Instruction set

Mnemonics	# Clocks	Description	Constraints/Remarks/Aliases	Flags written							Encoding (little endian)																	
				I	T	H	S	V	N	Z	C	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
												31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ALU (reg,reg)																												
MOV Rd,Rr		1	Move between two registers									0	0	0	0	1	1	r4	d4	d3	d2	d1	d0	r3	r2	r1	r0
MOVW Rd,Rr		1	Move word between two pairs of registers									0	0	0	0	0	0	0	1	d4	d3	d2	d1	r4	r3	r2	r1
AND Rd,Rr		1	Bitwise AND	**TST:= AND Rd,Rd**				S	V	N	Z	0	0	0	0	0	0	r4	d4	d3	d2	d1	d0	r3	r2	r1	r0
OR Rd,Rr		1	Bitwise OR					S	V	N	Z	0	0	0	0	1	0	r4	d4	d3	d2	d1	d0	r3	r2	r1	r0
EOR Rd,Rr		1	Bitwise XOR	**CLR:= EOR Rd,Rd**				S	V	N	Z	0	0	0	0	0	1	r4	d4	d3	d2	d1	d0	r3	r2	r1	r0
CPSE Rd,Rr		1/2/3	Skip next instruction if equal									0	0	0	1	0	0	r4	d4	d3	d2	d1	d0	r3	r2	r1	r0
CP Rd,Rr	CPC Rd,Rr	1	Compare (with carry)				H	S	V	N	Z	C	0	0	0	0/1	0	1	r4	d4	d3	d2	d1	d0	r3	r2	r1	r0
SUB Rd,Rr	SBC Rd,Rr	1	Subtract (with carry)				H	S	V	N	Z	C	0	0	0	0/1	1	0	r4	d4	d3	d2	d1	d0	r3	r2	r1	r0
ADD Rd,Rr	ADC Rd,Rr	1	Add (with carry)	**LSL = ADD Rd,Rd ROL:=ADC Rd,Rd**			H	S	V	N	Z	C	0	0	0	0/1	1	1	r4	d4	d3	d2	d1	d0	r3	r2	r1	r0
MUL Rd,Rs		2	Unsigned multiply	Result stored into R1:R0							Z	C	1	0	0	1	1	1	r4	d4	d3	d2	d1	d0	r3	r2	r1	r0
MULS Rd,Rr		2	Signed multiply	Works on R16 and higher (r4=d4=1) Result stored into R1:R0							Z	C	0	0	0	0	0	0	1	0	d3	d2	d1	d0	r3	r2	r1	r0
MULSU Rd,Rr		2	Signed by Unsigned multiply	Works on R24 and higher (r4=d4=1, r3=d3=1) Result stored into R1:R0 'Fractional' ops shift the result to left by 1 to support fixed point 1.7 format							Z	C	0	0	0	0	0	0	1	1	0	d2	d1	d0	0	r2	r1	r0
FMUL Rd,Rr		2	Fractional Unsigned multiply								Z	C	0	0	0	0	0	0	1	1	0	d2	d1	d0	1	r2	r1	r0
FMULS Rd,Rr		2	Fractional Signed multiply								Z	C	0	0	0	0	0	0	1	1	1	d2	d1	d0	0	r2	r1	r0
FMULSU Rd,Rr		2	Fractional Signed by Unsigned multiply								Z	C	0	0	0	0	0	0	1	1	1	d2	d1	d0	1	r2	r1	r0
ALU (reg, imm)																												
CPI Rd,i		1	Compare register with imm	Works on R16 and higher (d4=1) **SER:= LDI Rd, 0xFF**			H	S	V	N	Z	C	0	0	1	1	i7	i6	i5	i4	d3	d2	d1	d0	i3	i2	i1	i0
SUBI Rd,i	SBCI Rd,i	1	Subtract imm from register (with carry)				H	S	V	N	Z	C	0	1	0	0/1	i7	i6	i5	i4	d3	d2	d1	d0	i3	i2	i1	i0
ORI Rd,i		1	Bitwise OR register with imm					S	V	N	Z	C	0	1	1	0	i7	i6	i5	i4	d3	d2	d1	d0	i3	i2	i1	i0
ANDI Rd,i		1	Bitwise AND register with imm					S	V	N	Z	C	0	1	1	1	i7	i6	i5	i4	d3	d2	d1	d0	i3	i2	i1	i0
LDI Rd,i		1	Load imm to register										1	1	1	0	i7	i6	i5	i4	d3	d2	d1	d0	i3	i2	i1	i0
ADIW Rd,i		2	Add imm to a pair of registers	Works on W,X,Y,Z only (d3=d4=1,d0=0)				S	V	N	Z	C	1	0	0	1	0	1	1	0	i5	i4	d2	d1	i3	i2	i1	i0
SBIW Rd,i		2	Subtract imm from a pair of registers					S	V	N	Z	C	1	0	0	1	0	1	1	1	i5	i4	d2	d1	i3	i2	i1	i0
ALU (reg)																												
SWAP Rd		1	Swap nibbles										1	0	0	1	0	1	0	d4	d3	d2	d1	d0	0	0	1	0
NEG Rd		1	Two complement				H	S	V	N	Z	C	1	0	0	1	0	1	0	d4	d3	d2	d1	d0	0	0	0	1
COM Rd		1	1nd compliment					S	V	N	Z	C	1	0	0	1	0	1	0	d4	d3	d2	d1	d0	0	0	0	0
ASR Rd		1	Arithmetic shift right	Left shifts are done using ADD/ADC				S	V	N	Z	C	1	0	0	1	0	1	0	d4	d3	d2	d1	d0	0	1	0	1
LSR Rd		1	Logical shift right					S	V	N	Z	C	1	0	0	1	0	1	0	d4	d3	d2	d1	d0	0	1	1	0
ROR Rd		1	Logical shift right through carry					S	V	N	Z	C	1	0	0	1	0	1	0	d4	d3	d2	d1	d0	0	1	1	1
DEC Rd		1	Decrease by 1	C flag not affected unlike ADD				S	V	N	Z		1	0	0	1	0	1	0	d4	d3	d2	d1	d0	1	0	1	0
INC Rd		1	Increase by 1					S	V	N	Z		1	0	0	1	0	1	0	d4	d3	d2	d1	d0	0	0	1	1
Bit operations																												
BSET b	BCLR b	1	Set/Clear bit in SFLAGS	Aliases: (SE|CL)[ITHSVNZC]		Bit b							1	0	0	1	0	1	0	0	0/1	b2	b1	b0	1	0	0	0
BLD Rd, b	BST Rd, b	1	Read/Write a bit from register to/ftom T		T								1	1	1	1	1	0	0/1	d4	d3	d2	d1	d0	0	b2	b1	b0
SBRC Rd, b		1/2/3	Skip instruction if bit in register is clear										1	1	1	1	1	1	0	d4	d3	d2	d1	d0	0	b2	b1	b0
SBRS Rd, b		1/2/3	Skip instruction if bit in register is set										1	1	1	1	1	1	1	d4	d3	d2	d1	d0	0	b2	b1	b0
Memory load/stores																												
LDS Rd, a	STS a, Rr	2	Access *(a)										a15	a14	a13	a12	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2	a1	a0
													1	0	0	1	0	0	0/1	r/d4	r/d3	r/d2	r/d1	r/d0	0	0	0	0
LD Rd, X	ST X,Rr	2	Access *X										1	0	0	1	0	0	0/1	r/d4	r/d3	r/d2	r/d1	r/d0	1	1	0	0
LDD Rd, Y+o	STD Y+o, Rr	2	Access *(Y+o)	Offset is unsigned 6 bit value. Can be zero to access *Y, *Z									1	0	o5	0	o4	o3	0/1	r/d4	r/d3	r/d2	r/d1	r/d0	1	o2	o1	o0
LDD Rd, Z+o	STD Z+o, Rr	2	Access *(Z+o)										1	0	o5	0	o4	o3	0/1	r/d4	r/d3	r/d2	r/d1	r/d0	0	o2	o1	o0
LD Rd,X+	ST X+, Rr	2	Access *X++	Rd != 26,27									1	0	0	1	0	0	0/1	r/d4	r/d3	r/d2	r/d1	r/d0	1	1	0	1
LD Rd,Y+	ST Y+, Rr	2	Access *Y++	Rd != 28,29									1	0	0	1	0	0	0/1	r/d4	r/d3	r/d2	r/d1	r/d0	1	0	0	1
LD Rd,Z+	ST Z+, Rr	2	Access *Z++	Rd != 30,31									1	0	0	1	0	0	0/1	r/d4	r/d3	r/d2	r/d1	r/d0	0	0	0	1
LD Rd,-X	ST -X, Rr	2	Access --*X	Rd != 26,27									1	0	0	1	0	0	0/1	r/d4	r/d3	r/d2	r/d1	r/d0	1	1	1	0
LD Rd,-Y	ST -Y, Rr	2	Access --*Y	Rd != 28,29									1	0	0	1	0	0	0/1	r/d4	r/d3	r/d2	r/d1	r/d0	1	0	1	0
LD Rd,-Z	ST -Z, Rr	2	Access --*Z	Rd != 30,31									1	0	0	1	0	0	0/1	r/d4	r/d3	r/d2	r/d1	r/d0	0	0	1	0
POP Rd	PUSH Rr	2	Pop/push from/to the stack										1	0	0	1	0	0	0/1	r/d4	r/d3	r/d2	r/d1	r/d0	1	1	1	1
SBIC p,b	SBIS p,b	1/2/3	Skip instruction if bit in IO port is clear/set	Only 32 low IO ports supported (memory 0x20..0x3F)									1	0	0	1	1	0	0/1	1	p4	p3	p2	p1	p0	b2	b1	b0
CBI p,b	SBI p,b	2	Clear bit in IO port										1	0	0	1	1	0	0/1	0	p4	p3	p2	p1	p0	b2	b1	b0
IN Rd,p	OUT p, Rr	1	Read/write IO port	IO ports are memory 0x20..0x5F									1	0	1	1	0/1	p5	p4	r/d4	r/d3	r/d2	r/d1	r/d0	p3	p2	p1	p0
Code Memory load/flash																												
LPM Rd,Z		3	Read PROGMEM(Z)	Rd != 30,31									1	0	0	1	0	0	0	d4	d3	d2	d1	d0	0	1	0	0
LPM Rd, Z+		3	Read PROGMEM(Z++)										1	0	0	1	0	0	0	d4	d3	d2	d1	d0	0	1	0	1
LPM Z		3	Read PROGMEM(Z) to R0	Redundant encoding									1	0	0	1	0	1	0	1	1	1	0	0	1	0	0	0
SPM Z		LOT	Store to program memory using Z	Need usage of SPMCSR IO register first. Z must be page or word aligned									1	0	0	1	0	1	0	1	1	1	1	0	1	0	0	0
SPM Z+		LOT	Store to program memory using Z++										1	0	0	1	0	1	0	1	1	1	1	1	1	0	0	0
Branches																												
BRBC b,o		1/2	Relative conditional branch if flag bit clear	Offset is signed, from PC+2, even (o0 = 0) Aliases for each flag (BREQ/...)									1	1	1	1	0	0	o7	o6	o5	o4	o3	o2	o1	b2	b1	b0
BRBS b,o		1/2	Relative conditional branch if flag bit set										1	1	1	1	0	1	o7	o6	o5	o4	o3	o2	o1	b2	b1	b0
RJMP o	RCALL o	2/4	Relative branch	Offset is signed, from PC+2, even (o0 = 0)									1	1	0	0/1	o12	o11	o10	o9	o8	o7	o6	o5	o4	o3	o2	o1
JMP a	CALL a	3/5	Absolute branch	Address is even (a0 = 0), signed, 22 bit									a16	a15	a14	a13	a12	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2	a1
													1	0	0	1	0	1	0	a22	a21	a20	a19	a18	1	1	0/1	a17
IJMP Z	ICALL Z	2/4	Absolute indirect branch via Z	Address is even (bit 0 of Z is ignored)									1	0	0	1	0	1	0	0/1	0	0	0	0	1	1	0	1
RET		5	Return										1	0	0	1	0	1	0	1	0	0	0	0	1	0	0	0
RETI		5	Return from interrupt	I flag is set by this instruction	I								1	0	0	1	0	1	0	1	0	0	0	1	1	0	0	0
System Instructions																												
NOP		1	NO operation	Or is it?									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SLEEP		1	Enter low power sleep state	Only works when SE is enabled in SECR									1	0	0	1	0	1	0	1	1	0	0	0	1	0	0	0
BREAK		ANY	Break to the OCD debugger	NOP when OCD not enabled									1	0	0	1	0	1	0	1	1	0	0	1	1	0	0	0
WDR		1	Feed the watchdog (to avoid bites)	It bites, trust me....									1	0	0	1	0	1	0	1	1	0	0	1	0	0	0	0

Arch registers

	SREG	Flags register							
0x5F		I	T	H	S	V	N	Z	C
		Interrupt flag When clear, all interupts are disabled	**Bit read/write flag** Set/Clear only by BLD/BST	**Half carry flag** Set when bit 3 overflows to bit 4	**Signed Negative** S:=(N xor V)	**Signed overflow flag** Set when bit 6 overflows to bit 7	**Unsigned negative** Set to the bit 7 of the result	**Zero** Set iff the result is zero	**Carry** Unsigned carry flag. Set when bit 7 overflows
0x5D 0x5E	**SPL/H**	**Stack pointer** (must be initialized prior to using stack, should point to end of the RAM initially)							

Interrupt table

0x02	External Interrupt Request 0	**0x14**	USB General Interrupt request	**0x2A**	Timer0 Match A	**0x40**	Timer3 Compare A
0x04	External Interrupt Request 1	**0x16**	USB Endpoint Interrupt request	**0x2C**	Timer0 Match B	**0x42**	Timer3 Compare B
0x06	External Interrupt Request 2	**0x30**	SPI Serial Transfer Complete	**0x2E**	Timer0 Overflow	**0x44**	Timer3 Compare C
0x08	External Interrupt Request 3	**0x32**	USART1 Rx Complete	**0x20**	Timer1 Capture Event	**0x46**	Timer3 Overflow
0x0E	External Interrupt Request 6	**0x34**	USART1 Tx FIFO has space	**0x22**	Timer1 Compare A	**0x4C**	Timer4 Compare A
0x12	Pin Change Interrupt Request 0	**0x36**	USART1 Tx Complete	**0x24**	Timer1 Compare B	**0x4E**	Timer4 Compare B
0x18	Watchdog Timeout Interrupt	**0x48**	2-wire Serial Interface	**0x26**	Timer1Compare C	**0x50**	Timer4 Compare D
0x3C	EEPROM Programming Done	**0x38**	Analog Comparator	**0x28**	Timer1 Overflow	**0x52**	Timer4 Overflow
0x4A	Flash Programming Done	**0x3A**	ADC Conversion Complete	**0x3E**	Timer3 Capture Event	**0x54**	Timer4 Fault Protection

Timer0 (8 bit)

0x48	OCR0B	**Timer comparator B**							
0x47	OCR0A	**Timer comparator A** Can also be used as TOP							
0x46	TCNT0	**Timer counter value**							
0x45	TCCR0B	**FOC0A** **Force compare A** Toggle OCR0A pin as if compare happened. for non PWM mode	**FOC0B** **Force compare B** Toggle OCR0B pin as if compare happened. for non PWM mode			**WGM02** **Use OCRA instead of fixed 0xFF TOP** When on, the OCR0A can't be used for PWM	**CS (Timer0 clock source - from prescaler)** 0 - stopped || 1 - CLK || 2 - CLK >> 3 || 3 - CLK >> 6 || 4 - CLK >> 8 || 5 - CLK >> 10 6 - T0 (failing edge) || 7 - T0 (rising edge)	
0x44	TCCR0A	**COM0A**		**COM0B**				**WGM01**	**WGM00**
		Non PWM: **0** - **disconnected** **1** - **toggle** on match **2** - **clear** on match **3** - **set** on match	**PWM modes:** 0 - **disconnected** 1 - **disconnected** toggle on match if WGM02=1 2 - **normal** PWM not useful when WGM02=1 3 - **inverse** PWM not useful when WGM02=1	**Non PWM:** **0** - **disconnected** **1** - **toggle** on match **2** - **clear** on match **3** - **set** on match	**PWM modes:** **0**-**disconnected** **1**-**disconnected** **2**-**normal** PWM **3**-**inverse** PWM			**Non PWM modes :** * OCR0X update immediate * OCR0X pins update at overflow: **0** - **Normal** (only when WGM02 == 0) **2** - **CTC** (only when WGM02 == 1)	**1** - **Fast PWM mode** * Clear pin on compare * Set pin on TOP * OCR0X regs update/TOV at TOP **3** - **PWM, Phase correct** * Up/Down counting * OCR0X regs update/TOV at TOP * Pin clear at OCR0X when upcounting * Pin set at OCR0X when down counting
0x6E	TIMSK0						**OCIEnB** Interrupt when OCF0B == 1	**OCIEnA** Interrupt when OCF0A == 1	**TOIEn** Interrupt when TOV0 == 1
0x35	TIFR0						**OCF0B** **Comparator B match interrupt flag** Auto cleared by ISR	**OCF0A** **Comparator A match interrupt flag** Auto cleared by ISR Can trigger ADC	**TOV0** **Overflow interrupt flag** Auto cleared by ISR Can trigger ADC

Timer1/3 (16 bit)

0x(8,9)(C,D)	OCRnCL/H	**Timer 1/3 comparator C (16 bit)**							
0x(8,9)(A,B)	OCRnBL/H	**Timer 1/3 comparator B (16 bit)**							
0x(8,9)(8,9)	OCRnAL/H	**Timer 1/3 comparator A (16 bit)** Can be used as TOP value in some modes							
0x(8,9)(6,7)	ICRnL/H	**Input capture value 1/3 (16 bit)** Can be also used as TOP in some modes Will store here the timer counter value at the moment of the input capture event							
0x(8,9)(4,5)	TCNT1L/H	**Timer counter value (16 bit)**							
0x82/0x92	TCCR1C	**FOCnA** **Force compare A** Toggle OCRnA pin as if compare happened. for non PWM mode	**FOC1B (timer1)** **Force compare B** Toggle OCR1B pin as if compare happened. for non PWM mode	**FOC1C (timer1)** **Force compare C** Toggle OCR1C pin as if compare happened. for non PWM mode					
0x81/0x91	TCCR1B	**ICNCn** **Enable input Capture Noise Canceler**	**ICESn** **Input capture edge select:** **0** - failing edge || **1** - rising edge		**WGMn3** Upper bits of WGMn	**WGMn2**	**CSn (Timer1/3 clock source selector - taken from prescaler)** 0 - stopped || 1 - CLK || 2 - CLK >> 3 || 3 - CLK >> 6 || 4 - CLK >> 8 || 5 - CLK >> 10 6 - T1 (failing edge, timer1 only) || 7 - T1 (rising edge, timer1 only)		
0x80/0x90	TCCR1A	**COMnA** **Non PWM:** 0 - **disconnected** 1 - **toggle** on match 2 - **clear** on match 3 - **set** on match	**PWM modes:** 0 - **disconnected** 1 - **disconnected** (or toggle on match for PWM modes that use ICRn or OCRnA) 2 - **normal** PWM (not useful for modes that use OCRnA as TOP) 3 - **inverse** PWM (not useful for modes that use OCRnA as TOP)	**COMnB (timer 1 only)** **Non PWM:** 0 - **disconnected** 1 - **toggle** on match 2 - **clear** on match 3 - **set** on match **PWM modes:** 0 - disconnected 1 - disconnected 2 - **normal** PWM 3 - **inverse** PWM		**COMnC (timer 1 only)** Same as COMnB		**WGMn** **Non PWM:** * OCRnX reg update immediate * Pin toggle/clear/set on compare **0** - timer overflows at 0xFFFF **12/4** -timer overflows at ICRn/OCRnA **Fast PWM modes:** * Clear pin on compare * Set pin on TOP * OCRnX regs update/TOV at TOP **5/6/7** - TOP at 0xFF/ 0x1FF/ 0x3FF **14/15** - TOP at ICRn / OCRnA	**Phase correct PWM** * Up/Down counting * OCRnX regs update/TOV at TOP * Pin clear at OCRnX when upcounting * Pin set at OCRnX when down counting **1/2/3** - TOP at 0xFF/0x1FF/ 0x3FF **10/11** - TOP at ICRn / OCRnA **Phase and frequency correct PWM** * Up/Down counting * OCRnX regs update/TOV at 0, * Pin clear at OCRnX when upcounting, * Pin set at OCRnX when downcounting **8/9** - TOP at ICRn / OCRnA
0x6F/0x71	TIMSK1/3			**ICIEn** Interrupt when ICFn== 1		**OCIEnC** Interrupt when OCFnC == 1	**OCIEnB** Interrupt when OCFnB == 1	**OCIEnA** Interrupt when OCFnA== 1	**TOIEn** Interrupt when TOVn == 1
0x36/0x38	TIFR1/3			**ICFn** **Timer 1/3 Input capture interrupt flag** Auto cleared by ISR can trigger ADC		**OCFnC** **Timer 1/3 C match interrupt flag** Auto cleared by ISR	**OCFnB** **Timer 1/3 B match interrupt flag** Auto cleared by ISR Can trigger ADC	**OCFnA** **Timer 1/3 A match interrupt flag** Auto cleared by ISR	**TOVn** **Timer 1/3 overflow interrupt flag** Auto cleared by ISR Can trigger ADC

Timer4 (10 bit, can be feed from up to 96Mhz internal PLL)

0xD4	DT4	**DT4H** Delay rising edge of OCnX by (DT4H << DTPS) timer clock cycles				**DT4L** Delay rising edge of ~OCnX by (DT4H << DTPS) timer clock cycles			
0xD2	OCR4D	**Timer comparator D** (2/3 high bits read/write using TC4H)							
0xD1	OCR4C	**Timer TOP value** (2 high bits read/write using TC4H)							
0xD0	OCR4B	**Timer comparator B** (2/3 high bits read/write using TC4H)							
0xCF	OCR4A	**Timer comparator A** (2/3 high bits read/write using TC4H)							
0xC4	TCCR4E	**TLOCK4** **Lock timer registers** Delay update of al comparator internal registers till this bit is clear	**ENHC4** **11 Bit comparator mode.** Bit 0 in the comparator registers selects clock phase of the timer clock (0/1 - event on rising/falling edge of timer clock)	**OC4OE** **Enable bits for each PWM output in PWM6 mode to allow sofware to switch them on the fly** Enable PWM6 output on OCR4D	Enable PWM6 output on ~OCR4D	Enable PWM6 output on OCR4B	Enable PWM6 output on ~OCR4B	Enable PWM6 output on OCR4A	Enable PWM6 output on ~OCR4A
0xC3	TCCR4D	**FPIE4** Interrupt when FPF4 == 1	**FPEN4** **Fault protection mode enable** When fault protection source is triggered will stop the timer, and disconnect all its PWM outputs	**FPNC4** **Enable Fault protection noise canceler** (debouncer)	**FPES4** **Interrupt polarity of fault protection interrupt:** **0** - failing edge **1** - rising edge	**FPAC4** **Fault protection trigger source:** **0** - INT0 **1** - Analog comparator	**FPF4** **Fault protection interrupt flag** Auto cleared by ISR	**WGM4** **0 - Fast PWM** * Clear pin on compare * Set pin on TOP (OCR4C) * OCR4X load/TOV on TOP **1 - Phase and frequency correct PWM** * Up/Down counting * OCRnX regs update/TOV at 0, * Pin clear OCRnX when upcounting, * Pin set at OCRnX when downcounting	**2 - PWM6 / Single Slope** * same as Fast PWM, but all 6 outputs synced to comparator A (which is controlled by COM4A) every output can be disabled using OC4OE **3 - PWM6 / Dual slope** * same as Phase and freq correct PWM, but all 6 outputs synced to comparator A every output can be disabled using OC4OE
0xC2	TCCR4C	**COM4AS** COM4A shadow		**COM4BS** COM4B shadow		**COM4D** **Comparator mode for D comparator** Same as COM4A but without PWM6 modes and for OC4D pin	**FOC4D** **Force compare D** Toggle OCR4D pin as if compare happened. for non PWM mode	**PWM4D** **PWM mode for D comparator** Same as PWM4A but for D comparator See PWM4A for description	
0xC1	TCCR4B	**PWM4X** **Invert PWM outputs** When set , ~OC4x and ~OC4X are inverted (after dead time generator)	**PSR4** Reset timer4 prescaler	**DTPS4** **Dead time prescaler value** See DT4H/DT4L for description		**CS4** **Timer4 clock prescaler:** **0** - Timer stopped **otherwise** - (T4CLK >> (CS4-1)) T4CLK is either system clock or output of the PLL			
0xC0	TCCR4A	**COM4A** **PWM modes (0,1):** **0** - all outputs **disconnected** **1** - **normal PWM** on OC4A pin, **inverted PWM** on ~OC4A pin 2 - **normal** PWM on OC4A 3 - **inverted** PWM on OC4A	**PWM6 modes:** 0 - all outputs **disconnected** 1 - **normal** PWM on all 3 OC4X pins and **inverted** PWM on all 3 ~OC4X pins 2 - **normal** PWM on all 6 pins 3 - **inverted** PWM on all 6 pins)	**COM4B** **Comparator mode for B comparator** Same as COM4A but without PWM6 modes and for OC4B pin	**FOC4A** **Force compare A** Toggle OCR4A pin as if compare happened. for non PWM mode	**FOC4B** **Force compare B** Toggle OCR4B pin as if compare happened. for non PWM mode	**PWM4A** **Enable PWM mode for comparator A** When PWM mode is disabled for comp A, using this bit, the COM4A bits behave this way: 0 - all outputs **disconnected** 1 - **toggle** OC4A pin on compare 2 - **clear** OC4A pin on compare 3 - **set** OCR4A pin on compare corresponding OCRnA register is not buffered. ~OCR4A pin can't be used	**PWM4B** **PWM mode for comparator B** Same as PWM4A but for B comparator See PWM4A for description
0xBF	TC4H						**High part of all 10/11 bit registers**		
0xBE	TCNT4	**Timer counter value (low 8 bit)** (2 high bits read/write using TC4H)							
0x72	TIMSK4	**OCIE4D** Interrupt when OCF4D == 1	**OCIE4A** Interrupt when OCF4A == 1	**OCIE4B** Interrupt when OCF4B== 1			**TOIE4** Interrupt when TOV4== 1		
0x39	TIFR4	**OCF4D** **Comparator D interrupt flag** Auto cleared by ISR Can trigger ADC	**OCF4A** **Comparator A interrupt flag** Auto cleared by ISR Can trigger ADC	**OCF4B** **Comparator B interrupt flag** Auto cleared by ISR Can trigger ADC			**TOV4** **Overflow interrupt flag** Auto cleared by ISR Can trigger ADC		

GPIO port D (most interrupts, I2C, Serial, timer 1 input capture, misc ADC inputs, timer clock inputs,, timer 4 pwm D, timer 0 pwm B, control bus for HVPP)

0x2B	PORTD	**PORTD** (For output: level to set the port at || For input: 0 - hiZ, 1 - pullup enabled)						
0x2A	DDRD	**DDD** (Each pin data direction (input/output))						
0x29	PIND	**PIND** (Read: pin value for each pin || Write 1 to toggle the output value of the pin)						
IO functions		**T0**	**T1**	**XCK1**	**ICP1**	**INT3**	**INT2**	**INT1**	**INT0**
		TImer 0 clock input The clock should be less that CPU clock / 2	**Timer 1 clock input** The clock should be less that CPU clock / 2	Clock input/output for USART sync/SPI mode.	**Timer 1 Input capture pin** (ICP3 is on PORTC)	**Interrupt pins. 0-3** INT6 is on PORTE			
				~CTS		**TXD1** (output)	**RXD1**	**SDA** (output/low)	**SCL** (output/low)
				Serial clear to send input (async) When low, will pause transmission		USART transmitter output	USART receiver input	**I2C data line.** MCU will pull it low to transmit 0, and tri-state to transmit 1	**I2C clock line** MCU will pull it low to transmit 0, and tri-state to transmit 1
		ADC10	**ADC9**		**ADC8**				
		ADC inputs (can't be used for diff measurement)			**ADC inputs** (can't be used for diff measurement)				
		OC4D (output)	**~OC4D** (output)						**OC0B**
		Timer4 PWM regular output D	**Timer4 PWM inverted output D** Can only be enabled with OC4D						Timer 0 PWM output B

GPIO port B (pin change interrupts, SPI, misc ADC inputs, timer 4 pwm B, timer 1 pwms, timer 0 pwm A, data bus for HVPP)

0x25	PORTB	**PORTB** (For output: level to set the port at || For input: 0 - hiZ, 1 - pullup enabled)						
0x24	DDRB	**DDB** (Each pin data direction (input/output))						
0x23	PINB	**PINB** (Read: pin value for each pin || Write 1 to toggle the output value of the pin)						
IO functions		**PCINT7**	**PCINT6**	**PCINT5**	**PCINT4**	**PCINT3**	**PCINT2**	**PCINT1**	**PCINT0**
		Pin change interrupt pins. Subset of these pins can generate an interrupt, and the interrupt handler has to somehow check which pin caused the interrupt							
		RTS (output)				**MISO** (input/output)	**MOSI** (output/input)	**SCK** (output/input)	**~SS**
		USART ready to send output (async) Signal the other side if we are ready to receive another byte of data				**SPI data line** Master in / Slave out SPI serial data line	**SPI data line** Master out / Slave in SPI serial data line	**SPI clock line** output for master mode, input for slave mode	**SPI slave select line** can be input or output. If input, will detect for collisions by sampling this line. In slave mode will wake the SPI module when pulled low
			ADC13	**ADC12**	**ADC11**				
			ADC inputs (can't be used for diff measurement)						
			OC4B (output)	**~OC4B** (output)					
			Timer4 PWM regular output B	**Timer4 PWM inverted output D.** Can only be enabled with OC4B					
		OC1C (output)	**OC1B** (output)	**OC1A** (output)					
		Timer 1 PWM C	Timer 1 PWM B	Timer 1 PWM A					
		OC0A							
		Timer 0 PWM A When both OC1C and OC0A are enabled, they form output compare modulator. (PORTB7 value determines the function (0 => AND, PORTB7 =1 => OR)							

GPIO port F (main ADC inputs, and JTAG)

0x31	PORTF	**PORTF** (For output: level to set the port at || For input: 0 - hiZ, 1 - pullup enabled)							
0x30	DDRF	**DDRF** (Each pin data direction (input/output))							
0x2F	PINF	**PINF** (Read: pin value for each pin || Write 1 to toggle the output value of the pin)							
IO functions		**TDI**	**TDO** (output)	**TMS**	**TCK**				
		JTAG input/outputs							
		ADC7	**ADC6**	**ADC5**	**ADC4**			**ADC1**	**ADC0**
		ADC inputs Right side of the diff input						**ADC inputs** Left side of diff input + can make diff between these	

GPIO port E (interrupt 6, analog comparator input, bootloader override)

0x2E	PORTE	**PORTE** (For output: level to set the port at || For input: 0 - hiZ, 1 - pullup enabled)						
0x2D	DDRE	**DDE** (Each pin data direction (input/output))						
0x2C	PINE	**PINE** (Read: pin value for each pin || Write 1 to toggle the output value of the pin)						
IO functions			**INT6**				**~HWB**	
			Interrupt 6 input				**Force bootloader entry**	
			AIN0+				Also BS2 bit for HVPP	
			Analog comparator + input					

GPIO port C (timer 3 pwm/input capture, timer 4 pwm A, clock output)

0x28	PORTC	**PORTC** (For output: level to set the port at || For input: 0 - hiZ, 1 - pullup enabled)						
0x27	DDRC	**DDC** (Each pin data direction (input/output))						
0x26	PINC	**PINC** (Read: pin value for each pin || Write 1 to toggle the output value of the pin)						
IO functions		**ICP3**	**OC3A** (output)					
		Timer 3 input capture pin	**Timer 3 PWM A output** only counter A is exposed					
		OC4A (output)	**~OC4A** (output)					
		Timer4 PWM regular output A	**Timer4 PWM inverted output A.** Can only be enabled with OC4A					
		CLKO (output)						
		Clock output						

ADC

0x7C	ADMUX	**REFS**		**ADLAR**	**MUX4**	**MUX3**	**MUX2**	**MUX1**	**MUX0**
		Analog reference voltage (Vref)			**ADC input mux**				
		0 - AREF pin only (external voltage reference) **Keep it in the limits stated below** 1 - AVcc (single ended conversions) 3 - internal 2.56V voltage reference (amplified from bandgap) * **In modes 1,3, AREF pin must not be connected to a voltage source since it will cause short.** You should though connect a decoupling cap to it * First conversion after a change of analog reference might not be accurate		**Left align the ADC result** This allows to read only high part of the conversion result when needing only 8 bit precision	**Single ended inputs:** * 10 bit resolution * 38.5 Khz bandwidth **0x00..0x01** - ADC0,1 **0x04-0x07** - ADC4-7 **0x20-0x25** - ADC8-ADC13 **Special inputs:** **0x1E** - Band gap (1.1V) **0x1F** - Ground **0x27** - Temperature sensor * after selecting, need to wait 2usec	**Diff inputs (no gain):** * 8 bit resolution * 4 khz bandwidth **0x10** - ADC0-ADC1 **0x14-0x17** - ADC4-7 - ADC1 * MUX is buffered, and its real value stops updating after conversion start and till its end	**Diff inputs with 10x gain:** * 8 bit resolution * 4 khz bandwidth **0x09** - ADC1-ADC0 **0x28-0x2B** - ADC4-7 - ADC0 **0x2C-0x2F** - ADC4-7 - ADC1 * After selecting diff input, wait at least 125 usec prior to sampling.	**Diff inputs with 40x gain:** * 8 bit resolution * 4 khz bandwidth **0x26**-ADC1-ADC0 **0x30-0x33** - ADC4-7 - ADC0 **0x34-0x37** - ADC4-7 - ADC1 * AVcc Range: Vcc - 0.3V <-> Vcc + 0.3V (close to Vcc)	**Diff inputs with 200x gain:** * 7 bit resolution * 4 khz bandwidth **0x0B** - ADC1-ADC0 **0x38-0x3B** - ADC4-7 - ADC0 **0x3C-0x3F** - ADC4-7 - ADC1
		Single ended inputs: Vref : **[2.56V, AVcc]** Vinput : **[0, Vref]** Value = **[Vin *1024] / Vref**	**Diff inputs:** Vref: **[2.56V, AVcc - 0.5V]** Vpos/Vneg: **[0V,AVcc]** Value = **[(Vpos - Vneg) * GAIN * 512] / Vref**						
0x7B	ADCSRB	**ADHSM**	**ACME**	**MUX5**		**ADTS**			
		Hi-speed mode Uses more power but allows higher precision	**Analog comparator minus input:** 0 - bandgap reference (1.1V) 1 - ADC mux single ended input (if ADC is disabled)	**5th bit of the ADCMUX** See MUX3..MUX0		**ADC auto trigger source:** 0 - free running (prev conversion) 1 - analog comparator 2 - INT0	**3** - Timer0 comparator A **4** - Timer0 overflow	**5** - Timer1 comparator B **6** - Timer1 overflow **7** - Timer1 capture event	**8** - Timer4 overflow **9** - Timer4 comparator A **A** - Timer4 comparator B **B** - Timer4 comparator D
0x7A	ADCSRA	**ADEN**	**ADSC**	**ADATE**	**ADIF**	**ADIE**	**D**		
		Enable the ADC it is advised to discard few samples, after the ADC was enabled Inputs should have 15 ohm or less impedence	**Start conversion** Use in manual mode and to kick free running mode	**ADC auto trigger enable** Use ADTS to select the auto trigger source.	**ADC interrupt flag** Auto cleared by ISR execution	Interrupt when ADIF == 1	**ADC clock divider: (0 - CLK/2, != 0 - CLK >> ADCPS)** * ADC Clock should be between 50 and 200 kHz to get 10 bit precision * First conversion after ADEN is set takes 25 ADC clocks (extended conversion) * Regular ADC conversion takes 13 ADC clocks * Sample and hold happens 1.5 ADC clocks after start * Diff conversion is synced to ADCclk/2, and thus takes 13 or 14 ADC clocks. (It also needs stable prescaler, thus if diff conversion started by async trigger, ensure that it is an extended conversion. (by turning ADC on/off after conversion))		
0x78/79	ADCL/H	**ADC result - 10 bit (aligned to right or left)**							

Analog comparator

0x30	ACSR	**ACD**	**ACBG**	**ACO**	**ACI**	**ACIE**	**ACIC**	**ACIS1**	**ACIS0**
		DIsable the analog comparator when set Analog compartor should be disabled in sleep modes when not needed, especially when it uses bandgap	**Analog comparator plus input:** **0 -** AIN0+ **1 -** bandgap reference (1.1V) Bangap reference is enabled when this is set, ADC is enabled or BOD is. If you enable it using one of these, wait a bit for it to stablize	**Analog comparator value:** 0 - positive < negative 1 - positive > negative	**Analog comparator interrupt flag.** Auto cleared by ISR execution	**Interrupt when ACI == 1** Analog comparator can also trigger: **1.** input capture of timer 1 (ACIC) **2.** fault protection interrupt of timer4 **3.** ADC (ACME must be 0)	**TImer 1 input capture trigger:** 0 - ICP1 pin (default) 1 - Analog comparator	**Analog comparator interrupt mode:** **0** - ACO toggle **2** - ACO becomes 0 **3** - ACO becomes 1	

I2C module

0xBD	TWAMR	**TWAM** **I2C Slave address mask:** Each set bit, is a bit that is allowed to be both 1 and 0 in incoming slave address						
0xBA	TWAR	**TWA** **I2C Slave address** Note that address is 7 bit						**TWGCE** Enable answer to I2C general call
0xBC	TWCR	**TWINT** **Ready flag / Master slave selector Read:** 1 - I2c module is ready to be serviced 0 - sending/listening to our address **Write:** 1 - start sending data in master mode 0 - start listening in slave mode (TWEA should be 1) Clock is stretched while TWINT is set in slave mode	**TWEA** **Enable sending ACK on next transaction if needed.** Also used in slave transmit mode to indicate that we can't send more data	**TWSTA** **Send START condition** If TWSTO set too, will first send STOP, then START condition	**TWSTO** **Send STOP condition** Should be used only in the master mode	**TWWC** **Write collision to TWDR detected**. CPU attempted to write to this register while TWINT was low (module was busy)	**TWEN** **Enable the I2C module** Master module enable bit		**TWIE** Interrupt when TWINT == 1
0xBB	TWDR	**Data read/write (1 byte)** Should be read/written only when TWINT is 1 (Most of the times contains last byte present on the bus)						
0xB9	TWSR	**TWS - current state of I2C state machine** **Master mode:** **0x02/0x01** - (repeated) START transmitted Those is the states that are entered after start of I2C transaction **User expected response:** -> Send SLA+W - enters master write mode -> Send SLA+R - enters master read mode **0x07** - Bus arbitration lost when transmitting address/data/ack and address doesn't match slave address When this state is entered, the device doesn't touch the bus **User expected response:** -> Send START to wait for free bus and try again -> Do nothing to enter not addressed slave mode (responds to nothing) **Master write mode:** **0x03/0x04** - SLA+W transmitted; ACK/NACK received **0x05/0x06** - DATA transmitted; ACK/NACK received **User expected response:** -> Send first/next byte of DATA -> Send STOP -> Send Repeated START -> Send STOP+START **Master read mode:** **0x08/0x09** - SLA+R transmitted; ACK/NACK received **0x0A/0x0B** - DATA received; ACK/NACK transmitted **User expected response:** -> Receive first/next byte of DATA, ACK or NACK it (depends on TWEA) -> Send STOP -> Send Repeated START -> Send STOP+START		In slave mode, Initially the device is in not addressed slave mode It will ACK its address iff TWEA == 1 **Slave write mode (addressed):** **0x0C** - Own SLA+W received; ACK transmitted **0x0D** - Own SLA+W received; ACK transmitted; Arbitration lost in SLA+R/W as Master **0x0E** - General call address received; ACK transmitted **0x0F** - General call address received; ACK transmitted; Arbitration lost in SLA+R/W as Master **User expected response:** -> Receive first byte of DATA, ACK or NACK it (depends on TWEA) **0x10** - DATA received; ACK transmitted **0x12** - DATA received; ACK transmitted (after general call) **User expected response:** -> Receive next byte of DATA, ACK or NACK it (depends on TWEA) **0x11** - DATA received; NACK transmitted **0x13** - DATA received; NACK transmitted (after general call) Those states are entered after NACKing one byte from state 0xC/0xD/0xE/0xF/0x10/0x12 **User expected response:** -> Do nothing to enter not addressed slave mode, waiting for STOP or REPEATED START **0x14** - A STOP condition or repeated START condition has been received while still addressed as Slave This state is entered when host normally releases the bus **User expected response:** -> Do nothing to enter not addressed slave mode	**Slave read mode (addressed):** **0x15** - Own SLA+R received; ACK transmitted **0x16** - Own SLA+R received; ACK transmitted; Arbitration lost in SLA+R/W as Master **User expected response:** -> Send first byte of DATA. (Set TWEA == 0 iff last byte) **0x17** - DATA transmitted; ACK received **User expected response:** -> Send next byte of DATA. Set TWEA == 0 iff last byte **0x18** - DATA transmitted; NACK received **User expected response:** -> Do nothing to enter not addressed slave mode (Expected state after sending last byte) **0x19** - Last DATA byte transmitted (TWEA = "0"); ACK received **User expected response:** -> Do nothing to enter not addressed slave mode (Not expected state after sending last byte) **Other:** **0x00** - Bus error due to an illegal START or STOP **User expected response:** -> Set STOP bit (resets the i2c controller) **0x1F** - No relevant state information available; TWINT = "0", IDLE and state in between **User expected response:** none	**TWPS** **I2C bit rate prescaler value** See I2C bit rate The interface can work in sleep mode using the i2c clock to detect being addressed in slave mode. If it detects this, it will wake up the CPU while stretching the bones (the clock....)	
0xB8	TWBR	**I2C bit rate for master mode** CLK / [16+(TWBR<<(1+TWPS*2))] TWBR should be at least 10 for master mode For slave mode the SCL frequency given, must be at least 16 times less the CPU frequency						

USART module

0xCE	UDR1	**FIFO Data read/write (1 byte):** **RX fifo** : 2 bytes. Read from this byte, gets the oldest received byte and pops it from the fifo. As soon as the shift register is full, it will be added to the fifo. **TX fifo**: 1 byte. Every time the shift register is done sending, a new byte will be taken from here. Each "byte" is actually 9 bit internally and RXB01 and TXB81 are the 8th bit extensions of this fifo window						
0xCC 0xCD	UBRR1L UBRR1H	**Baud rate (12 bit)** **Async mode**: CLK / [16*(UBRR+1)]. Each bit is sampled 16 times, and the sample duration is 1+UBRR **Async mode + 2X**: CLK / [8*(UBRR+1)] Each bit is sampled 8 times, and the sample duration is 1+UBRR **Sync master mode /SPI mode**: CLK / [2*(UBRR+1)] **Sync slave mode:** clock is given on XCK1						
0xCB	UCSR1D						**CTSEN** **Enable CTS sensing.** Will delay copying of data from TX fifo to shift register when ~CTS line is high	**RTSEN** **Enable RTS output**. ~RTS will be high when RX fifo contains 2 bytes. One more byte can be received into the shift register
0xCA	UCSR1C	**UMSEL1** **Main mode selection:** **0** - **Async** (regular UART) **1** - **master/slave sync mode** (Bits synced on clock on XCK1 pin, start bit still starts RX) Direction of XCK1 pin selects master/slave sync mode **2** - **SPI master** SPI compatible maser sync mode. RX and TX are started at same time when UDR1 is written. TX is mandatory, RX is optional and done if RXEN1 is set		**UPM1** **Parity mode:** **0** - Disabled **2** - Even **3** - Odd NOT USED in SPI mode		**USBS1** **Stop bit count:** **0** - 1 bit **1** - 2 bits NOT USED in SPI mode	**UCSZ11** **Char size:** **0** - 5 bit **1** - 6 bit **2** - 7 bit **3** - 8 bit / 9 bit **Data order in SPI mode** Same as in SPI module	**UCSZ10** **Clock phase in SPI mode** Same as in SPI module	**UCPOL** **Clock polarity for sync/SPI mode** Same as in SPI module
0xC9	UCSR1B	**RXCIE1** Generate interrupt when RXCIF1 == 1	**TXCIE1** Generate interrupt when TXCIF1 == 1	**UDRIE1** Generate interrupt when UDRIF1 == 1	**RXEN1** **Receive enable** disable is immediate, data in RX fifo is lost	**TXEN1** **Transmit enable** disable will still send all outgoing data and then disable	**UCSZ12** **9 bit comm mode** Only when UCSZ10 ==3 NOT USED in SPI mode	**RXB81** **9th received bit** Should be read prior to reading from UDR1 NOT USED in SPI mode	**TXB81** **9th bit to transmit** Should be written prior to writing to UDR1 NOT USED in SPI mode
0xC8	UCSR1A	**RXCIF1** **RX fifo contains data** (and can be read via UDR1) (1 or 2 bytes)	**TXCIF1** **TX fully done** (TX fifo empty and shift register contents sent) Auto cleared by ISR	**UDRIF1** **TX FIFO has space to write to** (via UDR1) (since it is only one byte, this means that it is empty)	**FE1** **RX Frame error** (first stop bit of the next character in the receive buffer is zero) NOT USED in SPI mode	**DOR1** **RX Data overrun** (2 byte receive buffer full, another byte in RX shift register and new start bit detected) NOT USED in SPI mode	**PE1** **RX Parity error** NOT USED in SPI mode	**U2X1** **2x baud rate** Decrease number of clock samples from 16 to 8, thus doubling the baud rate Only used in async mode	**MPCM1** **Ignore frames for which address bit (bit 9) is 0** (meaning that this is data frame) Used for interprocessor communication mode (9 bit mode): NOT USED in SPI mode

SPI module

0x4E	SPIDR	**SPI data** Read from RX byte buffer, Write directly to the shift register, and will start transmission if master or if SS is low						
		Master mode: TX and RX are started by writing an byte to this register and when shift register is fully shifted, the SPIF is set and transmission halted You can read the received byte from this register If ~SS pin is input, and it goes low, the device will switch to slave mode (collision detection), and also set the SPIF flag				**Slave mode:** First transfer is started when SS is pulled low assuming that SPIDR is already written and ends when the byte is fully shifted out. Assuming that SS is still low, next transfer is started when a new byte is written to the shift register. You must use external means to make master not send new data byte while new SPI transfer is not continued by doing another write to this register. When master pulls ~SS high, SPI shift register is reset.			
0x4D	SPSR	**SPIF** **SPI interrupt flag**. Set when either of the conditions is true: **1.** done TX/RX, **2.** ~SS goes low in master mode (collision detection). In this case MSTR bit will be cleared to 0 Cleared when ISR is executed, or when reading SPSR and then SPIDR	**WCOL** **Write collision detected:** CPU wrote the data register while in transfer						**SPI2X** **Double the SPI bit rate** (for master mode):
0x4C	SOCR	**SPIE** Trigger interrupt when SPIF == 1	**SPE** **Enable the module** Master module enable bit	**DORD** **Data order:** **0** - LSB shifted first **1** - MSB shifted first	**MSTR** **SPI Master/Slave mode selector** 1 - Master mode 0 - Slave mode	**CPOL** **Clock polarity:** **0** - leading edge = rising, trailing edge = failing, idle level = low **1** - leading edge = failing, trailing edge = rising, idle level = high	**CPHA** **Clock phase:** **0** Setup on trailing edge Sample on leading edge **1** Setup on leading edge, Sample on trailing edge	**SPR** **SPI bitrate (for master mode):** 0 - CLK / 4 1 - CLK / 16 2 - CLK / 64 3 - CLK / 128 In slave mode, bit rate is set by the master and must be no more than CLK/4	

USB module (EP part)									
0xF2	UEBCLX	**BYCT**							
0xF3	UEBCHX								
		Byte count used in current EP bank. Will increase when writing to the bank, and decrease when reading from it.							
0xF1	UEDATX	**DAT**							
		a IO window to read/write data from/to current EP bank							
0xEF	UESTA1X						**CTRDIR**	**CURRBK**	
							CONTROL request direction 0 - OUT || 1 - IN (for debug/information) R/O	**Current FIFO bank number in use** (0/1) (for debug/information) R/O	
0xEE	UESTA0X	**CFGOK**	**OVERFI**	**UNDERFI**		**DTSEQ**		**NBUSYBK**	
		EP configuration OK bit R/O	**Packet overflow interrupt flag** Host wrote more data that bank can hold Must be cleared by ISR	**FIFO underflow/overflow interrupt flag:** There is no bank available to the controller to serve the host request Must be cleared by ISR		**Type of data packet** (for debug/information) **0** - DATA0 **1** - DATA1 R/O		**Number of busy banks in current EP** (banks owned by the controller) RX banks will be eventually given to the user, TX busy banks can be 'killed' with KILLBK bit R/O	
0xED	UECFG1X		**EPSIZE**			**EPBK**		**ALLOC**	
			Size of endpoint buffer (both banks) (8 << EPSIZE) **EP 0** - up to 64 bytes x 1 bank (control endpoint) **EP1** up to 256 bytes x 2 banks (or 512 x 1 bank) **EP2..6** - up to 32 bytes x 2 banks (or 64 x 1 bank)			**Number of banks assigned to this EP** **0** - single buffered **1** - double buffered CONTROL EP probably must be single banked		**Allocate memory for this EP** Set this bit to trigger FIFO memory allocation for this EP When deallocating non zero EP, the memory of EP with higher number slides down	
0xEC	UECFG0X	**EPTYPE**							**EPDIR**
		Type of the EP: **0** - Control **1** - ISO **2** - Bulk **3** - Interrupt							**Direction of this EP:** **0** - OUT (from host) **1** - IN (to host) Not used for control EP
0xEB	UECONTX			**STALLRQ**	**STALLRQC**	**RSTDT**			**EPEN**
				Send STALL on next host request on this EP (will auto reset when host sends SETUP)	**Stop sending STALL on next host request on this EP** Writing to this bit clears the STALLRQ Set this, then reset the EP	**Reset data toggle bit to DATA0** Used for resets when CLEAR_FEATURE usb commands asks for this			**Enable this EP** Disabling an EP, resets it and clears data toggle
0xF0	UEIENX	**FLERRE**	**NAKINE**		**NAKOUTE**	**RXSTPE**	**RXOUTE**	**STALLEDE**	**TXINE**
		Flow error interrupt enable Should only be used for ISO endpoints Interrupt flags in UESTA0X	Interrupt when NAKINI== 1		Interrupt when NAKOUTI== 1	Interrupt when RXSTPI== 1	Interrupt when RXOUTI== 1	Interrupt when STALLDI == 1	Interrupt when TXINI == 1
0xE8	UEINTX	**FIFOCON**	**NAKINI**	**RWAL**	**NAKOUTI**	**RXSTPI**	**RXOUTI/KILLBK**	**STALLDI**	**TXINI**
		Drop ownership of current FIFO bank Clear to end CPU ownership of current bank and let controller reuse it or send to the host Should not be used for control EP	**NAK IN sent interrupt flag** Must be cleared by ISR	**Current FIFO bank can be read/written** OUT: fifo not empty IN: fifo not full Should not be used for control EP	**NAK OUT sent interrupt flag** Must be cleared by ISR	**Valid SETUP received interrupt flag** Must be cleared by ISR	**Set when current bank contains OUT data received from the host which can be read** Must be cleared by ISR For IN endpoint, set this bit to kill last written bank	**STALL was sent to host flag** (in ISO mode: CRC error interrupt flag) Must be cleared by ISR	**Set when current bank is free and can be filled with IN data** Data will be given to host when it issues IN Must be cleared by ISR

USB module (general part)									
0xF4	UEINT		**EPINT**						
			Interrupt bit per EP Cleared by hardware when serviced						
0xEA	UERST		**EPRST**						
			Reset bit per endpoint Set a bit to reset an endpoint, and then clear it to complete the rest. Data Toggle is unchanged						
0xE9	UENUM						**EPNUM**		
							Selects current EP to access via EP specific registers (0..5)		
0xE6	UDMFN				**FNCERR**				
					CRC error in frame number of SOF (start of frame). R/O Set and cleared together with SOFI				
0xE4 0xE5	UDFNUML /H	**USB Frame number (11 bit)** Incremented on each valid Start of Frame packet. R/O							
0xE3	UDADDR	**ADDEN**	**UADDR**						
		Enable the USB address UADDR should be set first, and then ADDEN should be set	**USB address** Firmware must write here the value the host gave us						
0xE2	UDIEN		**UPRSME**	**EORSME**	**WAKEUPE**	**EORSTE**	**SOFE**		**SUSPE**
			Interrupt when UPRSMI== 1	Interrupt when EORSMI== 1	Interrupt when WAKEUPI== 1	Interrupt when EORSTI== 1	Interrupt when SOFI == 1		Interrupt when SUSPI == 1
0xE1	UDINT		**UPRSMI**	**EORSMI**	**WAKEUPI**	**EORSTI**	**SOFI**		**SUSPI**
			Remote wakeup request was sent interrupt flag Must be cleared by ISR	**Host successfully replied to remote wakeup request interrupt flag** Must be cleared by ISR	**Activity detected on USB bus interrupt flag** (async) When this bit is set, it clears SUSPI bit automatically Must be cleared by ISR otherwise	**End of USB reset detected interrupt flag** Must be cleared by ISR	**Valid Start of frame (SOF) detected interrupt flag**		**Inactivity on USB bus detected interrupt flag** When this bit is set, it clears SUSPI bit automatically Must be cleared by ISR otherwise
0xE0	UDCON					**RSTCPU**	**LSM**	**RMWKUP**	**DETACH**
						Reset the AVR cpu on detection of USB reset signal If set, will reset the CPU on end of USB reset	**Low speed mode** If set, the USB module will operate in low speed mode - this will connect the corresponding pull-up	**Send remote wakeup** (can only be done when SUSPI is set) Will be sent automatically after 5 seconds of inactivity when the USB interface is active	**Detach the device from the USB bus** When set to 1, both USB bus pull ups will be disconnected, signaling the host that device is detached This is default on boot
0xDA	USBINT								**VBUSI**
									VBUS level change detected interrupt flag (async) Must be cleared by ISR
0xD9	USBSTA								**VBUS**
									Current measured logic value of VBUS Value is buffered
0xD8	USBCON	**USBE**		**FRZCLK**	**OTGPADE**				**VBUSTE**
		Enable for USB module (except the VBUS pad)		**Freeze USB module clock** When set the USB module clock is halted, however you can still access the USBCON, USBSTA, USBINT, UDCON,UDINT, UDIEN registers and the VBUSI, WAKEUPI interrupts work	**Enable VBUS pad** If disabled, will not detect any changes in VBUS levels				Interrupt when VBUSI == 1
0xD7	UHWCON								**UVREGE**
									Enable USB 3.3V internal regulator (used to create the bus level pullups)

IN means that host wants data from us. we can NACK him, OUT means that hosts wants to send us data, we can NACK him, but we shouldn't

EEPROM

0x4(1,2)	EEARL/H	**EEPROM address** (12 bit)					
0x40	EEDR	**EEPROM data** (8 bit read/write)					
0x3F	EECR			**EEPM**	**EERIE**	**EEMPE**	**EEPE**	**EERE**
				EEPROM programming mode: **0** - erase and write (3.4ms) **1** - erase only (1.8ms) **2** - write only (1.8ms)	Interrupt when EEPE == 0	**Allow to set EEPE** for 4 cycles (allows write to EEPE) Safety bit	**Do EEPROM write/erase** EEPROM writes/erases are timed from RC oscillator	**Execute EEPROM read** takes just one CPU cycle Can only be done when no writes are pending

Main interrupts

0x54	MCUSR			**USBRF** **Reset caused by USB module.** Reset on power on and can be reset by software	**JTRF** **Reset caused by JTAG module** Reset on power on and can be reset by software	**WDRF** **Reset caused by watchdog** Reset on power on and can be reset by software. When set, forces WDE to be on	**BORF** **Reset caused by brown out detection** Reset on power on and can be reset by software	**EXTRF** **Reset caused by external reset pin** Reset on power on and can be reset by software	**PORF** **Reset caused by power on** cleared by software only
0x6B	PCMSK0	**PCINT** Bit per PORTB pin. A set bit, allows this pin to trigger PCIF0 flag and interrupt if enabled							
0x3B	PCIFR								**PCIF0** **Pin change interrupt flag** Auto cleared by ISR
0x68	PCICR								**PCIE0** Interrupt when PCIF ==1
0x6A	EICRB			**ISC6** **Interrupt trigger mode for int 6** 0 - low || 1 - any edge || 2 - failing edge || 3 - rising edge Only level interrupt can be detected in async mode					
0x69	EICRA	**ISC3** **Interrupt trigger mode for int 3** 0 - low ||1 - any edge || 2 - failing edge || 3 - rising edge		**ISC2** **Interrupt trigger mode for int 2** 0 - low || 1 - any edge || 2 - failing edge || 3 - rising edge		**ISC1** **Interrupt trigger mode for int 1** 0 - low || 1 - any edge || 2 - failing edge || 3 - rising edge		**ISC0** **Interrupt trigger mode for int 0** 0 - low || 1 - any edge || 2 - failing edge || 3 - rising edge	
0x3D	EIMSK		**INT6** Interrupt when INTF6 == 1			**INT3** Interrupt when INTF3 == 1	**INT2** Interrupt when INTF2 == 1	**INT1** Interrupt when INTF1 == 1	**INT0** Interrupt when INTF0 == 1
0x3C	EIFR		**INTF6** **INT6 Flag** Auto cleared by ISR			**INTF3** **INT3 Flag** Auto cleared by ISR	**INTF2** **INT2 Flag** Auto cleared by ISR	**INTF1** **INT1 Flag** Auto cleared by ISR	**INTF0** **INT0 Flag** INT0 can also trigger: **1.** timer4 fault protection. **2.** ADC Auto cleared by ISR

Power Management

0x65	PRR1	**PRUSB** Turn off USB module			**PRTIM4** Turn off Timer4	**PRTIM3** Turn off TImer3			**PRUSASRT1** Turn off USART
0x64	PRR0	**PRTWI** Turn off I2C		**PRTIM0** Turn off Timer0		**PRTIM1** Turn off Timer1	**PRSPI** Turn off SPI		**PRADC** Turn off ADC
0x7D	DIDR2			**ADC13D** Disable digital functions on ADC13 input	**ADC12D** Disable digital functions on ADC12 input	**ADC11D** Disable digital functions on ADC11 input	**ADC10D** Disable digital functions on ADC10 input	**ADC9D** Disable digital functions on ADC9 input	**ADC8D** Disable digital functions on ADC8 input
0x7F	DIDR1								**AIN0D** Disable digital functions on AIN0+ pin
0x7E	DIDR0	**ADC7D** Disable digital functions on ADC7 input	**ADC6D** Disable digital functions on ADC6 input	**ADC5D** Disable digital functions on ADC5 input	**ADC4D** Disable digital functions on ADC4 input			**ADC1D** Disable digital functions on ADC1 input	**ADC0D** Disable digital functions on ADC0 input
0x53	SMCR					**SM** **Sleep mode type selection:** **0** - **IDLE** - only CPU clock stopped **1** - **ADC noise reduction** - CPU and IO clock stopped, ADC,USB,FLASH clock running **2/3** - **Powerdown/Powersave** - All clock stopped, clock source stopped **6/7** - **Standby/Extended standby** - All clocks stopped, clock source running **Always working**: int pins, i2c slave address match, watchdog, usb async interrupts			**SE** **Enable SLEEP instruction to enter sleep mode** When disabled SLEEP will behave as NOP

System clock settings

0x61	CLKPR	**CLKPCE** **Enable clock prescaler change for 4 cycles** Safety bit				**CLKPS** **System clock prescaler value:** [input clock >> CLKPS)] Can divide up to input clock / 256. Default value is loaded from CLKDIV fuse			
0xC7	CLKSTA							**RCON** **RC oscillator is running** Check this after enabling the clock before switching to the clock source	**EXTON** **External clock is running** Check this after enabling the clock before switching to the clock source
0xC6	CLKSEL1	**RCCLKSEL** **RC "clock source" selector.** Must be 0010b (2)				**EXCKSEL** **External clock selector** See fuses			
0xC5	CLKSEL0	**RCSUT** **RC oscillator startup time** (value from fuses) Useless to change as used only on chip startup		**EXSUT** **External clock startup time** Will be used next time the external clock is enabled (value from fuses)		**RCE** **RC Oscillator enable** Can be used to change the clock on the fly	**EXTE** **External oscillator/clock enable** Can be used to change the clock on the fly		**CLKS** **Clock selector:** 0 - RC oscillator 1 - External clock
0x67	RCCTRL								**RCFREQ** **RC clock divider** **0** - RC has 8Mhz freq **1** - RC has 1Mhz freq
0x66	OSCAL	**RC freq calibration value**							

Hi-speed PLL (for USB and Timer4)

0x52	PLLFRQ	**PINMUX** **PLL input clock mux:** **0** - **system clock** (before system prescaler, must run at 8 or 16 Mhz depending on PINDIV) **1**-**RC oscillator** (must be at 8 Mhz)	**PLLUSB** **PLL output clock for USB:** **0** - PLL FREQ **1** - PLL FREQ / 2	**PLLTM** **PLL output clock for Timer4:** **0** - Disabled (Timer4 uses system clock) **1** - PLL FREQ **2** - PLL FREQ / 1.5 (Has some jitter) **3** - PLL FREQ / 2		**PDIV** **PLL internal frequency:** **3** - 40Mhz **4** - **48Mhz** **5** - 56 Mhz **7** - 72Mhz **8** - 80Mhz		**9** - 88Mhz **10** - **96Mhz** PLL must run at 48 Mhz or 96 Mhz to be able to use the USB module.	
0x49	PLLCSR				**PINDIV** **Select frequency the of the external clock must run at:** **0** - 8Mhz **1**- 16Mhz			**PLLE** **Enable the PLL** Set/Clear this bit to startup/shutdown the PLL	**PLOCK** **PLL is enabled and locked** After you enable the PLL, poll this bit untill it indicates that the PLL is done enabling. R/O

Watchdog Timer

0x60	WDTCSR	**WDIF** **Watchdog interrupt flag** If WDTON fuse is set, this bit will be forced to 0 Auto cleared by ISR routine	**WDIE** **Watchdog interrupt enable** if WDE==1, WDIE will autoclear on ISR entry, to reset CPU in case another WD event happens prior to this bit being set	**WDP3** **4th bit of watchdog clock divider** . See WDP0, WDP2 for info	**WDCE** **Watchdog settings change enable for 4 cycles** Must be written together with WDE = 1 allows change to WDE and prescaler bits Safety bit	**WDE** **Enable reset on Watchdog event, if WDIE is not set.** Note that when WDTON fuse set, or if the WD is the reset reason, this bit will be forced to 1	**WDP2**	**WDP1**	**WDP0** **Watchdog clock divider:** (2048 << WDPn) WD oscillator cycles (16<<WDPn) ms typical WD oscillator runs at 128KHz cycle rate

Timer0/TImer1/Timer3 Prescaler

0x43	GTCCR	**TSM** **Keep value written in PRSYNC bit -.** This allows to halt the prescaler							**PRSYNC** **Reset the prescaler.** Prescaler will start counting immediately unless TSM is set as well

Misc

0x55	MCUCR	**JTD** **Disable JTAG on the fly**. Software must write the new value twice within 4 cycles to take effect (for safety)			**PUD** **Disable all pull-ups** This bit allow to override enable of all pull-ups to avoid some undesired transitions			**IVSEL** **Move exception handlers table to start of the bootloader**	**IVCE** **Enable change to IVSEL bit for 4 clock cycles** Disables interrupts Safety bit
0x51	OCDR	**On Chip debug IO register:**							
		Can be written by the CPU to transfer data to the debugger. When read, bit 7 indicates that data was not yet read by the debugger							
0x3E	GPIOR0	Fast RAM like location with no purpose other than being used with fast IO instructions (SBI/CBI/SBIC/SBIS/IN/OUT)							
0x4A/B	GPIOR1/2	RAM like locations with no purpose other than being used with fast IO instructions (IN/OUT)							

Read-While-Write Self-Programming

0x57	SPMCSR	**SPMIE**	**RWWSB**	**SIGRD**	**RWWSRE**	**BLBSET**	**PGWRT**	**PGERS**	**SPMEN**
		Do an interrupt when SPMEN is 0 (after 4 cycles it is set or after completion of SPM instruction)	**Run while write section (application section) is busy** (can't execute from it) The section is all but last 2K of flash space	**Read factory data:** (use with SPMEN) **LPM/Z = 0:** - sig byte 1 **LPM/Z = 1:** - RC cal byte **LPM/Z = 2:** - sig byte 2 **LPM/Z = 3:** - sig byte 3	**Re-enable run while write section** Set to re-enable run while write (application) section after you done flashing it	**Read/write lock and read fuse bits** (use with SPMEN) **SPM:** Write lock bits (any Z) **LPM/Z = 0:** Read Fuse low byte **LPM/Z = 1:** Read lock bits **LPM/Z = 2:** Read extended fuse byte **LPM/Z = 3:** Read high fuse byte	**Write a flash page** (use with SPMEN) **SPM/Z=page address:** Write page using temp flash buffer Must write a page that was just erased Page is 64 bytes on this part	**Erase a flash page** (use with SPMEN) **SPM/Z=page address**: Erase an page Writes/Erases are timed from RC oscillator	**Activate SPM/Write flash buffer** If only SPMEN is written, SPM instruction will store R1:R0 to temp flash buffer addressed by the Z-pointer. Don't store more that once at same location. Store buffer will auto-erase after doing page write or setting RWWSRE

In-place Serial Programming (ISP)

To enable serial programming mode: 1. Apply power & clock as usual while reset and SCK is held low, or if SCK can't be held low on powerup, give reset 2 cpu clock positive pulse while SCK is low after powerup is complete	2 Wait 20 ms 3. Send programming enable command SCK must be at least 2x or 3x slower that CPU clock (if CPU clock less or greater than 12Mhz)	* Flash and EEPROM page writes are done by filling the programming buffer and then doing a page write. * EEPROM can also be written byte at time, and it auto erases prior to write * To write flash, chip must be first erased.	* Flash page buffer consists of words, each word has to be loaded as high and low byte. Low byte must be loaded first	* After each write command (including erase), poll using **0xF0**

Programming commands (sent MSB first on SPI interface):

Read Program Memory (Low)	**0x20**	Adr high	Adr low	value out	Programming Enable	**0xAC**	0x53	xx	xx
Read Program Memory (High)	**0x28**	Adr high	Adr low	value out	Chip Erase	**0xAC**	0x(8/9)X	xx	xx
Read Signature Byte	**0x30**	0x00	Index	value out	Program Flash page	**0x4C**	Adr high	Adr low	xx
Read Calibration Byte	**0x38**	0x00	0x00	value out	Program Low Fuse	**0xAC**	0xA0	xx	value in
Read Low Fuse bits	**0x50**	0x00	xx	value out	Program High Fuse	**0xAC**	0xA8	xx	value in
Read Lock bits	**0x58**	0x00	xx	value out	Program Extended Fuse	**0xAC**	0xA4	xx	value in
Read Extended Fuse Bits	**0x50**	0x08	xx	value out	Program Lock bits	**0xAC**	0x(E/F)X	xx	value in
Read Fuse High bits	**0x58**	0x08	xx	value out	Program EEPROM (Byte)	**0xC0**	Adr high	Adr low	value in
Read EEPROM	**0xA0**	Adr high	Adr low	value out	Write EEPROM page buffer	**0xC1**	0x00	Page Offset	value in
Write flash page buffer (Low)	**0x40**	xx	Page Offset	value in	Program EEPROM Page	**0xC2**	Adr high	Adr low	xx
Write flash page buffer (High)	**0x48**	xx	Page Offset	value in	Poll RDY/~BUSY	**0xF0**	0x00	xx	value out (bit 0 == 0 iff busy)
Load Extended Address Byte	**0x4D**	0x00	Adr ext	xx					

High Voltage Parallel Programming (HVPP)

Steps to enter programming mode:
0. Apply +5V on AVCC and VCC
1. Set low on RESET,
2. Toggle XTAL1 (clock) 6 times
3. set 4 high bits in PORTD to 0 for 100 ns
4. Apply +12V to RESET pin and
5. Wait 50 ms

Steps to read/write an register:
1. Set
 * register using XA pins,
 * register part using BS pins
 * DATA if write,
 * OE to indicate read/write
2. Give XTAL a pulse
3. Read value from DATA pins if read
4. Repeat 1-3 for other parts of the register if needed.
5. Return OE to 1

Steps to do flash/eeprom read/sig/cal byte/fuse read
1. Load the command to the command register

2. Load address of the row to read to address register
 * **flash**: has 3 byte row address, each row has 2 columns
 * **eeprom** has 2 byte address, each row has 1 column
 * **signature/calibration:** 1 byte address, 3 rows
  first column has the signature
  second column has the calibration byte in first row.
 * **fuse row** doesn't need to load an address

3. set BS to select which column to read.

4. As soon as OE is set to low, the byte will be available on data bus
5. Return OE to 1

Steps for chip erase command:
1. Load chip erase command to command register
2. Give WR negative pulse and wait for completion

Steps to do flash/eeprom write:
1. Load the command to command register
2.For each word in the page (flash has 2 byte words, eeprom 1 byte)
 a. load word page offset in the address register (only low byte usually),
 b. load data to data register (two halves using BS for flash)
 c. move the data to the flash buffer by issuing a high pulse on PAGEL.

3. Load full page address to the address register (3 bytes for flash, 2 bytes for eeprom)
4. Give WR negative pulse and wait for completion

Flash has 64 byte pages, EEPROM has 4 byte pages

Steps to do fuse/lock write
1. Load the command to the command register
2. Set BS1/BS2 to the byte address to write:
 * **Fuse**: 0 - low, 1 - high, 2 - extended
 * **Lock**: 0
3. Set ~OE to high, DATA to the value to write
4. Give WR negative pulse and wait for completion

Each fuse byte can be written separately

Wait:
After all write commands, wait till ~BUSY pin is high (buy it some weed if needed)

Set pins as written below, set AVCC and VCC to +5V. All other pins should be left floating

PD7	PD6	PD5	PE2	PD4	PD3	PD2	PD1	XTAL1	RESET
PAGEL	**XA1**	**XA0**	**BS2**	**BS1**	**~WR**	**~OE**	**RDY / ~BUSY**	**XTAL**	**~RESET**
Move flash data register to programming page buffer at address given by address register Address register must be set to page offset. Data register set with data to program. Then give this pin high pulse. Must be low during programming entry (Those pins are inputs)	**Select internal register to read/write from/to DATA on XTAL pulse:** **0** - address (24 bit, BS selects the byte) **1** - data (16 bit, BS selects the byte) **2** - command register (8 bit) **3** - no register Must be low during programming entry (This pin is input)		**Select offset in the register / fuses offset:** **Flash/EEPROM address:** select address byte **Flash data:** Selects low/high byte to load **Sig/Cal:** selects which one to read (Cal = 1) **Fuses write :** 0 - low, 1 - high, 2 - extended **Lock write**: 0 - only offset **Fuse read:** 0 - low, 1 - lock, 2-extended, 3-high Should be set to 0 when unused. Must be low during programming entry (Those pins are inputs)		**Execute write/erase** Low Pulse on this pin executes flash/eeprom/fuse write/erase For flash wites don't forget to load the flash buffer. (This pin is input)	**Select DATA bus direction:** 0 - DATA is output 1 - DATA is input Return the value of this input to 1 when not used to avoid trouble (this pin is input)	**Indicates completenes of write/erase command:** 0 - device busy programming 1 - device ready for new command (this pin is output)	Pulse on this pin loads the programming command / data Set **XA,BS** and in case of input the **DATA** first (This pin is input)	Used with +12V to enter programming mode (see above)
Port B									
Bi-directional 8 bit data bus for read or write the flash registers ~OE selects the direction of this bus									

Programming commands:

Chip erase command:	**0x80** - Load Chip erase command	**Special row read/write commands:**	**0x04** - Load Read fuse and lock bits
Flash/EEPROM read commands:	**0x02** - Load Read FLASH command		**0x40** - Load Write fuse bits command
	0x03 - Load Read EEPROM command		**0x08** - Load Read signature and calibration byte
Flash/EEPROM write commands:	**0x10** - Load Write FLASH command		**0x20** - Load Write lock bits command
	0x11 - Load Write EEPROM command		**0x00** - No command

Fuse bytes

0	**LOW**	**CKDIV8**	**CKOUT**	**SUT**		**CKSEL**			
		Set main clock prescaler to divide input clock by 8	Enable main CPU clock output on CLK0 pin	**Startup clock startup time delay** (See table below)		**External clock selector:** 0 - TTL clock on XTAL0 1 - RC oscillator 2 - Low frequency crystal (watch crystal) 4 - Low power crystal (0.4 - 0.9 Mhz) 5 - Low power crystal (0.9 - 3.0 Mhz) 6 - Low power crystal (3.0 - 8.0 Mhz) 7 - Low power crystal (8.0 -16.0 Mhz)			**CKSEL[0] - affects clock startup time** (See table below)
1	**LOCK**			**BLB11**	**BLB10**	**BLB02**	**BLB01**	**LB2**	**LB1**
		1	1	**LPM of bootloader section lock** 1 - no lock 0 - LPM in application section can't read data from bootloader section	**SPM of bootloader section lock** 1 - no lock 0 - SPM not allowed to write bootloader section	**LPM for application section lock** 1 - no lock 0 - LPM in bootloader section can't read data from application section	**SPM for application section lock** 1 - no lock 0 - SPM not allowed to write application section	**Programming lock** 3 - no lock 2 - programming disabled, fuse bits locked 0 -in addition, lock bits are locked (These bits can't be set with SPM) Only way out is chip erase	
2	**EXTENDED**					**HWBE**	**BODLEVEL**		
		1	1	1	1	**Alow ~HWB pin to be used to force bootloader entry** 1 - ~HWB pin is not used on boot 0 - ~HWB pin can be used to force boot vector at start of bootloader section by setting it to LOW value	**Brown out detection level: (min/typical/max)** 7 - disabled 6 - 1.8V | 2.0V | 2.2V 5 - 2.0V | 2.2V | 2.4V	4 - 2.2V | 2.4V | 2.6V 3 - 2.4V | 2.6V | 2.8V 2 - 3.2V | 3.4V | 3.6V	1 - 3.3V | 3.5V | 3.7V 0 - 4.0V | 4.3V | 4.5V Compare vs internal bandgap
3	**HIGH**	**OCDEN**	**JTAGEN**	**SPIEN**	**WDTON**	**EESAVE**	**BOOTSZ**		**BOOTRST**
		OCD enable/disable 1 - Disable OCD 0 - Enable OCD	**JTAG enable/disable** 1 - Disable JTAG 0 - Enable JTAG JTAG must be enabled when OCD is enabled	**Serial programming** 1- Disable 0 - Enable Can't be set to 1 from serial programming mode	**Force enable the watchdog** 1 - Watchdog on when enabled by the CPU 0 - Watchdog always on and in reset mode	**Erase EEPROM on chip erase** 1 - EEPROM preserved 0 - EEPROM is erased	**Size/Start of bootloader area in bytes** ROMEND - (2048 >> BOOTSZ)		**Move the boot vectror from 0 to start of the bootloader section** 1 - boot vector at 0 0 - boot vector at start of bootloader section

Startup clock time delay (CK timed from the clock, delay timed from WD oscillator)

	Scenario	Startup from powerdown/powersave	Additional delay from reset	CKSEL[0], SUT[1:0]
External clock / RC oscillator	BOD on	6CK	14CK	000
	Fast rising power	6CK	14CK + 4.1ms	001
	Slowly rising pwr	6CK	14CK + 65ms	010
Low frequency oscillator	BOD on	1K CK	14CK	000
	Fast rising power	1K CK	14CK + 4.1ms	001
	Slowly rising pwr	1K CK	14CK + 65ms	010
	BOD on	32K CK	14CK	100
	Fast rising power	32K CK	14CK + 4.1ms	101

	Scenario	Startup from powerdown/powersave	Additional delay from reset	CKSEL[0], SUT[1:0]
Low power oscillator	Ceramic resonator, fast rising power	258CK	14CK + 4.1ms	000
	Ceramic resonator, slowly rising power	258CK	14CK + 65ms	001
	Ceramic resonator, BOD on	1K CK	14CK	010
	Ceramic resonator, fast rising power	1K CK	14CK + 4.1ms	011
	Ceramic resonator, slowly rising power	1K CK	14CK + 65ms	100
	Crystal Oscillator, BOD on	16K CK	14CK	101
	Crystal Oscillator, fast rising power	16K CK	14CK + 4.1ms	110
	Crystal Oscillator, slowly rising power	16K CK	14CK + 65ms	111

JTAG general (Mandatory)

OP	Instruction name	Description	Register selected	Capture DR	Update DR	Run Test/Idle
0x1	**IDCODE**	Reads chip ID	**Boundary scan register** (88 bits) - (see below)	Boundary scan register is is loaded with Chip ID		
0xF	**BYPASS**	Enables TDI/TDO bypass	**Bypass register** (1 bit) Register is connected directly, so update in Shift DR state is done immediately	Loads 0 to BYPASS register when leaving this state This allows to disable the bypass using TMS input only		
0xC	**RESET**	Enters/Exits reset state	**Reset register** (1 bit) Register is connected directly, so update in Shift DR state is done immediately			

JTAG Boundary scan (Mandatory)

OP	Instruction name	Description	Register selected	Capture DR	Update DR	Run Test/Idle
0x0	**EXTEST**	Read and write boundary Output latches are connected to pins as soon as instruction is loaded to the IR, and only this instruction **Thus for first use, USE SAMPLE_PRELOAD to avoid damage**	**Boundary scan register** (88 bits) (see below)	Boundary scan register is loaded with pin state.	Updates the output latches for all pins from Boundary scan register	
0x2	**SAMPLE_PRELOAD**	Read boundary, and preloads the latches for the EXTEST	**Boundary scan register** (88 bits) (see below)	Boundary scan register is loaded with pin state.	Updates the output latches for all pins from Boundary scan registers. However since this isn't EXTEST instruction, the output latches are not connected to the pins	

Non reserved bits in the boundary scan described below.
DIR bit corresponds to input/output (DDR) and DATA bit to the PORTX value (pullup for the inputs, and output value for the outputs)
All reserved bits should be kept to 0
RSTT is observe only value of the reset logic

67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82
PB7.DIR	**PB7.DATA**	**PB6.DIR**	**PB6.DATA**	**PB5.DIR**	**PB5.DATA**	**PB4.DIR**	**PB4.DATA**	**PB3.DIR**	**PB3.DATA**	**PB2.DIR**	**PB2.DATA**	**PB1.DIR**	**PB1.DATA**	**PB0.DIR**	**PB0.DATA**

46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
PD7.DIR	**PD7.DATA**	**PD6.DIR**	**PD6.DATA**	**PD5.DIR**	**PD5.DATA**	**PD4.DIR**	**PD4.DATA**	**PD3.DIR**	**PD3.DATA**	**PD2.DIR**	**PD2.DATA**	**PD1.DIR**	**PD1.DATA**	**PD0.DIR**	**PD0.DATA**

62		2	3	0	1		87	88	24	25		26	27	28	29
RSTT		**PF1.DIR**	**PF1.DATA**	**PF0.DIR**	**PF0.DATA**		**PE6.DIR**	**PE6.DATA**	**PE2.DIR**	**PE2.DATA**		**PC7.DIR**	**PC7.DATA**	**PC6.DIR**	**PC6.DATA**

JTAG Memory programming (Vendor/Public)

OP	Instruction name	Description	Register selected	Capture DR	Update DR	Run Test/Idle
0x4	**PROG_ENABLE**		**Flash enable signature register** (16 bits)		Flash enable signature register is checked against enable signature 0xA370	
0x5	**PROG_COMMANDS**	**Apply inputs to the flash programming unit** This is more or less the same as setting the pins in HVPP mode	**Flash program register** Low 8 bit DATA, high 7 bit command (see below)	Flash program register is updated with result from the flash unit	Flash program register value is applied to flash inputs	Flash unit is executing the command (usually only one clock) **Same as pulsing XTAL in Parallel programming**
0x6	**PROG_PAGELOAD**	**Write data to flash program buffer** This can be used to do faster write of the flash data buffer instead of dong the slow HVPP like procedure. Each such command writes a byte to the buffer (toggles high/low automaticallly) You only need to specify full page aligned address prior to using this instruction	**Flash program register** (low 8 bit)		Flash page buffer is written byte after byte	You need 11 TCK clocks for the write to be done from the moment you entered UpdateDR state. If less was sent, spend some of the clocks in this state
0x7	**PROG_PAGEREAD**	**Read data from flash (up to a page)** This can be used to do faster read of the flash rather that using HVPP like procedure. You only need to setup the full page aligned address prior to sending this command	**Flash program register** (low 8 bit)	Flash is read byte, after byte		

Programming register - each bit is mostly same as input in HVPP.
Differences highlighted.

PAGEL	XA1	XA0	BS2	BS1	~WR / ~BUSY	~OE
Move flash data register to programming page buffer at address given by address register As in HVPP	**Select internal flash register to modify:** As in HVPP		**Select offset in the register / command sub-argument:** As in HVPP		**Write: Low pulse to start flash write/erase** (as in HVPP) **Read: After starting a write, poll until bit has high value** (as in HVPP)	**Select DATA bus direction** (as in HVPP) Return the value of this bit to 1 (meaning input direction) when not used to avoid trouble Register read/write will happen in run-test/idle state.

DATA (8 bit)
Bi-directional 8 bit data bus for read or write of the internal flash registers As in HVPP

JTAG On chip debug (Vendor/Private)

OP	Instruction name	Description	Register selected	Capture DR	Update DR	Run Test/Idle
0x8	**HALT**	Forces the CPU to halt				
0x9	**CONTINUE**	Resumes the CPU execution				
0xA	**EXEC_INSTR**	Runs an AVR instruction	**Internal scan chain** (16 bit)	Capture PC to Internal scan chain	Internal scan chain is copied to instruction register.	AVR CPU is executing the instruction (let debugger be in this state for # of cycles needed)
0xB	**OCD_COMMAND**	Reads/write Breakpoint scan chain which allows access to OCD registers	**Breakpoint scan chain**: 1 bit RW, then 4 bits register, then 16 bits register value	Capture result from breakpoint unit to the chain	Apply breakpoint chain to breakpoint unit	

Breakpoint scan chain registers:
To read an OCD register, first write its address + R bit, enter UpdateDR state, then read the register

0x0	PSB0	Program single breakpoint 0															
0x1	PSB1	Program single breakpoint 1															
0x2	PDMSB	Program/Data Mask or Single breakpoint															
0x3	PDSB	Program/Data or Single Breakpoint															
0x8	BCR	**RUNTIMER** Enable timers to run while stopped	**PC24** Selects between reading PC+2 and PC+4 during a Break	**SSTEP** Enable single stepping	**FLOW** Enable break on flow change	**PSB0_M** **PSB0 mode:** 0 - disabled 1 - enabled	**PSB1_M** **PSB1 mode:** 0 - disabled 1 - enabled	**PDMSB_M** **PDMSB mode:** 0 - disabled 1 - enabled 2 - mask for PDSB		**PDSB_M** **PDSB mode:** 0 - disabled 1 - enabled	**PDMSB_BC** **PDMSB break condition:** 3 - code breakpoint 2 - data read/write 1 - data write 0 - data read		**PDSB_BC** **PDSB break condition:** 3 - code breakpoint 2 - data read/write 1 - data write 0 - data read		**BCR2** ?	**BCR1** ?	**BCR0**
0x9	BSR								**STEPB** Break due to single stepping	**FLOWB** Break due to flow change	**PSB0B** Break due to PSB0 hit	**PSB1B** Break due to PSB1 hit	**PDMSBB** Break due to PDMSB hit	**PDSBB** Break due to PDSB hit	**BSR2** ?	**HALTB** Break due to HALT JTAG instruction	**BREAKB** Break due to BREAK instruction in the code
0xC	OCDR	**OCDR value**															
0xD	OCDC	**OCDRE** Enable OCDR	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?