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	Active
Core Processor	AVR
Core Size	8-Bit
Speed	16MHz
Connectivity	CANbus, LINbus, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, Temp Sensor, WDT
Number of I/O	-
Program Memory Size	32KB (16K x 16)
Program Memory Type	FLASH
EEPROM Size	1K x 8
RAM Size	2K x 8
Voltage - Supply (Vcc/Vdd)	2.7V ~ 5.5V
Data Converters	A/D 11x10b; D/A 1x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 125°C (TA)
Mounting Type	Surface Mount
Package / Case	32-TQFP
Supplier Device Package	32-TQFP (7x7)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atmega32c1-15az

- 10-bit ADC
  - Up to 11 single ended channels and 3 fully differential ADC channel pairs
  - Programmable gain (5x, 10x, 20x, 40x) on differential channels
  - Internal reference voltage
  - Direct power supply voltage measurement
- 10-bit DAC for variable voltage reference (comparators, ADC)
- Four analog comparators with variable threshold detection
- 100µA ±6% current source (LIN node identification)
- Interrupt and wake-up on pin change
- Programmable watchdog timer with separate on-chip oscillator
- On-chip temperature sensor

- Special microcontroller features
  - Low power idle, noise reduction, and power down modes
  - Power on reset and programmable brown out detection
  - In-system programmable via SPI port
  - High precision crystal oscillator for CAN operations (16MHz)
    - Internal calibrated RC oscillator (8MHz)
    - On-chip PLL for fast PWM (32MHz, 64MHz) and CPU (16MHz) (only Atmel® ATmega16/32/64M1)
  - Operating voltage:
    - 2.7V - 5.5V
  - Extended operating temperature:
    - –40°C to +125°C
  - Core speed grade:
    - 0 - 8MHz at 2.7 - 4.5V
    - 0 - 16MHz at 4.5 - 5.5V

Note: 1. See certification on Atmel web site and note on Section 16.4.3 “Baud Rate” on page 148.

Table 1. ATmega32/64/M1/C1 Product Line-up

Part Number	ATmega32C1	ATmega64C1	ATmega16M1	ATmega32M1	ATmega64M1
Flash size	32Kbyte	64Kbyte	16Kbyte	32Kbyte	64Kbyte
RAM size	2048 bytes	4096 bytes	1024 bytes	2048 bytes	4096 bytes
EEPROM size	1024 bytes	2048 bytes	512 bytes	1024 bytes	2048 bytes
8-bit timer	Yes				
16-bit timer	Yes				
PSC	No		Yes		
PWM outputs	4	4	10	10	10
Fault inputs (PSC)	0	0	3	3	3
PLL	No		Yes		
10-bit ADC channels	11 single 3 differential				
10-bit DAC	Yes				
analog comparators	4				
Current source	Yes				
CAN	Yes				
LIN/UART	Yes				
On-chip temp. sensor	Yes				
SPI interface	Yes				

Six of the 32 registers can be used as three 16-bit indirect address register pointers for data space addressing – enabling efficient address calculations. One of the these address pointers can also be used as an address pointer for look up tables in Flash program memory. These added function registers are the 16-bit X-, Y-, and Z-register, described later in this section.

The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the Status Register is updated to reflect information about the result of the operation.

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction.

Program flash memory space is divided in two sections, the boot program section and the application program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM (store program memory) instruction that writes into the application flash memory section must reside in the boot program section.

during interrupts and subroutine calls, the return address program counter (PC) is stored on the stack. The stack is effectively allocated in the general data SRAM, and consequently the stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the reset routine (before subroutines or interrupts are executed). The stack pointer (SP) is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture.

The memory spaces in the AVR® architecture are all linear and regular memory maps.

A flexible interrupt module has its control registers in the I/O space with an additional global interrupt enable bit in the status register. All interrupts have a separate interrupt vector in the interrupt vector table. The interrupts have priority in accordance with their interrupt vector position. The lower the interrupt vector address, the higher is the priority.

The I/O memory space contains 64 addresses for CPU peripheral functions as control registers, SPI, and other I/O functions. The I/O memory can be accessed directly, or as the data space locations following those of the register file, 0x20 - 0x5F. In addition, the Atmel ATmega16/32/64/M1/C1 has extended I/O space from 0x60 - 0xFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

3.3 ALU – Arithmetic Logic Unit

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format. See the “Instruction Set” section for a detailed description.

3.4 Status Register

The status register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the status register is updated after all ALU operations, as specified in the Instruction Set Reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

The status register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

The AVR Status Register – SREG – is defined as:

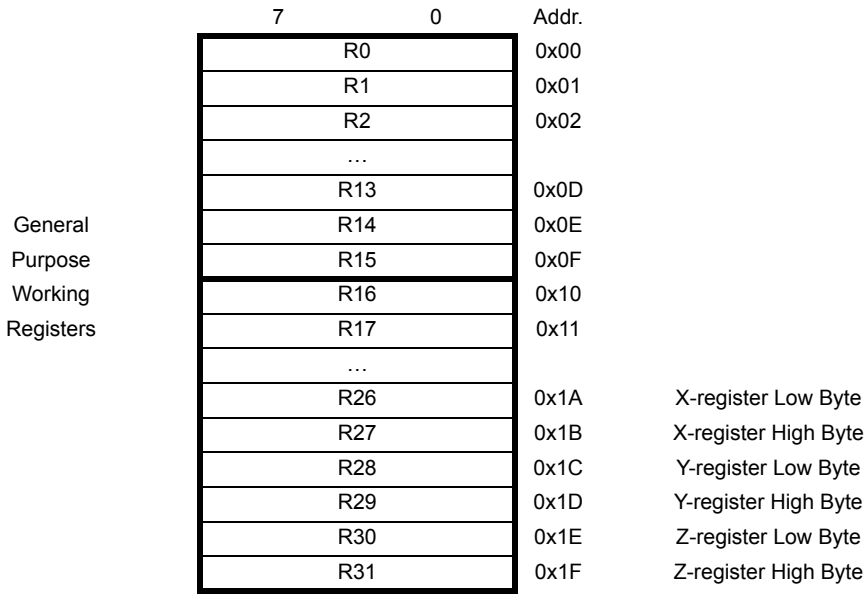
Bit	7	6	5	4	3	2	1	0	
	I	**T**	**H**	**S**	**V**	**N**	**Z**	**C**	**SREG**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – I: Global Interrupt Enable**

The global interrupt enable bit must be set to enabled the interrupts. The individual interrupt enable control is then performed in separate control registers. If the global interrupt enable register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

Figure 3-2 shows the structure of the 32 general purpose working registers in the CPU.

Figure 3-2. AVR CPU General Purpose Working Registers



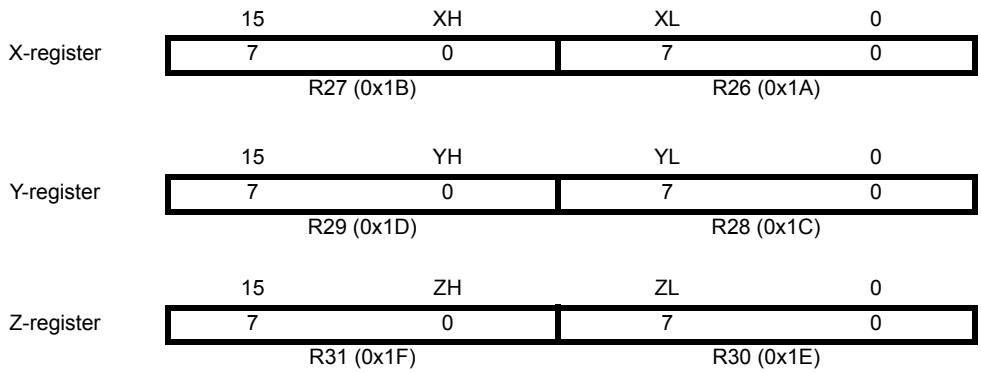
Most of the instructions operating on the register file have direct access to all registers, and most of them are single cycle instructions.

As shown in Figure 3-2, 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.

3.5.1 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 3-3.

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

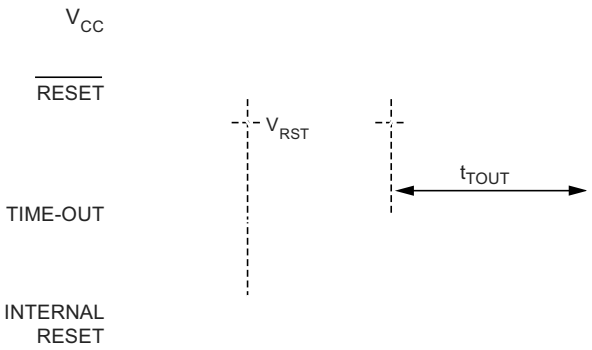


In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the instruction set reference for details).

7.2.2 External Reset

An external reset is generated by a low level on the $\overline{\text{RESET}}$ pin. Reset pulses longer than the minimum pulse width (see Table 7-1) will generate a reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a reset. When the applied signal reaches the Reset Threshold Voltage – V_{RST} – on its positive edge, the delay counter starts the MCU after the Time-out period – t_{TOUT} – has expired.

Figure 7-4. External Reset during Operation



7.2.3 Brown-out Detection

ATmega16/32/64/M1/C1 has an on-chip brown-out detection (BOD) circuit for monitoring the V_{CC} level during operation by comparing it to a fixed trigger level. The trigger level for the BOD can be selected by the BODLEVEL Fuses. The trigger level has a hysteresis to ensure spike free brown-out detection. The hysteresis on the detection level should be interpreted as $V_{BOT+} = V_{BOT} + V_{HYST}/2$ and $V_{BOT-} = V_{BOT} – V_{HYST}/2$.

Table 7-2. BODLEVEL Fuse Coding(1)(2)

BODLEVEL 2..0 Fuses	Typ V_{BOT}	Unit
111	Disabled	
110	4.5	V
011	4.4	V
100	4.3	V
010	4.2	V
001	2.8	V
101	2.7	V
000	2.6	V

Notes: 1. V_{BOT} may be below nominal minimum operating voltage for some devices. For devices where this is the case, the device is tested down to $V_{CC} = V_{BOT}$ during the production test. This guarantees that a brown-out reset will occur before V_{CC} drops to a voltage where correct operation of the microcontroller is no longer guaranteed. The test is performed using BODLEVEL = 010 for low operating voltage and BODLEVEL = 101 for high operating voltage.

2. Values are guidelines only.

Table 7-3. Brown-out Characteristics(1)

Parameter	Symbol	Min.	Typ.	Max.	Unit
Brown-out Detector Hysteresis	V_{HYST}		80		mV
Min Pulse Width on Brown-out Reset	t_{BOD}		2		µs

Note: 1. Values are guidelines only.

The following code example shows one assembly and one C function for turning off the Watchdog Timer. The example assumes that interrupts are controlled (e.g. by disabling interrupts globally) so that no interrupts will occur during the execution of these functions.

Assembly Code Example[1]

```
WDT_off:
       ; Turn off global interrupt
       cli
       ; Reset Watchdog Timer
       wdr
       ; Clear WDRF in MCUSR
       in    r16, MCUSR
       andi r16, (0xff & (0<<WDRF))
       out   MCUSR, r16
       ; Write logical one to WDCE and WDE
       ; Keep old prescaler setting to prevent unintentional time-out
       lds r16, WDTCSR
       ori   r16, (1<<WDCE) | (1<<WDE)
       sts WDTCSR, r16
       ; Turn off WDT
       ldi   r16, (0<<WDE)
       sts WDTCSR, r16
       ; Turn on global interrupt
       sei
       ret
```

C Code Example[1]

```
void WDT_off(void)
{
       __disable_interrupt();
       __watchdog_reset();
       /* Clear WDRF in MCUSR */
       MCUSR &= ~(1<<WDRF);
       /* Write logical one to WDCE and WDE */
       /* Keep old prescaler setting to prevent unintentional time-out */
       WDTCSR |= (1<<WDCE) | (1<<WDE);
       /* Turn off WDT */
       WDTCSR = 0x00;
       __enable_interrupt();
}
```

Notes: 1. The example code assumes that the part specific header file is included.

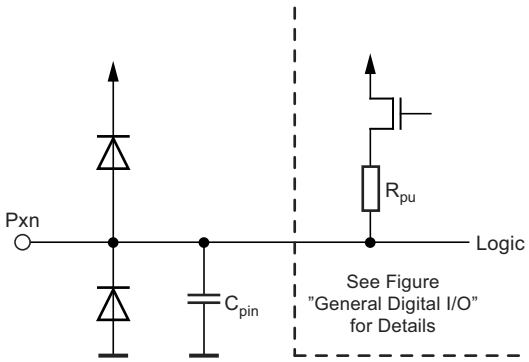
2. If the watchdog is accidentally enabled, for example by a runaway pointer or brown-out condition, the device will be reset and the watchdog timer will stay enabled. If the code is not set up to handle the watchdog, this might lead to an eternal loop of time-out resets. To avoid this situation, the application software should always clear the watchdog system reset flag (WDRF) and the WDE control bit in the initialization routine, even if the watchdog is not in use.

9. I/O-Ports

9.1 Introduction

All AVR® ports have true read-modify-write functionality when used as general digital I/O ports. This means that the direction of one port pin can be changed without unintentionally changing the direction of any other pin with the SBI and CBI instructions. The same applies when changing drive value (if configured as output) or enabling/disabling of pull-up resistors (if configured as input). Each output buffer has symmetrical drive characteristics with both high sink and source capability. All port pins have individually selectable pull-up resistors with a supply-voltage invariant resistance. All I/O pins have protection diodes to both V_{CC} and ground as indicated in Figure 9-1. Refer to Section 26. “Electrical Characteristics” on page 273 for a complete list of parameters.

Figure 9-1. I/O Pin Equivalent Schematic



All registers and bit references in this section are written in general form. A lower case “x” represents the numbering letter for the port, and a lower case “n” represents the bit number. However, when using the register or bit defines in a program, the precise form must be used. For example, PORTB3 for bit no. 3 in Port B, here documented generally as PORTxn. The physical I/O registers and bit locations are listed in “Register Description for I/O-Ports”.

Three I/O memory address locations are allocated for each port, one each for the data register – PORTx, data direction register – DDRx, and the port input pins – PINx. The port input pins I/O location is read only, while the data register and the data direction register are read/write. However, writing a logic one to a bit in the PINx register, will result in a toggle in the corresponding bit in the data register. In addition, the pull-up disable – PUD bit in MCUCR disables the pull-up function for all pins in all ports when set.

Using the I/O port as general digital I/O is described in “Ports as General Digital I/O”. Most port pins are multiplexed with alternate functions for the peripheral features on the device. How each alternate function interferes with the port pin is described in Section 9.3 “Alternate Port Functions” on page 55. Refer to the individual module sections for a full description of the alternate functions.

Note that enabling the alternate function of some of the port pins does not affect the use of the other pins in the port as general digital I/O.

13.10.9 Timer/Counter1 Interrupt Flag Register – TIFR1

Bit	7	6	5	4	3	2	1	0	
	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7, 6 – Res: Reserved Bits**

These bits are unused bits in the ATmega16/32/64/M1/C1, and will always read as zero.

- **Bit 5 – ICF1: Timer/Counter1, Input Capture Flag**

This flag is set when a capture event occurs on the ICP1 pin. When the input capture register (ICR1) is set by the WGMn3:0 to be used as the TOP value, the ICF1 Flag is set when the counter reaches the TOP value.

ICF1 is automatically cleared when the input capture interrupt vector is executed. Alternatively, ICF1 can be cleared by writing a logic one to its bit location.

- **Bit 4, 3 – Res: Reserved Bits**

These bits are unused bits in the ATmega16/32/64/M1/C1, and will always read as zero.

- **Bit 2 – OCF1B: Timer/Counter1, Output Compare B Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the output compare register B (OCR1B).

Note that a forced output compare (FOC1B) strobe will not set the OCF1B flag.

OCF1B is automatically cleared when the output compare match B interrupt vector is executed. Alternatively, OCF1B can be cleared by writing a logic one to its bit location.

- **Bit 1 – OCF1A: Timer/Counter1, Output Compare A Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the output compare register A (OCR1A).

Note that a forced output compare (FOC1A) strobe will not set the OCF1A flag.

OCF1A is automatically cleared when the output compare match A interrupt vector is executed. Alternatively, OCF1A can be cleared by writing a logic one to its bit location.

- **Bit 0 – TOV1: Timer/Counter1, Overflow Flag**

The setting of this flag is dependent of the WGMn3:0 bits setting. In normal and CTC modes, the TOV1 flag is set when the timer overflows. Refer to Table 13-4 on page 111 for the TOV1 flag behavior when using another WGMn3:0 bit setting.

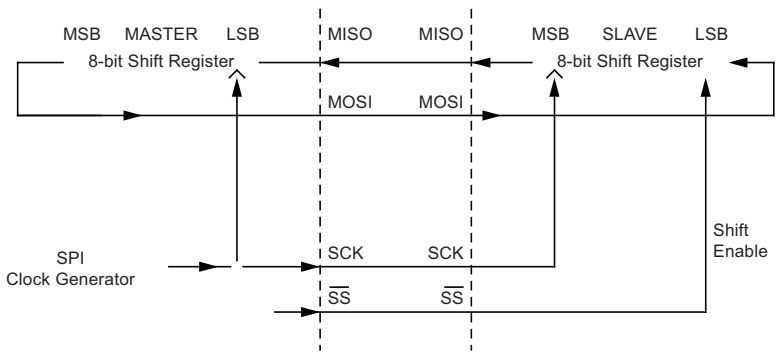
TOV1 is automatically cleared when the Timer/Counter1 overflow interrupt vector is executed. Alternatively, TOV1 can be cleared by writing a logic one to its bit location.

The interconnection between master and slave CPUs with SPI is shown in Figure 15-2. The system consists of two shift registers, and a master clock generator. The SPI master initiates the communication cycle when pulling low the slave select $\overline{SS}$ pin of the desired slave. Master and slave prepare the data to be sent in their respective shift registers, and the master generates the required clock pulses on the SCK line to interchange data. Data is always shifted from master to slave on the master out – slave in, MOSI, line, and from slave to master on the master in – slave out, MISO, line. After each data packet, the master will synchronize the slave by pulling high the slave select, $\overline{SS}$, line.

When configured as a master, the SPI interface has no automatic control of the $\overline{SS}$ line. This must be handled by user software before communication can start. When this is done, writing a byte to the SPI data register starts the SPI clock generator, and the hardware shifts the eight bits into the slave. After shifting one byte, the SPI clock generator stops, setting the end of transmission flag (SPIF). If the SPI interrupt enable bit (SPIE) in the SPCR register is set, an interrupt is requested. The master may continue to shift the next byte by writing it into SPDR, or signal the end of packet by pulling high the slave select, $\overline{SS}$ line. The last incoming byte will be kept in the buffer register for later use.

When configured as a slave, the SPI interface will remain sleeping with MISO tri-stated as long as the $\overline{SS}$ pin is driven high. In this state, software may update the contents of the SPI data register, SPDR, but the data will not be shifted out by incoming clock pulses on the SCK pin until the $\overline{SS}$ pin is driven low. As one byte has been completely shifted, the end of transmission flag, SPIF is set. If the SPI interrupt enable bit, SPIE, in the SPCR register is set, an interrupt is requested. The slave may continue to place new data to be sent into SPDR before reading the incoming data. The last incoming byte will be kept in the buffer register for later use.

Figure 15-2. SPI Master-slave Interconnection



The system is single buffered in the transmit direction and double buffered in the receive direction. This means that bytes to be transmitted cannot be written to the SPI data register before the entire shift cycle is completed. When receiving data, however, a received character must be read from the SPI data register before the next character has been completely shifted in. Otherwise, the first byte is lost.

In SPI slave mode, the control logic will sample the incoming signal of the SCK pin. To ensure correct sampling of the clock signal, the frequency of the SPI clock should never exceed $f_{clkio}/4$.

When the SPI is enabled, the data direction of the MOSI, MISO, SCK, and $\overline{SS}$ pins is overridden according to Table 15-1. For more details on automatic port overrides, refer to Section 9.3 “Alternate Port Functions” on page 55.

Table 15-1. SPI Pin Overrides(1)

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User defined	Input
MISO	Input	User defined
SCK	User defined	Input
$\overline{SS}$	User defined	Input

Note: 1. See Section 9.3.2 “Alternate Functions of Port B” on page 58 for a detailed description of how to define the direction of the user defined SPI pins.

16.5.2.2 Tx Data and Remote Frame

1. Several fields must be initialized before sending:
    - Identifier tag (IDT)
    - Identifier extension (IDE)
    - Remote transmission request (RTRTAG)
    - Data length code (DLC)
    - Reserved bit(s) tag (RBnTAG)
    - Data bytes of message (MSG)
2. The MOb is ready to send a data or a remote frame when the MOb configuration is set (CONMOB).
3. Then, the CAN channel scans all the MObs in Tx configuration, finds the MOb having the highest priority and tries to send it.
4. When the transmission is completed the TXOK flag is set (interrupt).
5. All the parameters and data are available in the MOb until a new initialization.

16.5.2.3 Rx Data and Remote Frame

1. Several fields must be initialized before receiving:
    - Identifier tag (IDT)
    - Identifier mask (IDMSK)
    - Identifier extension (IDE)
    - Identifier extension mask (IDEMSK)
    - Remote transmission request (RTRTAG)
    - Remote transmission request mask (RTRMSK)
    - Data length code (DLC)
    - Reserved bit(s) tag (RBnTAG)
2. The MOb is ready to receive a data or a remote frame when the MOb configuration is set (CONMOB).
3. When a frame identifier is received on CAN network, the CAN channel scans all the MObs in receive mode, tries to find the MOb having the highest priority which is matching.
4. On a hit, the IDT, the IDE and the DLC of the matched MOb are updated from the incoming (frame) values.
5. Once the reception is completed, the data bytes of the received message are stored (not for remote frame) in the data buffer of the matched MOb and the RXOK flag is set (interrupt).
6. All the parameters and data are available in the MOb until a new initialization.

16.5.2.4 Automatic Reply

A reply (data frame) to a remote frame can be automatically sent after reception of the expected remote frame.

1. Several fields must be initialized before receiving the remote frame:
    - Reply valid (RPLV) in a identical flow to the one described in Section 16.5.2.3 “Rx Data and Remote Frame” on page 150.
2. When a remote frame matches, automatically the RTRTAG and the reply valid bit (RPLV) are reset. No flag (or interrupt) is set at this time. Since the CAN data buffer has not been used by the incoming remote frame, the MOb is then ready to be in transmit mode without any more setting. The IDT, the IDE, the other tags and the DLC of the received remote frame are used for the reply.
3. When the transmission of the reply is completed the TXOK flag is set (interrupt).
4. All the parameters and data are available in the MOb until a new initialization.

16.8.2 Interrupt Behavior

When an interrupt occurs, an interrupt flag bit is set in the corresponding MOb-CANSTMOB register or in the general CANGIT register. If in the CANIE register, ENRX / ENTX / ENERR bit are set, then the corresponding MOb bit is set in the CANSITn register.

To acknowledge a MOb interrupt, the corresponding bits of CANSTMOB register (RXOK, TXOK,...) must be cleared by the software application. This operation needs a read-modify-write software routine.

To acknowledge a general interrupt, the corresponding bits of CANGIT register (BXOK, BOFFIT,...) must be cleared by the software application. This operation is made writing a logical one in these interrupt flags (writing a logical zero doesn't change the interrupt flag value).

OVRTIM interrupt flag is reset as the other interrupt sources of CANGIT register and is also reset entering in its dedicated interrupt handler.

When the CAN node is in transmission and detects a Form Error in its frame, a bit Error will also be raised. Consequently, two consecutive interrupts can occur, both due to the same error. When a MOb error occurs and is set in its own CANSTMOB register, no general error is set in CANGIT register.

17.4.6.3 Rx and TX Response Functions

These functions are initiated by the slave task of a LIN node. They must be used after sending an header (master task) or after receiving an header (considered as belonging to the slave task). When the TX response order is sent, the transmission begins. A Rx response order can be sent up to the reception of the last serial bit of the first byte (before the stop-bit).

In LIN 1.3, the header slot configures the LINDLR register. In LIN 2.1, the user must configure the LINDLR register, either LRXDL[3..0] for *Rx Response* either LTXDL[3..0] for *Tx Response*.

When the command starts, the controller checks the LIN13 bit of the LINCR register to apply the right rule for computing the checksum. Checksum calculation over the DATA bytes and the PROTECTED IDENTIFIER byte is called enhanced checksum and it is used for communication with LIN 2.1 slaves. Checksum calculation over the DATA bytes only is called classic checksum and it is used for communication with LIN 1.3 slaves. Note that identifiers 60 (0x3C) to 63 (0x3F) shall always use classic checksum.

At the end of this reception or transmission, the controller automatically returns to *Rx Header / LIN Abort* state (i.e. LCMD[1..0] = 00) after setting the appropriate flags.

If an LIN error occurs, the reception or the transmission is stopped, the appropriate flags are set and the LIN bus is left to recessive state.

During these functions, the controller is responsible for:

- The initialization of the checksum operator,
- The transmission or the reception of ‘n’ data with the update of the checksum calculation,
- The transmission or the checking of the CHECKSUM field,
- The checking of the Frame_Time_Out,
- The checking of the LIN communication integrity.

While the controller is sending or receiving a response, BREAK and SYNCH fields can be detected and the identifier of this new header will be recorded. Of course, specific errors on the previous response will be maintained with this identifier reception.

17.4.6.4 Handling Data of LIN response

A FIFO data buffer is used for data of the LIN response. After setting all parameters in the LINSEL register, repeated accesses to the LINDAT register perform data read or data write (c.f. Section 17.5.15 “Data Management” on page 189).

Note that LRXDL[3..0] and LTXDL[3..0] are not linked to the data access.

17.4.7 UART Commands

Setting the LCMD[2] bit in LINENR register enables UART commands.

Tx Byte and Rx Byte services are independent as shown in Table 17-1 on page 178.

- Byte transfer: the UART is selected but both Rx and Tx services are disabled,
- Rx Byte: only the Rx service is enable but Tx service is disabled,
- Tx Byte: only the Tx service is enable but Rx service is disabled,
- Full duplex: the UART is selected and both Rx and Tx services are enabled.

This combination of services is controlled by the LCMD[1..0] bits of LINENR register (c.f. Figure 17-5 on page 178).

17.4.7.1 Data Handling

The FIFO used for LIN communication is disabled during UART accesses. LRXDL[3..0] and LTXDL[3..0] values of LINDLR register are then irrelevant. LINDAT register is then used as data register and LINSEL register is not relevant.

17.4.7.2 Rx 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 17.6.2 “LIN Status and Interrupt Register - LINSIR” on page 192). 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.

- **Bit 3 - LSERR: Synchronization Error Flag**
  - 0 = No error,
  - 1 = Synchronization error. This bit is cleared when LERR bit in LINSIR is cleared.
- **Bit 2 - LPERR: Parity Error Flag**
  - 0 = No error,
  - 1 = Parity error. This bit is cleared when LERR bit in LINSIR is cleared.
- **Bit 1 - LCERR: Checksum Error Flag**
  - 0 = No error,
  - 1 = Checksum error. This bit is cleared when LERR bit in LINSIR is cleared.
- **Bit 0 - LBERR: Bit Error Flag**
  - 0 = no error,
  - 1 = Bit error. This bit is cleared when LERR bit in LINSIR is cleared.

17.6.5 LIN Bit Timing Register - LINBTR

Bit	7	6	5	4	3	2	1	0	
	LDISR	**-**	**LBT5**	**LBT4**	**LBT3**	**LBT2**	**LBT1**	**LBT0**	**LINBTR**
Read/Write	R/W	R	R/(W)	R/(W)	R/(W)	R/(W)	R/(W)	R/(W)	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 7 - LDISR: Disable Bit Timing Re synchronization**
  - 0 = Bit timing re-synchronization enabled (default),
  - 1 = Bit timing re-synchronization disabled.
- **Bits 5:0 - LBT[5:0]: LIN Bit Timing**
  Gives the number of samples of a bit.
  sample-time = (1 / $f_{clk_{i/o}}$) x (LDIV[11..0] + 1)
  Default value: LBT[6:0]=32 — Min. value: LBT[6:0]=8 — Max. value: LBT[6:0]=63

17.6.6 LIN Baud Rate Register - LINBRR

Bit	7	6	5	4	3	2	1	0	
	LDIV7	**LDIV6**	**LDIV5**	**LDIV4**	**LDIV3**	**LDIV2**	**LDIV1**	**LDIV0**	**LINBRRL**
	-	**-**	**-**	**-**	**LDIV11**	**LDIV10**	**LDIV9**	**LDIV8**	**LINBRRH**
Bit	15	14	13	12	11	10	9	8	
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 15:12 - Reserved Bits**
  These bits are reserved for future use. For compatibility with future devices, they must be written to zero when LINBRR is written.
- **Bits 11:0 - LDIV[11:0]: Scaling of $clk_{i/o}$ Frequency**
  The LDIV value is used to scale the entering $clk_{i/o}$ frequency to achieve appropriate LIN or UART baud rate.

20. Analog Comparator

The analog comparator compares the input values on the positive pin ACMPx and negative pin ACMPM or ACMPMx.

20.1 Features

- 4 analog comparators
- High-speed clocked comparators
- 4 reference levels
- Generation of configurable interrupts

20.2 Overview

The ATmega16/32/64/M1/C1 features 4 fast analog comparators.

Each comparator has a dedicated input on the positive input, and the negative input of each comparator can be configured as:

- a steady value among the 4 internal reference levels defined by the Vref selected thanks to the REFS1:0 bits in ADMUX register.
- a value generated from the internal DAC
- an external analog input ACMPMx.

When the voltage on the positive ACMPn pin is higher than the voltage selected by the ACnM multiplexer on the negative input, the analog comparator output, ACnO, is set.

The comparator is a clocked comparator. The comparators can run with a clock frequency of up to 16MHz (typical value) when the supply voltage is in the 4.5V-5.5V range and with a clock frequency of up to 8MHz (typical value) otherwise.

Each comparator can trigger a separate interrupt, exclusive to the analog comparator. In addition, the user can select Interrupt triggering on comparator output rise, fall or toggle.

The interrupt flags can also be used to synchronize ADC or DAC conversions.

Moreover, the comparator's output of the comparator 1 can be set to trigger the Timer/Counter1 Input Capture function.

A block diagram of the four comparators and their surrounding logic is shown in Figure 20-1 on page 226.

20.3 Use of ADC Amplifiers

Thanks to AMPCMP0 configuration bit, comparator 0 positive input can be connected to amplifier O output. In that case, the clock of comparator 0 is twice the amplifier 0 clock. See Section 18.11.1 “Amplifier 0 control and status register – AMP0CSR” on page 218.

Thanks to AMPCMP1 configuration bit, comparator 1 positive input can be connected to amplifier 1 output. In that case, the clock of comparator 1 is twice the amplifier 1 clock. See Section 18.11.2 “Amplifier 1 Control and Status Register – AMP1CSR” on page 219.

Thanks to AMPCMP2 configuration bit, comparator 2 positive input can be connected to amplifier 2 output. In that case, the clock of comparator 2 is twice the amplifier 2 clock. See Section 18.11.2 “Amplifier 1 Control and Status Register – AMP1CSR” on page 219.

20.4 Analog Comparator Register Description

Each analog comparator has its own control register.

A dedicated register has been designed to consign the outputs and the flags of the 4 analog comparators.

20.4.1 Analog Comparator 0 Control Register – AC0CON

Bit	7	6	5	4	3	2	1	0	
	AC0EN	**AC0IE**	**AC0IS1**	**AC0IS0**	**ACCKSEL**	**AC0M2**	**AC0M1**	**AC0M0**	**AC0CON**
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	

- **Bit 7– AC0EN: analog comparator 0 Enable Bit**

Set this bit to enable the analog comparator 0.
Clear this bit to disable the analog comparator 0.

- **Bit 6– AC0IE: analog comparator 0 Interrupt Enable bit**

Set this bit to enable the analog comparator 0 interrupt.
Clear this bit to disable the analog comparator 0 interrupt.

- **Bit 5, 4– AC0IS1, AC0IS0: analog comparator 0 Interrupt Select bit**

These 2 bits determine the sensitivity of the interrupt trigger.
The different setting are shown in Table 18-7.

Table 20-1. Interrupt Sensitivity Selection

AC0IS1	AC0IS0	Description
0	0	Comparator interrupt on output toggle
0	1	Reserved
1	0	Comparator interrupt on output falling edge
1	1	Comparator interrupt on output rising edge

- **Bit 3 – ACCKSEL: Analog Comparator Clock Select**

Set this bit to use the 16MHz PLL output as comparator clock. Clear this bit to use the CLK_{IO} as comparator clock.

- **Bit 2, 1, 0– AC0M2, AC0M1, AC0M0: Analog Comparator 0 Multiplexer Register**

These 3 bits determine the input of the negative input of the analog comparator.
The different setting are shown in Table 20-2 on page 228.

23.4 Software Break Points

debugWIRE supports program memory break points by the AVR® break instruction. Setting a break point in AVR Studio® will insert a BREAK instruction in the program memory. The instruction replaced by the BREAK instruction will be stored. When program execution is continued, the stored instruction will be executed before continuing from the program memory. A break can be inserted manually by putting the BREAK instruction in the program.

The flash must be re-programmed each time a break point is changed. This is automatically handled by AVR Studio through the debugWIRE interface. The use of break points will therefore reduce the flash data retention. Devices used for debugging purposes should not be shipped to end customers.

23.5 Limitations of debugWIRE

The debugWIRE communication pin (dW) is physically located on the same pin as external reset (RESET). An external reset source is therefore not supported when the debugWIRE is enabled.

The debugWIRE system accurately emulates all I/O functions when running at full speed, i.e., when the program in the CPU is running. When the CPU is stopped, care must be taken while accessing some of the I/O registers via the debugger (AVR Studio).

A programmed DWEN fuse enables some parts of the clock system to be running in all sleep modes. This will increase the power consumption while in sleep. Thus, the DWEN Fuse should be disabled when debugWire is not used.

23.6 debugWIRE Related Register in I/O Memory

The following section describes the registers used with the debugWire.

23.6.1 debugWire Data Register – DWDR

Bit	7	6	5	4	3	2	1	0	
	DWDR[7:0]								DWDR
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 DWDR register provides a communication channel from the running program in the MCU to the debugger. This register is only accessible by the debugWIRE and can therefore not be used as a general purpose register in the normal operations.

24. Boot Loader Support – Read-while-write Self-Programming ATmega16/32/64/M1/C1

In ATmega16/32/64/M1/C1, the boot loader support provides a real read-while-write self-programming mechanism for downloading and uploading program code by the MCU itself. This feature allows flexible application software updates controlled by the MCU using a flash-resident boot loader program. The boot loader program can use any available data interface and associated protocol to read code and write (program) that code into the Flash memory, or read the code from the program memory. The program code within the boot loader section has the capability to write into the entire flash, including the boot loader memory. The boot loader can thus even modify itself, and it can also erase itself from the code if the feature is not needed anymore. The size of the boot loader memory is configurable with fuses and the boot loader has two separate sets of boot lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

24.1 Boot Loader Features

- Read-while-write self-programming
- Flexible boot memory size
- High security (separate boot lock bits for a flexible protection)
- Separate fuse to select reset vector
- Optimized page(1) size
- Code efficient algorithm
- Efficient read-modify-write support

Note: 1. A page is a section in the flash consisting of several bytes (see Table 25-12 on page 260) used during programming. The page organization does not affect normal operation.

24.2 Application and Boot Loader Flash Sections

The flash memory is organized in two main sections, the application section and the boot loader section (see Figure 24-2 on page 243). The size of the different sections is configured by the BOOTSZ fuses as shown in Table 24-7 on page 251 and Figure 24-2 on page 243. These two sections can have different level of protection since they have different sets of lock bits.

24.2.1 Application Section

The application section is the section of the Flash that is used for storing the application code. The protection level for the application section can be selected by the application boot lock bits (boot lock bits 0), see Table 24-2 on page 244. The application section can never store any boot loader code since the SPM instruction is disabled when executed from the application section.

24.2.2 BLS – Boot Loader Section

While the application section is used for storing the application code, the The boot loader software must be located in the BLS since the SPM instruction can initiate a programming when executing from the BLS only. The SPM instruction can access the entire flash, including the BLS itself. The protection level for the boot loader section can be selected by the boot loader lock bits (boot lock bits 1), see Table 24-3 on page 244.

24.3 Read-while-write and no Read-while-write Flash Sections

Whether the CPU supports read-while-write or if the CPU is halted during a Boot Loader software update is dependent on which address that is being programmed. In addition to the two sections that are configurable by the BOOTSZ Fuses as described above, the flash is also divided into two fixed sections, the read-while-write (RWW) section and the no read-while-write (NRWW) section. The limit between the RWW- and NRWW sections is given in Table 24-8 on page 252 and Figure 24-2 on page 243. The main difference between the two sections is:

- When erasing or writing a page located inside the RWW section, the NRWW section can be read during the operation.
- When erasing or writing a page located inside the NRWW section, the CPU is halted during the entire operation.

G. Load address high byte

1. Set XA1, XA0 to “00”. This enables address loading.
2. Set BS1 to “1”. This selects high address.
3. Set DATA = Address high byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the address high byte.

H. Program page

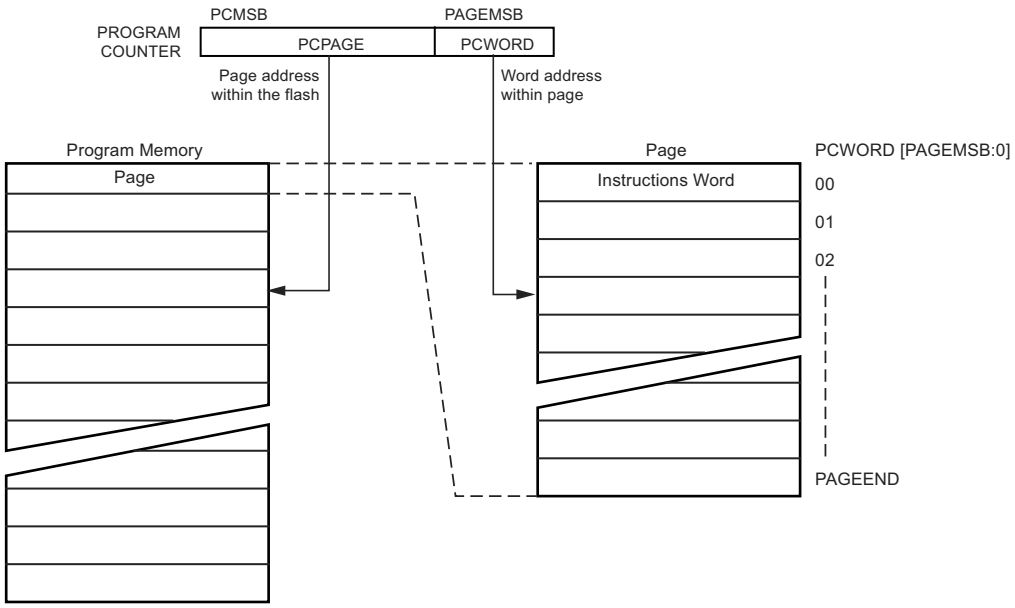
1. Give $\overline{WR}$ a negative pulse. This starts programming of the entire page of data. RDY/$\overline{BSY}$ goes low.
2. Wait until RDY/$\overline{BSY}$ goes high (See Figure 25-3 for signal waveforms).

I. Repeat B through H until the entire flash is programmed or until all data has been programmed.

J. End page programming

1. 1. Set XA1, XA0 to “10”. This enables command loading.
2. Set DATA to “0000 0000”. This is the command for no operation.
3. Give XTAL1 a positive pulse. This loads the command, and the internal write signals are reset.

Figure 25-2. Addressing the Flash which is Organized in Pages(1)



Note: 1. PCPAGE and PCWORD are listed in Table 25-12 on page 260.

26. Electrical Characteristics

All DC/AC characteristics contained in this datasheet are based on simulations and characterization of similar devices in the same process and design methods. These values are preliminary representing design targets, and will be updated after characterization of actual automotive silicon data.

26.1 Absolute Maximum Ratings

Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Parameters	Min.	Typ.	Max.	Unit
Operating temperature	–40		+125	°C
Storage temperature	–65		+150	°C
Voltage on any pin except $\overline{\text{RESET}}$ with respect to ground	–0.5		V_{CC} + 0.5	V
Voltage on $\overline{\text{RESET}}$ with respect to ground	–0.5		+13	V
Maximum operating voltage			6	V
DC current per I/O pin		40		mA
DC current V_{CC} and GND pins		200		mA
Injection current at V_{CC} = 0V to 5V		±5(1)		mA

Note: 1. Maximum current per port = ±30mA

26.2 DC Characteristics

T_A = –40°C to +125°C, V_{CC} = 2.7V to 5.5V (unless otherwise noted)

Parameter	Condition	Symbol	Min.	Typ.	Max.	Unit
Input low voltage	Port B, C and D and XTAL1, XTAL2 pins as I/O	V_{IL}	–0.5		$0.2V_{CC}$(1)	V
Input high voltage	Port B, C and D and XTAL1, XTAL2 pins as I/O	V_{IH}	$0.6V_{CC}$(2)		V_{CC} + 0.5	V
Input low voltage	XTAL1 pin, external clock Selected	V_{IL1}	–0.5		$0.1V_{CC}$(1)	V
Input high voltage	XTAL1 pin, external clock selected	V_{IH1}	$0.8V_{CC}$(2)		V_{CC} + 0.5	V

Notes:
1. “Max” means the highest value where the pin is guaranteed to be read as low
2. “Min” means the lowest value where the pin is guaranteed to be read as high
3. Although each I/O port can sink more than the test conditions (10mA at V_{CC} = 5V, 6mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
   1] The sum of all IOL, for ports B0 - B1, C2 - C3, D4, E1 - E2 should not exceed 70mA.
   2] The sum of all IOL, for ports B6 - B7, C0 - C1, D0 -D3, E0 should not exceed 70mA.
   3] The sum of all IOL, for ports B2 - B5, C4 - C7, D5 - D7 should not exceed 70mA.
   If IOL exceeds the test condition, VOL may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition.
4. Although each I/O port can source more than the test conditions (10mA at V_{CC} = 5V, 8mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
   1] The sum of all IOH, for ports B0 - B1, C2 - C3, D4, E1 - E2 should not exceed 100mA.
   2] The sum of all IOH, for ports B6 - B7, C0 - C1, D0 -D3, E0 should not exceed 100mA.
   3] The sum of all IOH, for ports B2 - B5, C4 - C7, D5 - D7 should not exceed 100mA.
   If IOH exceeds the test condition, VOH may exceed the related specification. Pins are not guaranteed to source current greater than the listed test condition.
5. Minimum V_{CC} for power-down is 2.5V.
6. The analog comparator Propogation Delay equals 1 comparator clock plus 30nS. See Section 20. “Analog Comparator” on page 225 for comparator clock definition.

26.10 Parallel Programming Characteristics

Figure 26-5. Parallel Programming Timing, Including some General Timing Requirements

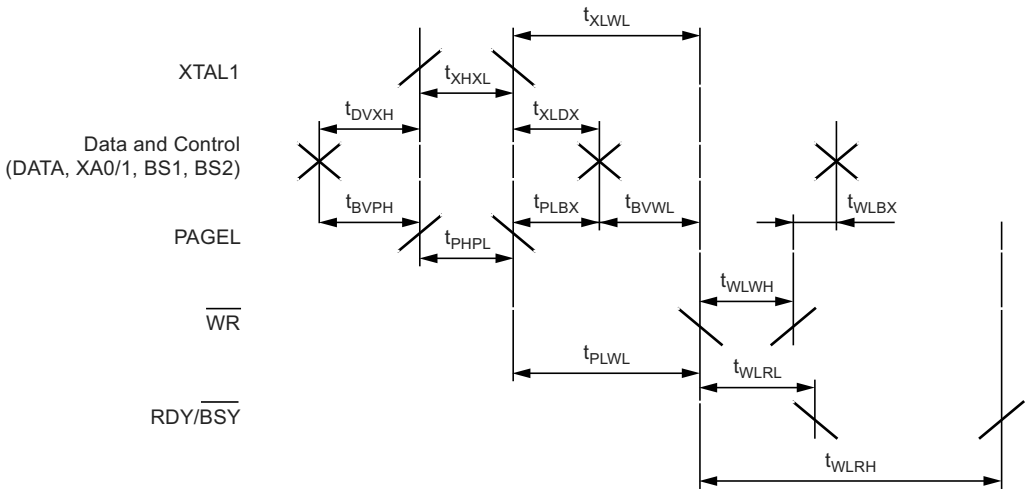
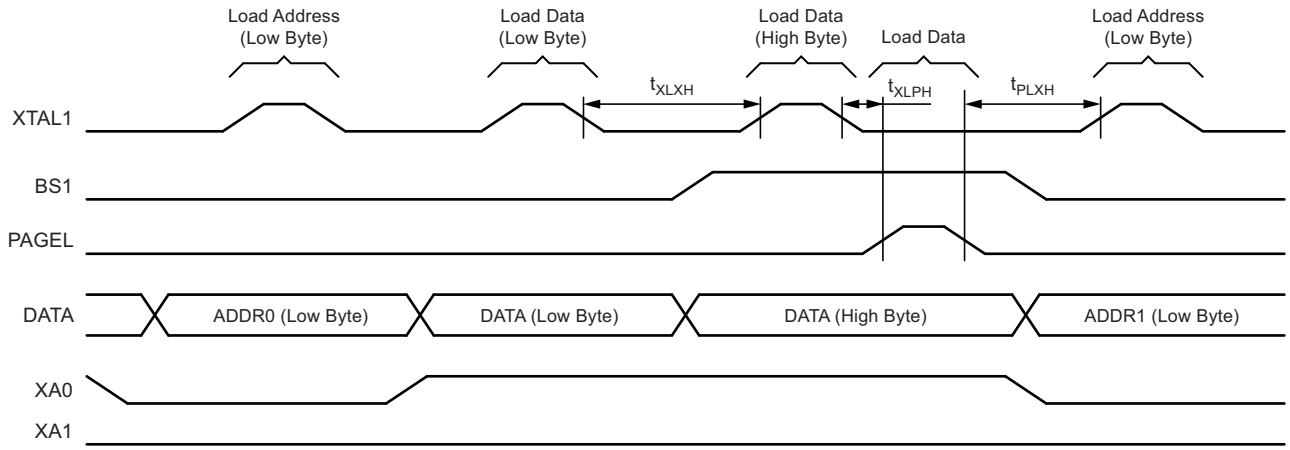


Figure 26-6. Parallel Programming Timing, Loading Sequence with Timing Requirements(1)



Note: 1. The timing requirements shown in Figure 25-7 on page 268 (i.e., t_{DVXH}, t_{XHXL}, and t_{XLDX}) also apply to loading operation.

29. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x10 (0x30)	Reserved	–	–	–	–	–	–	–	–	
0x0F (0x2F)	Reserved	–	–	–	–	–	–	–	–	
0x0E (0x2E)	PORTE	–	–	–	–	–	PORTE2	PORTE1	PORTE0	69
0x0D (0x2D)	DDRE	–	–	–	–	–	DDE2	DDE1	DDE0	69
0x0C (0x2C)	PINE	–	–	–	–	–	PINE2	PINE1	PINE0	69
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	69
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	69
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	69
0x08 (0x28)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	68
0x07 (0x27)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	69
0x06 (0x26)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	69
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	68
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	68
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	68
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–	
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–	
0x00 (0x20)	Reserved	–	–	–	–	–	–	–	–	

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the status flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such status flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATmega16/32/64/M1/C1 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.
5. These registers are only available on ATmega32/64M1. For other products described in this datasheet, these locations are reserved.