAM->au8RAM[ ucAddr_ ] = ucValue_;
00536         UART_WriteBaudReg();

```

```

00537         break;
00538     case 0xC6: //UDR0
00539         DEBUG_PRINT("Write UDR0\n");
00540         stCPU.pstRAM->au8RAM[ ucAddr_ ] = ucValue_;
00541         UART_WriteDataReg();
00542         break;
00543     default:
00544         break;
00545     }
00546 }
00547
00548 //-----
00549 static void UART_TxClock(void *context_ )
00550 {
00551     //DEBUG_PRINT("TX clock...\n");
00552     if (UART_IsTxEnabled() && u32TxTicksRemaining)
00553     {
00554         DEBUG_PRINT("Countdown %d ticks remain\n", u32TxTicksRemaining);
00555         u32TxTicksRemaining--;
00556         if (!u32TxTicksRemaining)
00557         {
00558             // Local echo of the freshly "shifted out" data to the terminal
00559             Echo_Tx();
00560
00561             // If there's something queued in the TXB, reload the TSR
00562             // register, flag the UDR as empty, and TSR as full.
00563             if (!bUDR_Empty)
00564             {
00565                 TSR = TXB;
00566                 TXB = 0;
00567                 bUDR_Empty = true;
00568                 bTSR_Empty = false;
00569
00570                 UART_SetEmpty();
00571
00572                 if (UART_IsDREIntEnabled())
00573                 {
00574                     DEBUG_PRINT("DRE Interrupt\n");
00575                     AVR_InterruptCandidate( stCPU.pstVectorMap->USART0_UDRE );
00576                 }
00577             }
00578             // Nothing pending in the TXB? Flag the TSR as empty, and
00579             // set the "Transmit complete" flag in the register.
00580             else
00581             {
00582                 TXB = 0;
00583                 TSR = 0;
00584                 bTSR_Empty = true;
00585
00586                 UART_TxComplete();
00587                 if (UART_IsTxIntEnabled())
00588                 {
00589                     DEBUG_PRINT("TXC Interrupt\n");
00590                     AVR_InterruptCandidate( stCPU.pstVectorMap->USART0_TX );
00591                 }
00592             }
00593         }
00594     }
00595 }
00596
00597 //-----
00598 static void UART_RxClock(void *context_ )
00599 {
00600     if (UART_IsRxEnabled())
00601     {
00602         if (u32RxTicksRemaining) {
00603             u32RxTicksRemaining--;
00604             if (!u32RxTicksRemaining)
00605             {
00606                 // Move data from receive shift register into the receive buffer
00607                 RXB = RSR;
00608                 RSR = 0;
00609
00610                 stCPU.pstRAM->stRegisters.UDR0 = RXB;
00611
00612                 // Set the RX Complete flag
00613                 UART_RxComplete();
00614                 if (UART_IsRxIntEnabled())
00615                 {
00616                     DEBUG_PRINT("RXC Interrupt\n");
00617                     AVR_InterruptCandidate( stCPU.pstVectorMap->USART0_RX );
00618                 }
00619             }
00620         } else {
00621             if (use_uart_socket) {
00622                 uint8_t rx_byte;
00623                 int bytes_read = recv(uart_socket, &rx_byte, 1, 0);

```

```
00624         if (bytes_read == 1) {
00625             RSR = rx_byte;
00626             u32RxTicksRemaining = u32BaudTicks;
00627         }
00628     }
00629 }
00630 }
00631 }
00632 //-----
00633 static void UART_Clock(void *context_ )
00634 {
00635     // Handle Rx and TX clocks.
00636     UART_TxClock(context_);
00637     UART_RxClock(context_);
00638 }
00639
00640 //-----
00641 AVRPeripheral stUART =
00642 {
00643     UART_Init,
00644     UART_Read,
00645     UART_Write,
00646     UART_Clock,
00647     0,
00648     0xC0,
00649     0xC6
00650 };
```

#### 4.153 src/peripheral/mega\_uart.h File Reference

ATMega UART implementation.

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

## Variables

- AVRPeripheral **stUART**

#### 4.153.1 Detailed Description

## ATMega UART implementation.

Definition in file [mega\\_uart.h](#).

#### 4.154 mega\_uart.h

```

00001 /*****
00002 *      (      (      (      |
00003 *      )\ ) )\ )      )\ ) |
00004 *      ()/( ()/(      )\      ( ( ()/(      | -- [ Funkenstein ] -----
00005 *      /( ) )/( ) ((( ( )\ )\ /( ) ) | -- [ Little ] -----
00006 *      ( )_ ( ) )\ _ )\ ( ( ( ) ( ) | -- [ AVR ] -----
00007 *      | _ | | ( )_ ( ) \ \ / / | _ \ | -- [ Virtual ] -----
00008 *      | _ | | _ / _ \ \ \ / / | _ / | -- [ Runtime ] -----
00009 *      |_ | | _ | / _ \ \ \ / / |_ |_ |
00010 *
00011 * -----|-----
00012 * (c) Copyright 2014-17, Funkenstein Software Consulting, All rights reserved
00013 * See license.txt for details
00014 *****/
00021 #ifndef __MEGA_UART_H__
00022 #define __MEGA_UART_H__
00023
00024 #include "avr_peripheral.h"
00025
00026 extern AVRPeripheral stUART;
00027
00028 #endif //__MEGA_UART_H__

```

