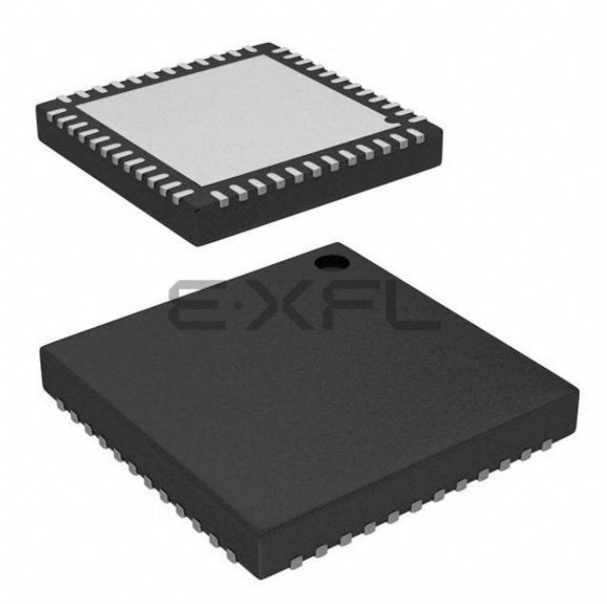


Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Obsolete
Core Processor	AVR
Core Size	8-Bit
Speed	15MHz
Connectivity	LINbus, SPI, UART/USART, LINbus-SBC
Peripherals	Brown-out Detect/Reset, POR, WDT
Number of I/O	10
Program Memory Size	64KB (64K x 8)
Program Memory Type	FLASH
EEPROM Size	1K x 8
RAM Size	4K x 8
Voltage - Supply (Vcc/Vdd)	3V ~ 3.6V
Data Converters	A/D 1x17b Sigma Delta, 1x18b Sigma Delta
Oscillator Type	Internal
Operating Temperature	-40°C ~ 125°C (TA)
Mounting Type	Surface Mount
Package / Case	48-VFQFN Exposed Pad
Supplier Device Package	48-VQFN (7x7)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atmega64hve2-plpw

1. Description

With the ATmega32HVE2/ATmega64HVE2 Atmel® provides an 8-bit AVR® microcontroller with very precise analog frontend for voltage and current measurement and 32bit computing power. The circuit is a complete single-package system solution for applications like, e.g., 12V lead acid or Li-ion battery monitoring or particle filtering in automotive applications.

The device includes 2 dies, the first die (AVR MCU) with the very precise analog frontend consisting of

- a 17bit and a 18bit sigma delta ADC
- programmable gain amplifier with various chopper modes and extreme low offset
- 8-bit microcontroller with 32bit math-extensions module and 32/64Kbytes flash memory

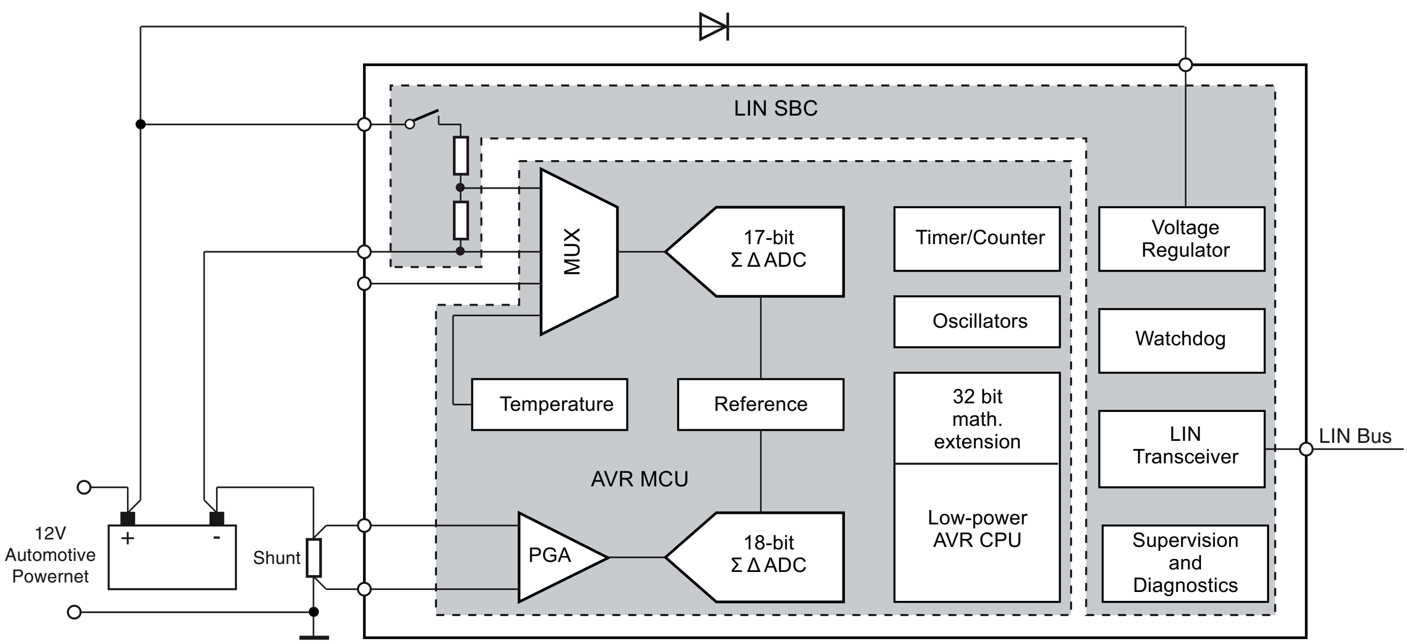
and a LIN(1) system basis chip (LIN SBC) including

- LIN transceiver according to the LIN2.0, 2.1 and SAEJ2602-2 standards
- 3.3V low drop voltage regulator
- window watchdog
- integrated voltage divider with reverse polarity protection for very precise sensing of the battery voltage

The device includes the same LIN SBC die as used in the Atmel ATA6628 LIN system basis chip from Atmel.

Note: 1. LIN: Local Interconnect Network

Figure 1-1. Atmel ATmega32HVE2/ATmega64HVE2 Block Diagram



Atmel LIN System Basis Chip (LIN SBC)

LIN Bus Transceiver with 3.3V Regulator and Watchdog

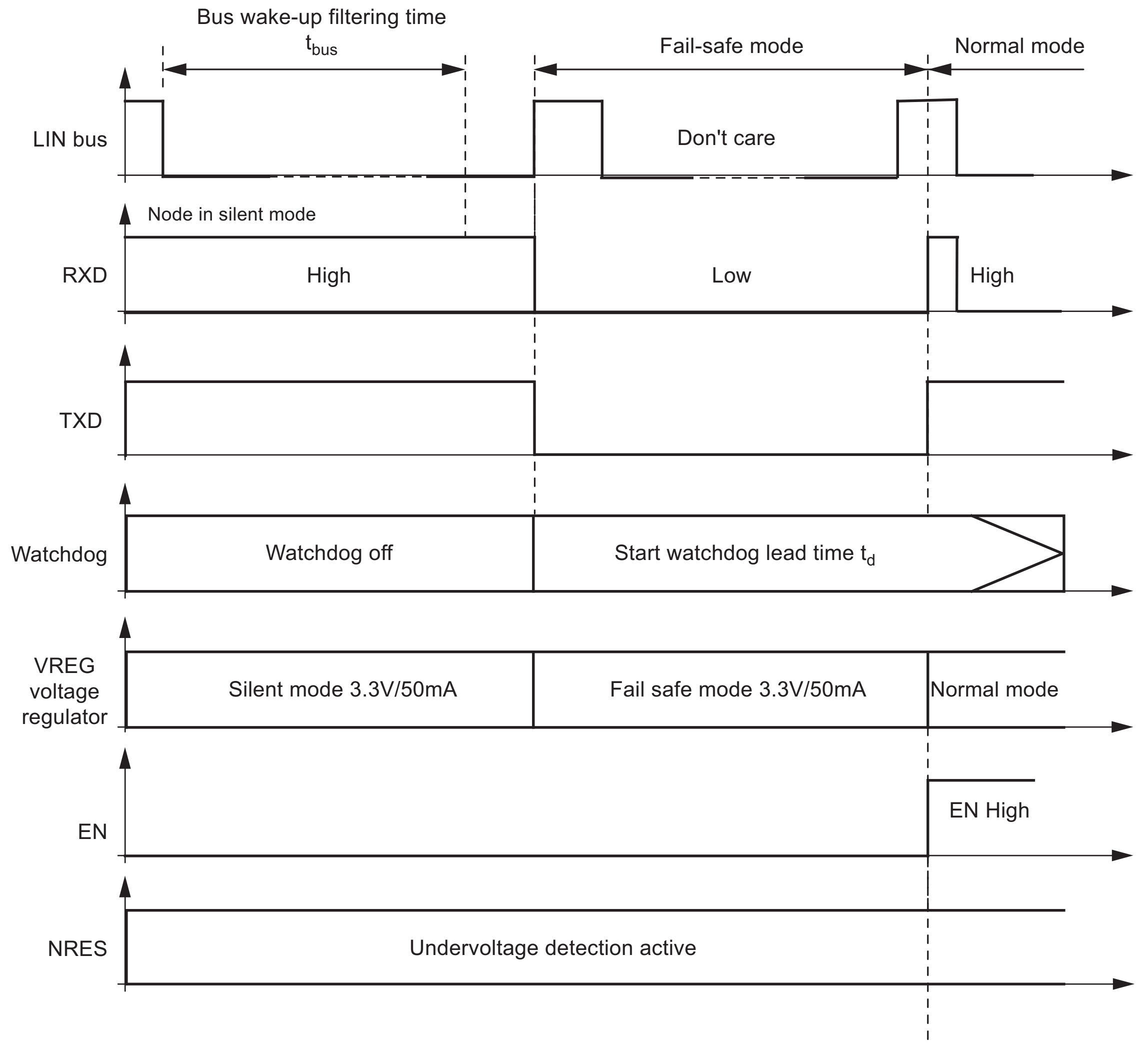
PRELIMINARY DATASHEET

Features

- Master and Slave Operation Possible
- Supply Voltage –27V to +40V
- Operating Voltage V_S = 5V to 27V
- Typically 10µA Supply Current During Sleep Mode
- Typically 40µA Supply Current in Silent Mode
- Linear Low-drop Voltage Regulator:
  - Normal, Fail-safe, and Silent Mode
    - V_{REG} = 3.3V ±2%
  - In Sleep Mode V_{REG} is Switched Off
- VREG-Undervoltage Detection (4ms Reset Time) and Watchdog Reset Logical Combined at Open Drain Output NRES
- High-speed Mode Up to 115kBaud
- Internal 1:24 Voltage Divider for $V_{Battery}$ Sensing
- Negative Trigger Input for Watchdog
- Boosting the Voltage Regulator Possible with an External NPN Transistor
- LIN Physical Layer According to LIN 2.0, 2.1 and SAEJ2602-2
- Wake-up Capability via LIN-bus
- Bus Pin is Overtemperature and Short-circuit Protected versus GND and Battery
- Adjustable Watchdog Time via External Resistor
- Advanced EMC and ESD Performance
- Fulfills the OEM “Hardware Requirements for LIN in Automotive Applications Rev. 1.1”
- Atmel® ATA6628 LIN SBC inside

A falling edge at the LIN pin followed by a dominant bus level maintained for a certain time period (t_{bus}) and the following rising edge at the LIN pin (see Figure 7-3) result in a remote wake-up request, which is only possible if TXD is high. The device switches from Silent Mode to Fail-safe Mode. The internal LIN slave termination resistor is switched on. The remote wake-up request is indicated by a low level at the RXD pin to interrupt the microcontroller (see Figure 7-3). EN high can be used to switch directly to Normal Mode.

Figure 7-3. LIN Wake-up from Silent Mode



7.5 Fail-safe Mode

The device automatically switches to Fail-safe Mode at system power-up. The voltage regulator is switched on (V_{REG} = 3.3V/2%/50mA) (see Figure 8-1 on page 22). The NRES output switches to low for t_{res} = 4ms and gives a reset to the microcontroller. LIN communication is switched off. The IC stays in this mode until EN is switched to high. The IC then changes to Normal Mode. A power down of V_{Bat} ($V_S < VS_{thU}$) during Silent or Sleep Mode switches the IC into Fail-safe Mode after power up. A low at NRES switches into Fail-safe Mode directly. During Fail-safe Mode, the TXD pin is an output and signals the fail-safe source. The watchdog is switched on.

The LIN SBC can operate in different Modes, like Normal, Silent, or Sleep Mode. The functionality of these modes is described in Table 7-2.

Table 7-2. TXD, RXD Depending from Operation Modes

Different Modes	TXD	RXD
Fail-safe Mode	Signalling fail-safe sources (see Table 7-3 and Table 7-4)	
Normal Mode	Follows data transmission	
Silent Mode	High	High

A wake-up event from either Silent or Sleep Mode will be signalled to the microcontroller using the two pins RXD and TXD. The coding is shown in Table 7-3.

A wake-up event will lead the IC to the Fail-safe Mode.

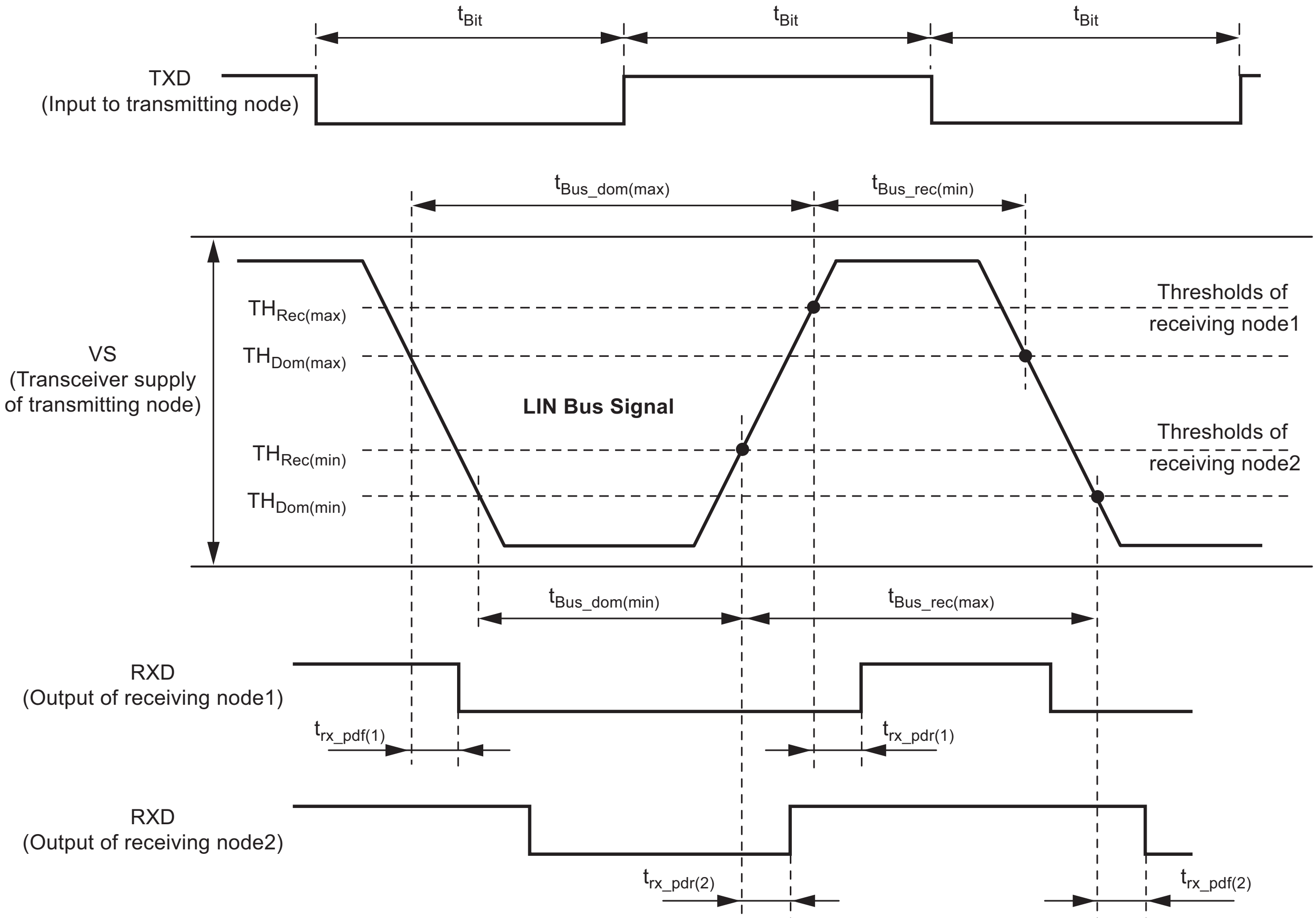
Table 7-3. Signalling Fail-safe Sources

Fail-safe Sources	TXD	RXD
LIN wake-up (pin LIN)	Low	Low
VS_{th} (battery) undervoltage detection	High	Low

Table 7-4. Signalling in Fail-safe Mode after Reset (NRES was Low), Shows the Reset Source at TXD and RXD Pins

Fail-safe Sources	TXD	RXD
VREG undervoltage at NRES	High	Low
Watchdog reset at NRES	High	High

Figure 10-1. Definition of Bus Timing Characteristics



- **Bit 4 – S: Sign Bit, S = N $\oplus$ V**

  The S-bit is always an exclusive or between the negative flag N and the Two's Complement Overflow Flag V. See the "Instruction Set Description" for detailed information.
- **Bit 3 – V: Two's Complement Overflow Flag**

  The Two's Complement Overflow Flag V supports two's complement arithmetics. See the "Instruction Set Description" for detailed information.
- **Bit 2 – N: Negative Flag**

  The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the "Instruction Set Description" for detailed information.
- **Bit 1 – Z: Zero Flag**

  The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the "Instruction Set Description" for detailed information.
- **Bit 0 – C: Carry Flag**

  The Carry Flag C indicates a carry in an arithmetic or logic operation. See the "Instruction Set Description" for detailed information.

13.4 General Purpose Register File

The Register File is optimized for the AVR Enhanced RISC instruction set. In order to achieve the required performance and flexibility, the following input/output schemes are supported by the Register File:

- One 8-bit output operand and one 8-bit result input
- Two 8-bit output operands and one 8-bit result input
- Two 8-bit output operands and one 16-bit result input
- One 16-bit output operand and one 16-bit result input

Figure 13-2 shows the structure of the 32 general purpose working registers in the CPU.

Figure 13-2. AVR CPU General Purpose Working Registers

	7 0	Addr	
	R0	0x00	
	R1	0x01	
	R2	0x02	
	…		
	R13	0x0D	
General	R14	0x0E	
Purpose	R15	0x0F	
Working	R16	0x10	
Registers	R17	0x11	
	…		
	R26	0x1A	X-register Low Byte
	R27	0x1B	X-register High Byte
	R28	0x1C	Y-register Low Byte
	R29	0x1D	Y-register High Byte
	R30	0x1E	Z-register Low Byte
	R31	0x1F	Z-register High Byte

Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions. As shown in Figure 13-2, each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user Data Space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y- and Z-pointer registers can be set to index any register in the file.

13.4.1 The X-register, Y-register, and Z-register

The registers R26..R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the data space. The three indirect address registers X, Y, and Z are defined as described in Figure 13-3 on page 38.

Figure 13-3. The X-, Y-, and Z-registers

	15	XH			XL	0
X-register	7		0	7		0
	R27 (0x1B)			R26 (0x1A)		

	15	YH			YL	0
Y-register	7		0	7		0
	R29 (0x1D)			R28 (0x1C)		

	15	ZH			ZL	0
Z-register	7		0	7		0
	R31 (0x1F)			R30 (0x1E)		

In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the instruction set reference for details).

13.5 Stack Pointer

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack Pointer Register always points to the top of the Stack. Note that the Stack is implemented as growing from higher memory locations to lower memory locations. This implies that a Stack PUSH command decreases the Stack Pointer.

The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. This Stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. The Stack Pointer must be set to point above 0x100. The Stack Pointer is decremented by one when data is pushed onto the Stack with the PUSH instruction, and it is decremented by two when the return address is pushed onto the Stack with subroutine call or interrupt. The Stack Pointer is incremented by one when data is popped from the Stack with the POP instruction, and it is incremented by two when data is popped from the Stack with return from subroutine RET or return from interrupt RETI.

The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present.

13.5.1 SPH and SPL – Stack Pointer High and Stack Pointer Low

Bit	15	14	13	12	11	10	9	8	
0x3E (0x5E)	**SP15**	**SP14**	**SP13**	**SP12**	**SP11**	**SP10**	**SP9**	**SP8**	**SPH**
0x3D (0x5D)	**SP7**	**SP6**	**SP5**	**SP4**	**SP3**	**SP2**	**SP1**	**SP0**	**SPL**
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

19.3 Moving Interrupts Between Application and Boot Space

The General Interrupt Control Register controls the placement of the Interrupt Vector table.

Assembly Code Example

```
Move_interrupts:
        ; Enable change of Interrupt Vectors
        ldi             r16, (1<<IVCE)
        out             MCUCR, r16
        ; Move interrupts to Boot Flash section
        ldi             r16, (1<<IVSEL)
        out             MCUCR, r16
        ret
```

C Code Example

```
void Move_interrupts(void)
{
        /* Enable change of Interrupt Vectors */
        MCUCR = (1<<IVCE);
        /* Move interrupts to Boot Flash section */
        MCUCR = (1<<IVSEL);
}
```

19.4 Register Description

19.4.1 MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	JTD	–	–	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R/W	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 1 – IVSEL: Interrupt Vector Select**

  When the IVSEL bit is cleared (zero), the Interrupt Vectors are placed at the start of the Flash memory. When this bit is set (one), the Interrupt Vectors are moved to the beginning of the Boot Loader section of the Flash. The actual address of the start of the Boot Flash Section is determined by the BOOTSZ Fuses. Refer to Section 29. "Boot Loader Support – Read-While-Write Self-Programming" on page 167 for details. To avoid unintentional changes of Interrupt Vector tables, a special write procedure must be followed to change the IVSEL bit:

  a. Write the Interrupt Vector Change Enable (IVCE) bit to one.

  b. Within four cycles, write the desired value to IVSEL while writing a zero to IVCE.

  Interrupts will automatically be disabled while this sequence is executed. Interrupts are disabled in the cycle IVCE is set, and they remain disabled until after the instruction following the write to IVSEL. If IVSEL is not written, interrupts remain disabled for four cycles. The I-bit in the Status Register is unaffected by the automatic disabling.

Note: If Interrupt Vectors are placed in the Boot Loader section and Boot Lock bit BLB02 is programmed, interrupts are disabled while executing from the Application section. If Interrupt Vectors are placed in the Application section and Boot Lock bit BLB12 is programed, interrupts are disabled while executing from the Boot Loader section. Refer to Section 29. "Boot Loader Support – Read-While-Write Self-Programming" on page 167 for details on Boot Lock bits.

- **Bit 0 – IVCE: Interrupt Vector Change Enable**

  The IVCE bit must be written to logic one to enable change of the IVSEL bit. IVCE is cleared by hardware four cycles after it is written or when IVSEL is written. Setting the IVCE bit will disable interrupts, as explained in the IVSEL description above. See Code Example below.

- **CKOUT/PCINT4 - Port B, Bit2**
  CKOUT: Clock output. This pin can serve as clock output pin.
  PCINT4: Pin Change Interrupt 4. This pin can serve as external interrupt source.
- **RXD/PCINT3 - Port B, Bit1**
  RXD: This pin can serve as RXD pin for the LIN interface.
  PCINT3: Pin Change Interrupt 3. This pin can serve as external interrupt source.
- **FH/PCINT2 - Port B, Bit0**
  FH: Force High. When the PBOE0 bit in the PBOV register is set, this pin is forced high.
  PCINT2: Pin Change Interrupt 2. This pin can serve as external interrupt source.

Table 21-6. Overriding Signals for Alternate Functions in PB7:PB4

Signal Name	PB7/MISO/ICP10/ INT0/ PCINT9	PB6/MOSI/PCINT8	PB5/SCK/PCINT7	PB4/$\overline{\text{SS}}$/PCINT6
PUOE	SPE × MASTER	SPE × $\overline{\text{MASTER}}$	SPE × $\overline{\text{MASTER}}$	SPE × $\overline{\text{MASTER}}$
PUOV	PORTB7 × $\overline{\text{PUD}}$	PORTB7 × $\overline{\text{PUD}}$	PORTB7 × $\overline{\text{PUD}}$	PORTB7 × $\overline{\text{PUD}}$
DDOE	SPE × MASTER	SPE × $\overline{\text{MASTER}}$	SPE × $\overline{\text{MASTER}}$	SPE × $\overline{\text{MASTER}}$
DDOV	0	0	0	0
PVOE	SPE × $\overline{\text{MASTER}}$	SPE × MASTER	SPE × MASTER	0
PVOV	SPI SLAVE	SPI MASTER		
PTOE	0	0	0	0
DIEOE	PCINT9 × PCIE | INT0 Enable	PCINT8 × PCIE	PCINT7 × PCIE	PCINT6 × PCIE
DIEOV	1	1	1	1
DI	INT0 ICP10 SPI MASTER PCINT9	SPI SLAVE PCINT8	SCK PCINT7	$\overline{\text{SS}}$ PCINT6
AIO	-	-	-	-

Table 21-7. Overriding Signals for Alternate Functions in PB3:PB0

Signal Name	PB3/TXD/PCINT5	PB2/CKOUT/ PCINT4	PB1/RXD/ PCINT3	PB0/FH/PCINT2
PUOE	LINTXEN	CKOE	LINRXEN	PBOE0
PUOV	LINTXD × PBOE3 × PORTB3	0	PORTB2 × PUD	0
DDOE	LINTXEN	CKOE	LINRXEN	PBOE0
DDOV	$\overline{\text{LINTXD}}$ × $\overline{\text{PBOE3}}$	CKOE	0	1
PVOE	LINTXEN	CKOE	0	PBOE0
PVOV	LINTXD × $\overline{\text{PBOE3}}$	CKOUT	0	1
PTOE	0	0	0	0
DIEOE	PCINT5 × PCIE	(PCINT4 × PCIE) | CKOE	PCINT3	PCINT2 × PCIE
DIEOV	1	PCINT4 × PCIE | $\overline{\text{CKOE}}$	1	1
DI	T1 PCINT5	LINRXD PCINT4	PCINT3	T0 PCINT2
AIO	-	-	-	-

23. Timer/Counter(T/C0, T/C1)

23.1 Features

- Clear timer on compare match (auto reload)
- Input capture unit
- Four independent interrupt sources (TOVn, OCFnA, OCFnB, ICFn)
- 8-bit mode with two independent output compare units
- 16-bit mode with one independent output compare unit

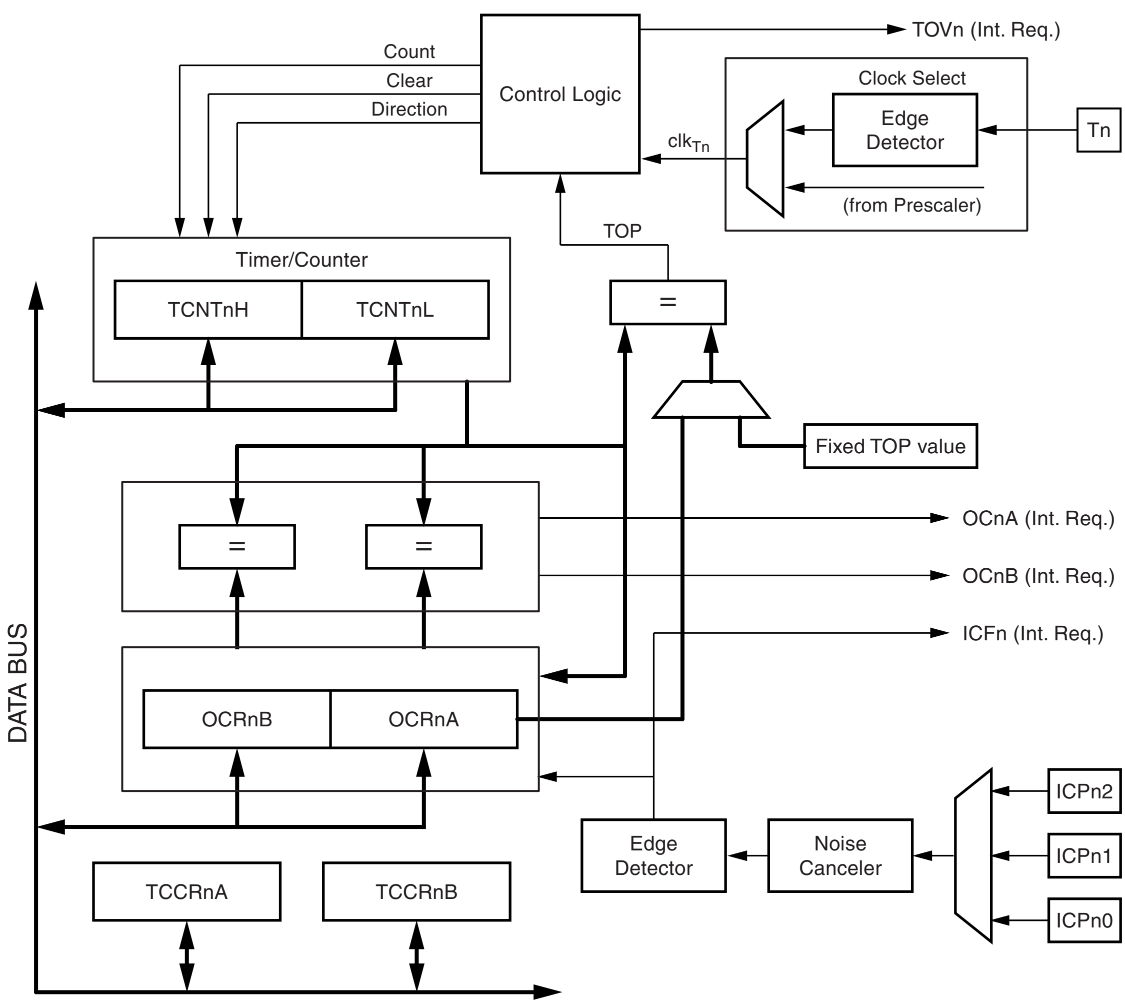
23.2 Overview

Timer/Counter n is a general purpose 8-/16-bit Timer/Counter module, with two/one Output Compare units and Input Capture feature.

The Atmel® AVR MCU has two Timer/Counters, Timer/Counter0 and Timer/Counter1. The functionality for both Timer/Counters is described below. Timer/Counter0 and Timer/Counter1 have different Timer/Counter registers, as shown in Section 32. "Register Summary" on page 203.

The Timer/Counter general operation is described in 8-/16-bit mode. A simplified block diagram of the 8-/16-bit Timer/Counter is shown in Figure 23-1. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the Section 23.10 "Register Description" on page 104.

Figure 23-1. 8-/16-bit Timer/Counter Block Diagram



- **Bit 5 – ICNCn: Input Capture Noise Canceler**

  Setting this bit activates the Input Capture Noise Canceler. When the noise canceler is activated, the input from the Input Capture Source is filtered. The filter function requires four successive equal valued samples of the Input Capture Source for changing its output. The Input Capture is therefore delayed by four System Clock cycles when the noise canceler is enabled.

- **Bit 4 – ICESn: Input Capture Edge Select**

  This bit selects which edge on the Input Capture Source that is used to trigger a capture event. When the ICESn bit is written to zero, a falling (negative) edge is used as trigger, and when the ICESn bit is written to one, a rising (positive) edge will trigger the capture. When a capture is triggered according to the ICESn setting, the counter value is copied into the Input Capture Register. The event will also set the Input Capture Flag (ICFn), and this can be used to cause an Input Capture Interrupt, if this interrupt is enabled.

- **Bits 3 – Reserved**

  This bit is reserved in the Atmel® AVR MCU and should always be written to zero.

- **Bits 2:1 – Reserved**

  These bits are reserved bits in the AVR MCU and will always read as zero.

- **Bit 0 – WGMn0: Waveform Generation Mode**

  This bit controls the counting sequence of the counter, the source for maximum (TOP) counter value, see Figure 23-6 on page 100. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter) and Clear Timer on Compare Match (CTC) mode (see Section 23.8 “Timer/Counter Timing Diagrams” on page 100).

23.10.2 TCCRnC – Timer/Counter n Control Register C

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	ICn1	ICn0	TCCRnC
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:2 – Reserved**

  These bits are reserved bits in the Atmel® AVR MCU and will always read as zero.

- **Bit 1:0 – ICS[1:0]: Input Capture Select 1:0**

  These bits control which Input Capture source that should trigger the Timer/Counter Input Capture functionality. To also trigger the Timer/Counter n Input Capture interrupt, the TICIEn bit in the Timer Interrupt Mask Register TIMSK) must be set.

  See Table 23-3 on page 99 and Table 23-4 on page 99 for Input Capture sources.

23.10.3 TCNTnL – Timer/Counter n Register Low Byte

Bit	7	6	5	4	3	2	1	0	
	TCNTnL[7:0]								TCNTnL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Timer/Counter Register TCNTnL gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNTnL Register blocks (disables) the Compare Match on the following timer clock. Modifying the counter (TCNTnL) while the counter is running, introduces a risk of missing a Compare Match between TCNTnL and the OCRnx Registers. In 16-bit mode the TCNTnL register contains the lower part of the 16-bit Timer/Counter n Register.

25.5.3 Data Transport

Two types of data may be transported in a frame; signals or diagnostic messages.

- Signals
  Signals are scalar values or byte arrays that are packed into the data field of a frame. A signal is always present at the same position in the data field for all frames with the same identifier.
- Diagnostic messages
  Diagnostic messages are transported in frames with two reserved identifiers. The interpretation of the data field depends on the data field itself as well as the state of the communicating nodes.

25.5.4 Schedule Table

The master task (in the master node) transmits frame headers based on a schedule table. The schedule table specifies the identifiers for each header and the interval between the start of a frame and the start of the following frame. The master application may use different schedule tables and select among them.

25.5.5 Compatibility with LIN 1.3

LIN 2.1 is a super-set of LIN 1.3.

A LIN 2.1 master node can handle clusters consisting of both LIN 1.3 slaves and/or LIN 2.1 slaves. The master will then avoid requesting the new LIN 2.1 features from a LIN 1.3 slave:

- Enhanced checksum,
- Re-configuration and diagnostics,
- Automatic baud rate detection,
- "Response error" status monitoring.

LIN 2.1 slave nodes can not operate with a LIN 1.3 master node (e.g., the LIN1.3 master does not support the enhanced checksum).

The LIN 2.1 physical layer is backwards compatible with the LIN1.3 physical layer. But not the other way around. The LIN 2.1 physical layer sets greater requirements, i.e. a master node using the LIN 2.1 physical layer can operate in a LIN 1.3 cluster.

25.6 LIN / UART Controller

The LIN/UART controller is divided in three main functions:

- Tx LIN Header function,
- Rx LIN Header function,
- LIN Response function.

These functions mainly use two services:

- Rx service,
- Tx service.

Because these two services are basically UART services, the controller is also able to switch into an UART function.

25.6.1 LIN Overview

The LIN/UART controller is designed to match as closely as possible to the LIN software application structure. The LIN software application is developed as independent tasks, several slave tasks and one master task (c.f. Section 25.5.4 on page 118). The Atmel® AVR MCU conforms to this perspective. The only link between the master task and the slave task will be at the cross-over point where the interrupt routine is called once a new identifier is available. Thus, in a master node, housing both master and slave task, the Tx LIN Header function will alert the slave task of an identifier presence. In the same way, in a slave node, the Rx LIN Header function will alert the slave task of an identifier presence.

When the slave task is warned of an identifier presence, it has first to analyze it to know what to do with the response. Hardware flags identify the presence of one of the specific identifiers from 60 (0x3C) up to 63 (0x3F).

25.6.7.2Rx Service

Once this service is enabled, the user is warned of an in-coming character by the LRXOK flag of LINSIR register. Reading LINDAT register automatically clears the flag and makes free the second stage of the buffer. If the user considers that the in-coming character is irrelevant without reading it, he directly can clear the flag (see specific flag management described in Section 25.8.2 on page 133).

The intrinsic structure of the Rx service offers a 2-byte buffer. The fist one is used for serial to parallel conversion, the second one receives the result of the conversion. This second buffer byte is reached reading LINDAT register. If the 2-byte buffer is full, a new in-coming character will overwrite the second one already recorded. An OVRERR error in LINERR register will then accompany this character when read.

A FERR error in LINERR register will be set in case of framing error.

25.6.7.3Tx Service

If this service is enabled, the user sends a character by writing in LINDAT register. Automatically the LTXOK flag of LINSIR register is cleared. It will rise at the end of the serial transmission. If no new character has to be sent, LTXOK flag can be cleared separately (see specific flag management described in Section 25.8.2 on page 133).

There is no transmit buffering.

No error is detected by this service.

25.7 LIN / UART Description

25.7.1 Reset

The AVR core reset logic signal also resets the LIN/UART controller. Another form of reset exists, a software reset controlled by LSWRES bit in LINCR register. This self-reset bit performs a partial reset as shown in Table 25-2.

Table 25-2. Reset of LIN/UART Registers

Register	Name	Reset Value	LSWRES Value	Comment
LIN Control Reg.	LINCR	$0000\ 0000_b$	$0000\ 0000_b$	x=unknown u=unchanged
LIN Status and Interrupt Reg.	LINSIR	$0000\ 0000_b$	$0000\ 0000_b$	
LIN Enable Interrupt Reg.	LINENIR	$0000\ 0000_b$	$xxxx\ 0000_b$	
LIN Error Reg.	LINERR	$0000\ 0000_b$	$0000\ 0000_b$	
LIN Bit Timing Reg.	LINBTR	$0010\ 0000_b$	$0010\ 0000_b$	
LIN Baud Rate Reg. Low	LINBRRL	$0000\ 0000_b$	$uuuu\ uuuu_b$	
LIN Baud Rate Reg. High	LINBRRH	$0000\ 0000_b$	$xxxx\ uuuu_b$	
LIN Data Length Reg.	LINDLR	$0000\ 0000_b$	$0000\ 0000_b$	
LIN Identifier Reg.	LINIDR	$1000\ 0000_b$	$1000\ 0000_b$	
LIN Data Buffer Selection	LINSEL	$0000\ 0000_b$	$xxxx\ 0000_b$	
LIN Data	LINDAT	$0000\ 0000_b$	$0000\ 0000_b$	

25.7.2 LIN Protocol Selection

LIN13 bit in LINCR register is used to select the LIN protocol:

- LIN13 = 0 (default): LIN 2.1 protocol,
- LIN13 = 1: LIN 1.3 protocol.

The controller checks the LIN13 bit in computing the checksum (enhanced checksum in LIN2.1 / classic checksum in LIN 1.3).

This bit is irrelevant for UART commands.

25.8.7 LINDLR – LIN Data Length Register

Bit	7	6	5	4	3	2	1	0	
(0xC7)	LTXDL3	LTXDL2	LTXDL1	LTXDL0	LRXDL3	LRXDL2	LRXDL1	LRXDL0	LINDLR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:4 – LTXDL[3:0]: LIN Transmit Data Length**

  In LIN mode, this field gives the number of bytes to be transmitted (clamped to 8 Max).

  In UART mode this field is unused.

- **Bits 3:0 – LRXDL[3:0]: LIN Receive Data Length**

  In LIN mode, this field gives the number of bytes to be received (clamped to 8 Max).

  In UART mode this field is unused.

25.8.8 LINIDR – LIN Identifier Register

Bit	7	6	5	4	3	2	1	0	
(0xC8)	LP1	LP0	LID5 / LDL1	LID4 / LDL0	LID3	LID2	LID1	LID0	LINIDR
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:6 – LP[1:0]: Parity**

  In LIN mode:

  ```
  LP0 = LID4 ^ LID2 ^ LID1 ^ LID0
  LP1 = ! ( LID1 ^ LID3 ^ LID4 ^ LID5 )
  ```

  In UART mode this field is unused.

- **Bits 5:4 – LDL[1:0]: LIN 1.3 Data Length**

  In LIN 1.3 mode:

  - 00 = 2-byte response,
  - 01 = 2-byte response,
  - 10 = 4-byte response,
  - 11 = 8-byte response.

  In UART mode this field is unused.

- **Bits 3:0 – LID[3:0]: LIN 1.3 Identifier**

  In LIN 1.3 mode: 4-bit identifier.

  In UART mode this field is unused.

- **Bits 5:0 – LID[5:0]: LIN 2.1 Identifier**

  In LIN 2.1 mode: 6-bit identifier (no length transported).

  In UART mode this field is unused.

29.8.13 Atmel ATmega32HVE Boot Loader Parameters

In Table 29-5 through Table 29-7, the parameters used in the description of the Self-Programming are given.

Table 29-5. Boot Size Configuration(1)

BOOTSZ1	BOOTSZ0	Boot Size	Pages	Application Flash Section	Boot Loader Flash Section	End Application Section	Boot Reset Address (Start Boot Loader Section)
1	1	256 words	4	0x0000 - 0x3EFF	0x3F00 - 0x3FFF	0x3EFF	0x3F00
1	0	512 words	8	0x0000 - 0x3DFF	0x3E00 - 0x3FFF	0x3DFF	0x3E00
0	1	1024 words	16	0x0000 - 0x3BFF	0x3C00 - 0x3FFF	0x3BFF	0x3C00
0	0	2048 words	32	0x0000 - 0x37FF	0x3800 - 0x3FFF	0x37FF	0x3800

Note: 1. The different BOOTSZ Fuse configurations are shown in Figure 29-2

Table 29-6. Read-While-Write Limit(1)

Section	Pages	Address
Read-While-Write section (RWW)	224	0x0000 - 0x37FF
No Read-While-Write section (NRWW)	32	0x3800 - 0x3FFF

Note: 1. For details about these two section, see Section 29.4.2 “NRWW – No Read-While-Write Section” on page 168 and Section 29.4.1 “RWW – Read-While-Write Section” on page 168.

Table 29-7. Explanation of Different Variables Used in Figure 29-3 on page 170 and the Mapping to the Z-pointer(1)

Variable		Corresponding Z-value	Description
PCMSB	13		Most significant bit in the Program Counter. (The Program Counter is 14 bits PC[13:0])
PAGEMSB	5		Most significant bit which is used to address the words within one page (64 words in a page requires six bits PC [5:0]).
ZPCMSB		Z14	Bit in Z-register that is mapped to PCMSB. Because Z0 is not used, the ZPCMSB equals PCMSB + 1.
ZPAGEMSB		Z6	Bit in Z-register that is mapped to PCMSB. Because Z0 is not used, the ZPAGEMSB equals PAGEMSB + 1.
PCPAGE	PC[13:6]	Z13:Z7	Program Counter page address: Page select, for Page Erase and Page Write
PCWORD	PC[5:0]	Z6:Z1	Program Counter word address: Word select, for filling temporary buffer (must be zero during Page Write operation)

Note: 1. Z0: should be zero for all SPM commands, byte select for the LPM instruction.
See Section 29.7 “Addressing the Flash During Self-Programming” on page 170 for details about the use of Z-pointer during Self-Programming.

Table 30-11. Pin Name Mapping

Signal Name in High-voltage Serial Programming Mode	Pin Name	I/O	Function
SDO	PB5	O	Serial Data Output
SDI	PB6	I	Serial Data Input
SII	PB7	I	Serial Instruction Input
SCI	PB2	I	Serial Clock Input (min. $2/f_{ck}$ period)

Table 30-12. Pin Values Used to Enter Programming Mode

Pin Name	Symbol	Value
PB4	Prog_enable[0]	0
PB5	Prog_enable[1]	0
PB6	Prog_enable[2]	0
PB7	Prog_enable[3]	0

30.8 High-voltage Serial Programming Algorithm

To program and verify the Atmel® AVR MCU in the High-voltage Serial Programming mode, the following sequence is recommended (See instruction formats in Table 30-14):

30.8.1 Enter High-voltage Serial Programming Mode

The following algorithm puts the device in Serial (High-voltage) Programming mode:

1. Set Prog_enable pins listed in Table 30-12 on page 187 to "0000", RESET pin to 0V and V_{CC} to 0V.
2. Apply 3.0 - 3.5V between V_{CC} and GND. Ensure that V_{CC} reaches at least 1.8V within the next 20 µs.
3. Wait 20 - 60 µs, and apply V_{HRST} - 12.5V to RESET.
4. Keep the Prog_enable pins unchanged for at least t_{HVRST} after the High-voltage has been applied to ensure the Prog_enable Signature has been latched.
5. Release Prog_enable[1] pin to avoid drive contention on the Prog_enable[1]/SDO pin.
6. Wait at least 1.3 ms before giving any serial instructions on SDI/SII.
7. If the rise time of the V_{CC} is unable to fulfill the requirements listed above, the following alternative algorithm can be used.
8. Set Prog_enable pins listed in Table 30-12 on page 187 to "0000", RESET pin to 0V and V_{CC} to 0V.
9. Apply 3.0 - 3.5V between V_{CC} and GND.
10. Monitor V_{CC}, and as soon as V_{CC} reaches 0.9 - 1.1V, apply V_{HRST} - 12.5V to RESET.
11. Keep the Prog_enable pins unchanged for at least t_{HVRST} after the High-voltage has been applied to ensure the Prog_enable Signature has been latched.
12. Release Prog_enable[1] pin to avoid drive contention on the Prog_enable[1]/SDO pin.
13. Wait until V_{CC} actually reaches 3.0 - 3.5V.
14. Wait at least 1.3ms before giving any serial instructions on SDI/SII.

Table 30-13. High-voltage Reset Characteristics

Supply Voltage	RESET Pin High-voltage Threshold	Minimum High-voltage Period for Latching Prog_enable
V_{CC}	V_{HVRST}	t_{HVRST}
3.0V	11.5V	10 µs
3.5V	11.5V	10 µs

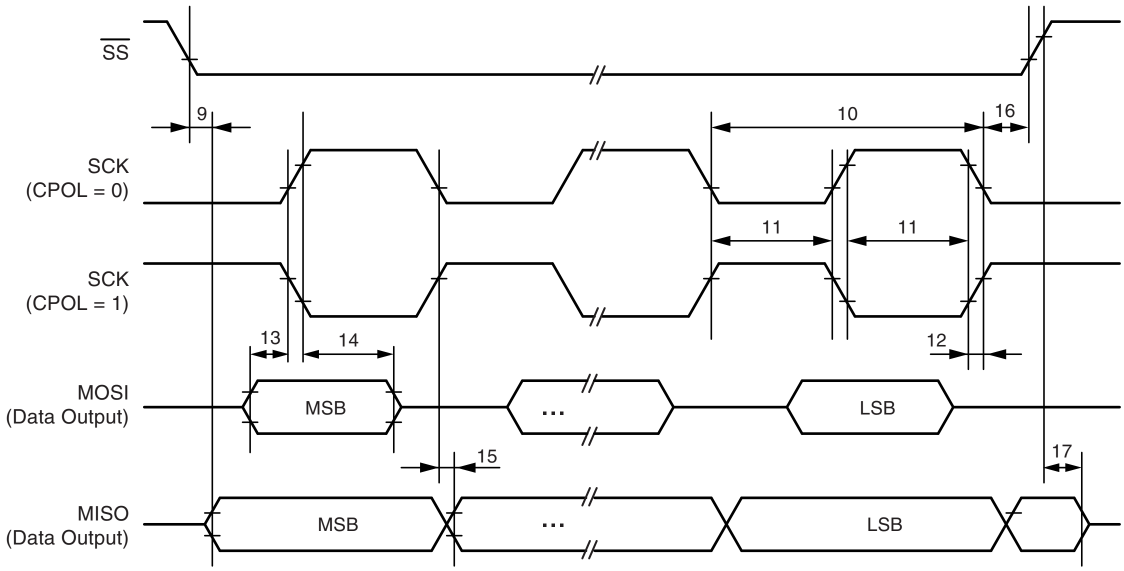
30.8.10 Power-off sequence

Exit Programming mode by powering the device down, or by bringing RESET pin to 0V.

Table 30-14. High-voltage Serial Programming Instruction Set for Atmel® AVR MCU

Instruction		Instruction Format				Operation Remarks
		Instr.1/5	Instr.2/6	Instr.3	Instr.4	
Chip Erase	SDI	0_1000_0000_00	0_0000_0000_00	0_0000_0000_00		
	SII	0_0100_1100_00	0_0110_0100_00	0_0110_1100_00		
	SDO	x_xxxx_xxxx_xx	x_xxxx_xxxx_xx	x_xxxx_xxxx_xx		
Load “Write Flash” Command	SDI	0_0001_0000_00				
	SII	0_0100_1100_00				
	SDO	x_xxxx_xxxx_xx				
Load Flash Page Buffer	SDI	0_**bbbb_bbbb**_00	0_**eeee_eeee**_00	0_**dddd_dddd**_00	0_0000_0000_00	
	SII	0_0000_1100_00	0_0010_1100_00	0_0011_1100_00	0_0111_1101_00	
	SDO	x_xxxx_xxxx_xx	x_xxxx_xxxx_xx	x_xxxx_xxxx_xx	x_xxxx_xxxx_xx	
	SDI	0_0000_0000_00				
	SII	0_0111_1100_00				
	SDO	x_xxxx_xxxx_xx				
Load Flash High Address and Program Page	SDI	0_**aaaa_aaaa**_00	0_0000_0000_00	0_0000_0000_00		
	SII	0_0001_1100_00	0_0110_0100_00	0_0110_1100_00		
	SDO	x_xxxx_xxxx_xx	x_xxxx_xxxx_xx	x_xxxx_xxxx_xx		
Load “Read Flash” Command	SDI	0_0000_0010_00				
	SII	0_0100_1100_00				
	SDO	x_xxxx_xxxx_xx				
Read Flash Low and High Bytes	SDI	0_**bbbb_bbbb**_00	0_**aaaa_aaaa**_00	0_0000_0000_00	0_0000_0000_00	
	SII	0_0000_1100_00	0_0001_1100_00	0_0110_1000_00	0_0110_1100_00	
	SDO	x_xxxx_xxxx_xx	x_xxxx_xxxx_xx	x_xxxx_xxxx_xx	**q_qqqq_qqq**x_xx	
	SDI	0_0000_0000_00	0_0000_0000_00			
	SII	0_0111_1000_00	0_0111_1100_00			
	SDO	x_xxxx_xxxx_xx	**p_pppp_ppp**x_xx			
Load “Write EEPROM” Command	SDI	0_0001_0001_00				
	SII	0_0100_1100_00				
	SDO	x_xxxx_xxxx_xx				
Load EEPROM Page Buffer	SDI	0_**bbbb_bbbb**_00	0_**eeee_eeee**_00	0_0000_0000_00	0_0000_0000_00	
	SII	0_0000_1100_00	0_0010_1100_00	0_0110_1101_00	0_0110_1100_00	
	SDO	x_xxxx_xxxx_xx	x_xxxx_xxxx_xx	x_xxxx_xxxx_xx	x_xxxx_xxxx_xx	
Program EEPROM Page	SDI	0_0000_0000_00	0_0000_0000_00			
	SII	0_0110_0100_00	0_0110_1100_00			
	SDO	x_xxxx_xxxx_xx	x_xxxx_xxxx_xx			
Write EEPROM Byte	SDI	0_**bbbb_bbbb**_00	0_**eeee_eeee**_00	0_0000_0000_00	0_0000_0000_00	
	SII	0_0000_1100_00	0_0010_1100_00	0_0110_1101_00	0_0110_0100_00	
	SDO	x_xxxx_xxxx_xx	x_xxxx_xxxx_xx	x_xxxx_xxxx_xx	x_xxxx_xxxx_xx	
	SDI	0_0000_0000_00				
	SII	0_0110_1100_00				
	SDO	x_xxxx_xxxx_xx				
Load “Read EEPROM” Command	SDI	0_0000_0011_00				
	SII	0_0100_1100_00				
	SDO	x_xxxx_xxxx_xx				

Figure 31-2. SPI Interface Timing Requirements (Slave Mode)



31.8 Programming Characteristics

31.8.1 Serial Programming

Figure 31-3. Serial Programming Timing

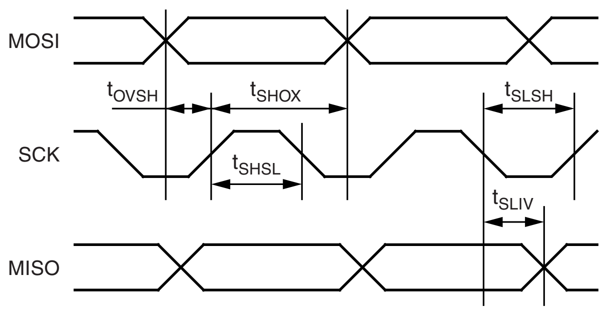
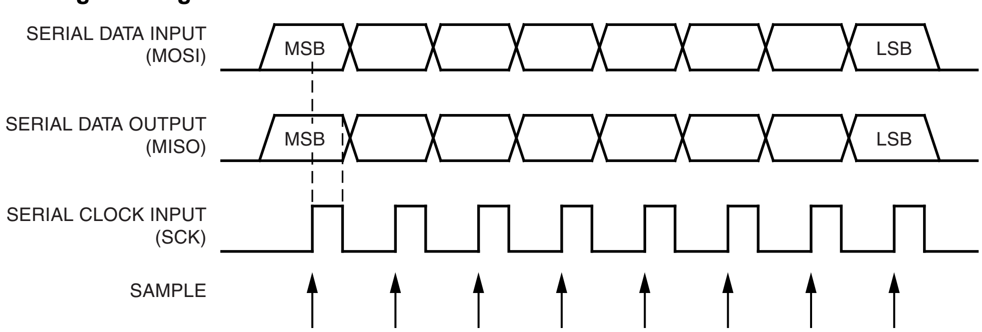


Figure 31-4. Serial Programming Waveforms



33. Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	if (Z = 1) then PC ←PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ←PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ←PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ←PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ←PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ←PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ←PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ←PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ←PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ←PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ←PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ←PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ←PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ←PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ←PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ←PC + k + 1	None	1/2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ←PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ←PC + k + 1	None	1/2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ←1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ←0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ←Rd(n), Rd(0) ←0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ←Rd(n+1), Rd(7) ←0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)←Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)←Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ←Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ←1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ←0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ←Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ←T	None	1
SEC		Set Carry	C ←1	C	1
CLC		Clear Carry	C ←0	C	1
SEN		Set Negative Flag	N ←1	N	1
CLN		Clear Negative Flag	N ←0	N	1
SEZ		Set Zero Flag	Z ←1	Z	1
CLZ		Clear Zero Flag	Z ←0	Z	1
SEI		Global Interrupt Enable	I ←1	I	1
CLI		Global Interrupt Disable	I ←0	I	1
SES		Set Signed Test Flag	S ←1	S	1
CLS		Clear Signed Test Flag	S ←0	S	1
SEV		Set Twos Complement Overflow.	V ←1	V	1
CLV		Clear Twos Complement Overflow	V ←0	V	1
SET		Set T in SREG	T ←1	T	1
CLT		Clear T in SREG	T ←0	T	1

33. Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
SEH		Set Half Carry Flag in SREG	H ←1	H	1
CLH		Clear Half Carry Flag in SREG	H ←0	H	1
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ←Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ←Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ←K	None	1
LD	Rd, X	Load Indirect	Rd ←(X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ←(X), X ←X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ←X - 1, Rd ←(X)	None	2
LD	Rd, Y	Load Indirect	Rd ←(Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ←(Y), Y ←Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ←Y - 1, Rd ←(Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ←(Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ←(Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ←(Z), Z ←Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ←Z - 1, Rd ←(Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ←(Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ←(k)	None	2
ST	X, Rr	Store Indirect	(X) ←Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ←Rr, X ←X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ←X - 1, (X) ←Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ←Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ←Rr, Y ←Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ←Y - 1, (Y) ←Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ←Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ←Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ←Rr, Z ←Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ←Z - 1, (Z) ←Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ←Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ←Rr	None	2
LPM		Load Program Memory	R0 ←(Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ←(Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ←(Z), Z ←Z+1	None	3
SPM		Store Program Memory	(Z) ←R1:R0	None	-
IN	Rd, P	In Port	Rd ←P	None	1
OUT	P, Rr	Out Port	P ←Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ←Rr	None	2
POP	Rd	Pop Register from Stack	Rd ←STACK	None	2
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A
MATHS EXTENSION INSTRUCTIONS					
ADD.L	Rd.I, Rs.I	32-bit Add	Rd.I ←Rd.I + Rs.I	1	1
ADC.L	Rd.I, Rs.I	32-bit Add with Carry	Rd.I ←Rd.I + Rs.I + C	1	1