EEL 3744: Microprocessor Applications

Summer 2017

Dr. Eric M. Schwartz

XMEGA Documentation

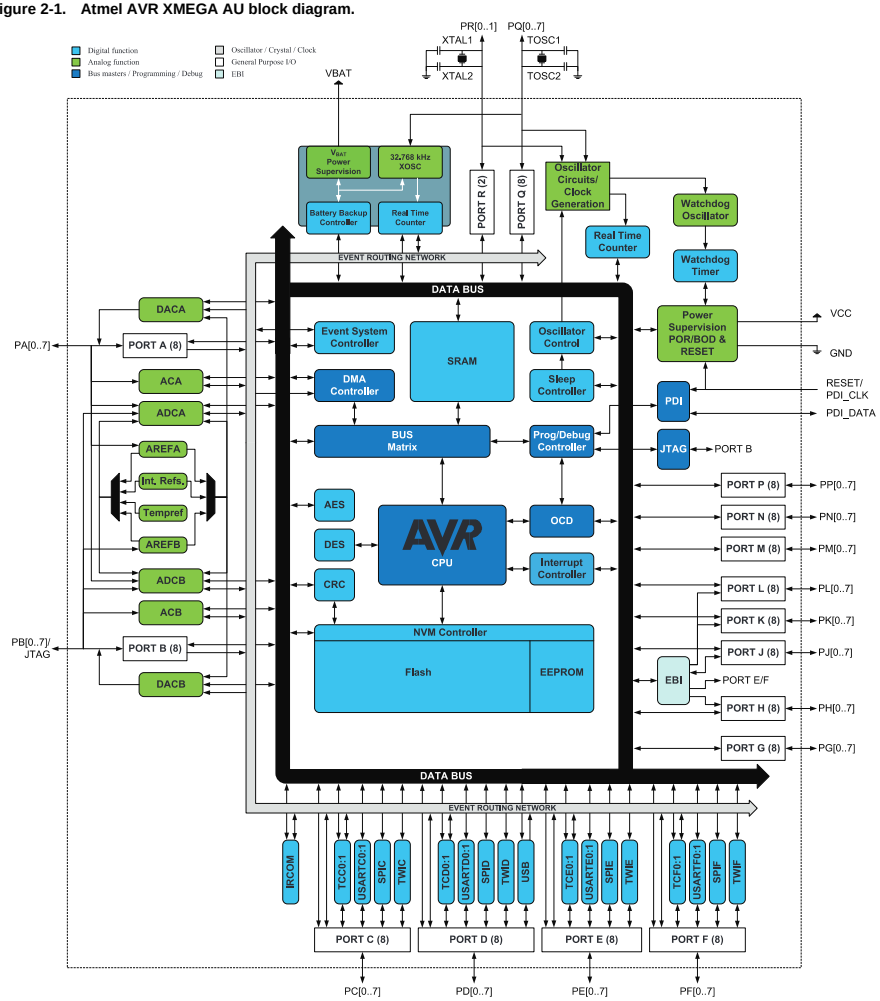
Atmel Documents: pg	Atmel Section	# pages in our doc	Pg # in Our Doc
doc8331: pg 4	2	1	2
doc8331: pg 7-19	3	6	3-9
doc8331: pg 20-25	4	3	10-12
doc8331: pg 53-69	5	9	13-21
doc8331: pg 70-81	6	6	22-27
doc8331: pg 82-102	7	11	28-38
doc8331: pg 131-138	12	4	39-42
doc8331: pg 139-161	13	12	43-54
doc8331: pg 162-185	14	24	55-66
doc8331: pg 212-218	18	4	67-70
doc8331: pg 273-279	22	4	71-74
doc8331: pg 280-300	23	11	75-85
doc8331: pg 319-338	27	10	86-95
doc8331: pg 339-366	28	14	96-109

Atmel Documents: pg	Atmel Section	# pages in our doc	Pg # in Our Doc
doc8331: pg 367-376	29	5	110-114
doc8331: pg 434-440	36	4	115-118
doc8331: pg 458-463	37	3	119-121
doc8331: pg i-ix / 468-477	TOC	5	122-126
doc8385: pg 3	2	1	127
doc8385: pg 11-16	7	3	128-130
doc8385: pg 27-29	14	2	131-132
doc8385: pg 56-62	33	4	133-136
doc8385: pg 66-70	35	3	137-139
doc8385: pg 91-92	37.1.15	1	140
doc0856: pg 1-188		94	141-234
doc1022: pg 9-20	4.5-4.6	6	235-240
Include file Definitions		15	241-255

doc8331: *ATxmega128A1U Manual*
doc0856: *AVR Instruction Set*
doc8385: *XMEGA AU Manual*
doc1022: *Assembler Directives*, Section 4.5-4.6 (from Section 4: *AVR Assembler User Guide*)
ATxmega128A1U Definitions (from include file)

2.1 Block Diagram

Figure 2-1. Atmel AVR XMEGA AU block diagram.



In Table 2-1 on page 5 a feature summary for the XMEGA AU family is shown, split into one feature summary column for each sub-family. Each sub-family has identical feature set, but different memory options, refer to their device datasheet for ordering codes and memory options.

3. AVR CPU

3.1 Features

- 8/16-bit, high-performance Atmel AVR RISC CPU
  - 142 instructions
  - Hardware multiplier
- 32x8-bit registers directly connected to the ALU
- Stack in RAM
- Stack pointer accessible in I/O memory space
- Direct addressing of up to 16MB of program memory and 16MB of data memory
- True 16/24-bit access to 16/24-bit I/O registers
- Efficient support for 8-, 16-, and 32-bit arithmetic
- Configuration change protection of system-critical features

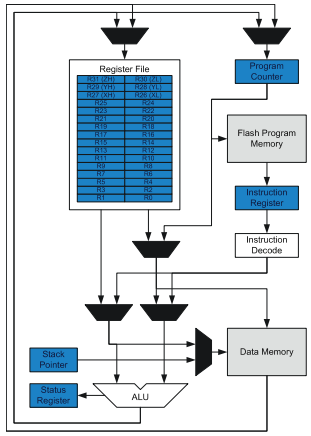
3.2 Overview

All Atmel AVR XMEGA devices use the 8/16-bit AVR CPU. The main function of the CPU is to execute the code and perform all calculations. The CPU is able to access memories, perform calculations, control peripherals, and execute the program in the flash memory. Interrupt handling is described in a separate section, "Interrupts and Programmable Multilevel Interrupt Controller" on page 131.

3.3 Architectural Overview

In order to maximize performance and parallelism, the AVR CPU uses a Harvard architecture with separate memories and buses for program and data. Instructions in the program memory are executed with single-level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This enables instructions to be executed on every clock cycle. For a summary of all AVR instructions, refer to "Instruction Set Summary" on page 429. For details of all AVR instructions, refer to http://www.atmel.com/avr.

Figure 3-1. Block diagram of the AVR CPU architecture.



The arithmetic logic unit (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.

The ALU is directly connected to the fast-access register file. The 32 x 8-bit general purpose working registers all have single clock cycle access time allowing single-cycle arithmetic logic unit operation between registers or between a register and an immediate. Six of the 32 registers can be used as three 16-bit address pointers for program and data space addressing, enabling efficient address calculations.

The memory spaces are linear. The data memory space and the program memory space are two different memory spaces.

The data memory space is divided into I/O registers, SRAM, and external RAM. In addition, the EEPROM can be memory mapped in the data memory.

All I/O status and control registers reside in the lowest 4KB addresses of the data memory. This is referred to as the I/O memory space. The lowest 64 addresses can be accessed directly, or as the data space locations from 0x00 to 0x3F. The rest is the extended I/O memory space, ranging from 0x0040 to 0x0FFF. I/O registers here must be accessed as data space locations using load (LD/LDS/LDD) and store (ST/STS/STD) instructions.

The SRAM holds data. Code execution from SRAM is not supported. It can easily be accessed through the five different addressing modes supported in the AVR architecture. The first SRAM address is 0x2000.

Data addresses 0x1000 to 0x1FFF are reserved for memory mapping of EEPROM.

The program memory is divided in two sections, the application program section and the boot program section. Both sections have dedicated lock bits for write and read/write protection. The SPM instruction that is used for self-programming of the application flash memory must reside in the boot program section. The application section contains an application table section with separate lock bits for write and read/write protection. The application table section can be used for save storing of nonvolatile data in the program memory.

3.4 ALU - Arithmetic Logic Unit

The arithmetic logic unit supports arithmetic and logic operations between registers or between a constant and a register. Single-register operations can also be executed. The ALU operates in direct connection with all 32 general purpose registers. In a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed and the result is stored in the register file. After an arithmetic or logic operation, the status register is updated to reflect information about the result of the operation.

ALU operations are divided into three main categories – arithmetic, logical, and bit functions. Both 8- and 16-bit arithmetic is supported, and the instruction set allows for efficient implementation of 32-bit arithmetic. The hardware multiplier supports signed and unsigned multiplication and fractional format.

3.4.1 Hardware Multiplier

The multiplier is capable of multiplying two 8-bit numbers into a 16-bit result. The hardware multiplier supports different variations of signed and unsigned integer and fractional numbers:

- Multiplication of unsigned integers
- Multiplication of signed integers
- Multiplication of a signed integer with an unsigned integer
- Multiplication of unsigned fractional numbers
- Multiplication of signed fractional numbers
- Multiplication of a signed fractional number with an unsigned one

A multiplication takes two CPU clock cycles.

3.5 Program Flow

After reset, the CPU starts to execute instructions from the lowest address in the flash program memory '0.' The program counter (PC) addresses the next instruction to be fetched.

Program flow is provided by conditional and unconditional jump and call instructions capable of addressing the whole address space directly. Most AVR instructions use a 16-bit word format, while a limited number use a 32-bit format.

During interrupts and subroutine calls, the return address PC is stored on the stack. The stack is 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. After reset, the stack pointer (SP) points to the highest address in the internal SRAM. The SP is read/write accessible in the I/O memory space, enabling easy implementation of multiple stacks or stack areas. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR CPU.

3.6 Instruction Execution Timing

The AVR CPU is clocked by the CPU clock, clk_{CPU}. No internal clock division is used. Figure 3-2 on page 9 shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast-access register file concept. This is the basic pipelining concept used to obtain up to 1MIPS/MHz performance with high power efficiency.

Figure 3-2. The parallel instruction fetches and instruction executions.

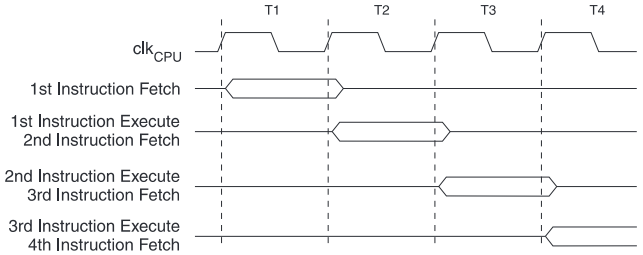
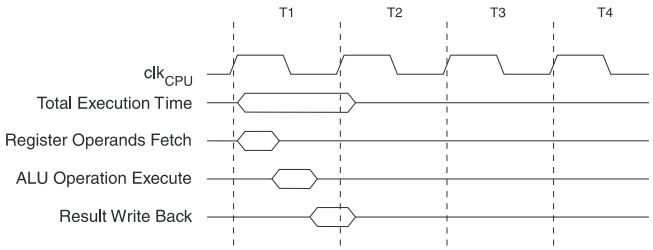


Figure 3-3 on page 9 shows the internal timing concept for the register file. In a single clock cycle, an ALU operation using two register operands is executed and the result is stored back to the destination register.

Figure 3-3. Single Cycle ALU Operation.



3.7 Status Register

The status register (SREG) contains information about the result of the most recently executed arithmetic or logic 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 Summary" on page 429. 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 nor restored when returning from an interrupt. This must be handled by software.

The status register is accessible in the I/O memory space.

3.8 Stack and Stack Pointer

The stack is used for storing return addresses after interrupts and subroutine calls. It can also be used for storing temporary data. The stack pointer (SP) register always points to the top of the stack. It is implemented as two 8-bit registers that are accessible in the I/O memory space. Data are pushed and popped from the stack using the PUSH and POP instructions. The stack grows from a higher memory location to a lower memory location. This implies that pushing data onto the stack decreases the SP, and popping data off the stack increases the SP. The SP is automatically loaded after reset, and the initial value is the highest address of the internal SRAM. If the SP is changed, it must be set to point above address 0x2000, and it must be defined before any subroutine calls are executed or before interrupts are enabled.

During interrupts or subroutine calls, the return address is automatically pushed on the stack. The return address can be two or three bytes, depending on program memory size of the device. For devices with 128KB or less of program memory, the return address is two bytes, and hence the stack pointer is decremented/incremented by two. For devices with more than 128KB of program memory, the return address is three bytes, and hence the SP is decremented/incremented by three. The return address is popped off the stack when returning from interrupts using the RETI instruction, and from subroutine calls using the RET instruction.

The SP is decremented by one when data are pushed on the stack with the PUSH instruction, and incremented by one when data is popped off the stack using the POP instruction.

To prevent corruption when updating the stack pointer from software, a write to SPL will automatically disable interrupts for up to four instructions or until the next I/O memory write.

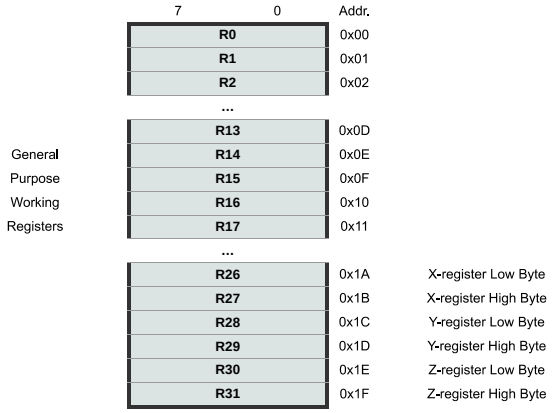
3.9 Register File

The register file consists of 32 x 8-bit general purpose working registers with single clock cycle access time. The register file supports the following input/output schemes:

- 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

Six of the 32 registers can be used as three 16-bit address register pointers for data space addressing, enabling efficient address calculations. One of these address pointers can also be used as an address pointer for lookup tables in flash program memory.

Figure 3-4. AVR CPU general purpose working registers.



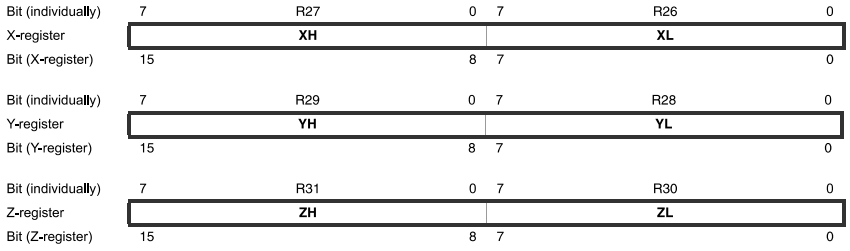
The register file is located in a separate address space, and so the registers are not accessible as data memory.

3.9.1 The X-, Y-, and Z- Registers

Registers R26..R31 have added functions besides their general-purpose usage.

These registers can form 16-bit address pointers for addressing data memory. These three address registers are called the X-register, Y-register, and Z-register. The Z-register can also be used as an address pointer to read from and/or write to the flash program memory, signature rows, fuses, and lock bits.

Figure 3-5. The X-, Y- and Z-registers.



The lowest register address holds the least-significant byte (LSB), and the highest register address holds the most-significant byte (MSB). In the different addressing modes, these address registers function as fixed displacement, automatic increment, and automatic decrement (see "Instruction Set Summary" on page 429 for details).

3.10 RAMP and Extended Indirect Registers

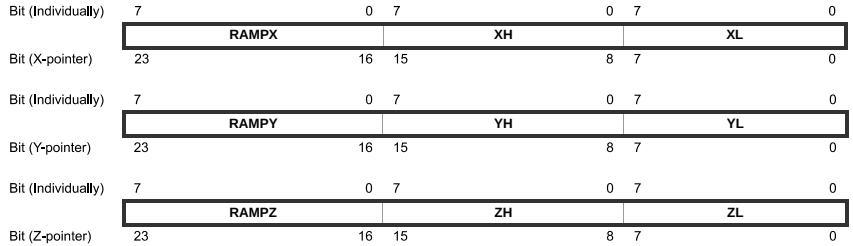
In order to access program memory or data memory above 64KB, the address pointer must be larger than 16 bits. This is done by concatenating one register to one of the X-, Y-, or Z-registers. This register then holds the most-significant byte (MSB) in a 24-bit address or address pointer.

These registers are available only on devices with external bus interface and/or more than 64KB of program or data memory space. For these devices, only the number of bits required to address the whole program and data memory space in the device is implemented in the registers.

3.10.1 RAMPX, RAMPY and RAMPZ Registers

The RAMPX, RAMPY and RAMPZ registers are concatenated with the X-, Y-, and Z-registers, respectively, to enable indirect addressing of the whole data memory space above 64KB and up to 16MB.

Figure 3-6. The combined RAMPX + X, RAMPY + Y and RAMPZ + Z registers.



When reading (ELPM) and writing (SPM) program memory locations above the first 128KB of the program memory, RAMPZ is concatenated with the Z-register to form the 24-bit address. LPM is not affected by the RAMPZ setting.

3.10.2 RAMPD Register

This register is concatenated with the operand to enable direct addressing of the whole data memory space above 64KB. Together, RAMPD and the operand will form a 24-bit address.

Figure 3-7. The combined RAMPD + K register.

3.10.3 EIND - Extended Indirect Register

EIND is concatenated with the Z-register to enable indirect jump and call to locations above the first 128KB (64K words) of the program memory.

Figure 3-8. The combined EIND + Z register.

3.11 Accessing 16-bit Registers

The AVR data bus is 8 bits wide, and so accessing 16-bit registers requires atomic operations. These registers must be byte-accessed using two read or write operations. 16-bit registers are connected to the 8-bit bus and a temporary register using a 16-bit bus.

For a write operation, the low byte of the 16-bit register must be written before the high byte. The low byte is then written into the temporary register. When the high byte of the 16-bit register is written, the temporary register is copied into the low byte of the 16-bit register in the same clock cycle.

For a read operation, the low byte of the 16-bit register must be read before the high byte. When the low byte register is read by the CPU, the high byte of the 16-bit register is copied into the temporary register in the same clock cycle as the low byte is read. When the high byte is read, it is then read from the temporary register.

This ensures that the low and high bytes of 16-bit registers are always accessed simultaneously when reading or writing the register.

Interrupts can corrupt the timed sequence if an interrupt is triggered and accesses the same 16-bit register during an atomic 16-bit read/write operation. To prevent this, interrupts can be disabled when writing or reading 16-bit registers.

The temporary registers can also be read and written directly from user software.

3.11.1 Accessing 24- and 32-bit Registers

For 24- and 32-bit registers, the read and write access is done in the same way as described for 16-bit registers, except there are two temporary registers for 24-bit registers and three for 32-bit registers. The least-significant byte must be written first when doing a write, and read first when doing a read.

3.12 Configuration Change Protection

System critical I/O register settings are protected from accidental modification. The SPM instruction is protected from accidental execution, and the LPM instruction is protected when reading the fuses and signature row. This is handled globally by the configuration change protection (CCP) register. Changes to the protected I/O registers or bits, or execution of protected instructions, are only possible after the CPU writes a signature to the CCP register. The different signatures are described in the register description.

There are two modes of operation: one for protected I/O registers, and one for the protected instructions, SPM/LPM.

3.12.1 Sequence for write operation to protected I/O registers

1. The application code writes the signature that enable change of protected I/O registers to the CCP register.
2. Within four instruction cycles, the application code must write the appropriate data to the protected register. Most protected registers also contain a write enable/change enable bit. This bit must be written to one in the same operation as the data are written. The protected change is immediately disabled if the CPU performs write operations to the I/O register or data memory or if the SPM, LPM, or SLEEP instruction is executed.

3.12.2 Sequence for execution of protected SPM/LPM

1. The application code writes the signature for the execution of protected SPM/LPM to the CCP register.
2. Within four instruction cycles, the application code must execute the appropriate instruction. The protected change is immediately disabled if the CPU performs write operations to the data memory or if the SLEEP instruction is executed.

Once the correct signature is written by the CPU, interrupts will be ignored for the duration of the configuration change enable period. Any interrupt request (including non-maskable interrupts) during the CCP period will set the corresponding interrupt flag as normal, and the request is kept pending. After the CCP period is completed, any pending interrupts are executed according to their level and priority. DMA requests are still handled, but do not influence the protected configuration change enable period. A signature written by DMA is ignored.

3.13 Fuse Lock

For some system-critical features, it is possible to program a fuse to disable all changes to the associated I/O control registers. If this is done, it will not be possible to change the registers from the user software, and the fuse can only be reprogrammed using an external programmer. Details on this are described in the datasheet module where this feature is available.

3.14 Register Descriptions

3.14.1 CCP – Configuration Change Protection register

Bit	7	6	5	4	3	2	1	0
+0x04	CCP[7:0]							
Read/Write	W	W	W	W	W	W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7:0 – CCP[7:0]: Configuration Change Protection**
  The CCP register must be written with the correct signature to enable change of the protected I/O register or execution of the protected instruction for a maximum period of four CPU instruction cycles. All interrupts are ignored during these cycles. After these cycles, interrupts will automatically be handled again by the CPU, and any pending interrupts will be executed according to their level and priority. When the protected I/O register signature is written, CCP[0] will read as one as long as the protected feature is enabled. Similarly when the protected SPM/LPM signature is written, CCP[1] will read as one as long as the protected feature is enabled. CCP[7:2] will always read as zero. Table 3-1 on page 15 shows the signature for the various modes.

Table 3-1. Modes of CPU change protection.

Signature	Group configuration	Description
0x9D	SPM	Protected SPM/LPM
0xD8	IOREG	Protected IO register

3.14.2 RAMPD – Extended Direct Addressing register

This register is concatenated with the operand for direct addressing (LDS/STS) of the whole data memory space on devices with more than 64KB of data memory. This register is not available if the data memory, including external memory, is less than 64KB.

Bit	7	6	5	4	3	2	1	0
+0x08	RAMPD[7:0]							
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 7:0 – RAMPD[7:0]: Extended Direct Addressing Bits**
  These bits hold the MSB of the 24-bit address created by RAMPD and the 16-bit operand. Only the number of bits required to address the available data memory is implemented for each device. Unused bits will always read as zero.

3.14.3 RAMPX – Extended X-Pointer register

This register is concatenated with the X-register for indirect addressing (LD/LDD/ST/STD) of the whole data memory space on devices with more than 64KB of data memory. This register is not available if the data memory, including external memory, is less than 64KB.

Bit	7	6	5	4	3	2	1	0
+0x09	RAMPX[7:0]							
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 7:0 – RAMPX[7:0]: Extended X-pointer Address Bits**
  These bits hold the MSB of the 24-bit address created by RAMPX and the 16-bit X-register. Only the number of bits required to address the available data memory is implemented for each device. Unused bits will always read as zero.

3.14.4 RAMPY – Extended Y-Pointer register

This register is concatenated with the Y-register for indirect addressing (LD/LDD/ST/STD) of the whole data memory space on devices with more than 64KB of data memory. This register is not available if the data memory, including external memory, is less than 64KB.

Bit	7	6	5	4	3	2	1	0
+0x0A	RAMPY[7:0]							
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 7:0 – RAMPY[7:0]: Extended Y-pointer Address Bits**
  These bits hold the MSB of the 24-bit address created by RAMPY and the 16-bit Y-register. Only the number of bits required to address the available data memory is implemented for each device. Unused bits will always read as zero.

3.14.5 RAMPZ – Extended Z-Pointer register

This register is concatenated with the Z-register for indirect addressing (LD/LDD/ST/STD) of the whole data memory space on devices with more than 64KB of data memory. RAMPZ is concatenated with the Z-register when reading (ELPM) program memory locations above the first 64KB and writing (SPM) program memory locations above the first 128KB of the program memory.

This register is not available if the data memory, including external memory and program memory in the device, is less than 64KB.

Bit	7	6	5	4	3	2	1	0
+0x0B	RAMPZ[7:0]							
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 7:0 – RAMPZ[7:0]: Extended Z-pointer Address Bits**
  These bits hold the MSB of the 24-bit address created by RAMPZ and the 16-bit Z-register. Only the number of bits required to address the available data and program memory is implemented for each device. Unused bits will always read as zero.

3.14.6 EIND – Extended Indirect register

This register is concatenated with the Z-register for enabling extended indirect jump (EIJMP) and call (EICALL) to the whole program memory space on devices with more than 128KB of program memory. The register should be used for jumps to addresses below 128KB if ECALL/EIJMP are used, and it will not be used if CALL and IJMP commands are used. For jump or call to addresses below 128KB, this register is not used. This register is not available if the program memory in the device is less than 128KB.

Bit	7	6	5	4	3	2	1	0
+0x0C	EIND[7:0]							
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 7:0 – EIND[7:0]: Extended Indirect Address Bits**
  These bits hold the MSB of the 24-bit address created by EIND and the 16-bit Z-register. Only the number of bits required to access the available program memory is implemented for each device. Unused bits will always read as zero.

3.14.7 SPL – Stack Pointer register Low

The SPH and SPL stack pointer pair represent the 16-bit SP value. The SP holds the stack pointer that points to the top of the stack. After reset, the stack pointer points to the highest internal SRAM address. To prevent corruption when updating the stack pointer from software, a write to SPL will automatically disable interrupts for the next four instructions or until the next I/O memory write.

Only the number of bits required to address the available data memory, including external memory, up to 64KB is implemented for each device. Unused bits will always read as zero.

Bit	7	6	5	4	3	2	1	0
+0x0D	SP[7:0]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value(1)	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

Note: 1. Refer to specific device datasheets for exact size.

- **Bit 7:0 – SP[7:0]: Stack Pointer Low Byte**
  These bits hold the LSB of the 16-bit stack pointer (SP).

3.14.8 SPH – Stack Pointer register High

Bit	7	6	5	4	3	2	1	0
+0x0E	SP[15:8]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial value(1)	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

Note: 1. Refer to specific device datasheets for the exact size.

- **Bit 7:0 – SP[15:8]: Stack Pointer High Byte**
  These bits hold the MSB of the 16-bit stack pointer (SP).

3.14.9 SREG – Status register

The status register (SREG) contains information about the result of the most recently executed arithmetic or logic instruction. For details information about the bits in this register and how they are affected by the different instructions see "Instruction Set Summary" on page 429.

Bit	7	6	5	4	3	2	1	0
+0x0F	**I**	**T**	**H**	**S**	**V**	**N**	**Z**	**C**
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 for interrupts to be enabled. If the global interrupt enable register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. This bit is not cleared by hardware after an interrupt has occurred. This bit can be set and cleared by the application with the SEI and CLI instructions, as described in "Instruction Set Summary" on page 429. Changing the I flag through the I/O-register result in a one-cycle wait state on the access.

- **Bit 6 – T: Bit Copy Storage**
  The bit copy instructions bit load (BLD) and bit store (BST) use the T bit as source or destination for the operated bit. A bit from a register in the register file can be copied into this bit by the BST instruction, and this bit can be copied into a bit in a register in the register file by the BLD instruction.
- **Bit 5 – H: Half Carry Flag**
  The half carry flag (H) indicates a half carry in some arithmetic operations. Half carry is useful in BCD arithmetic.
- **Bit 4 – S: Sign Bit, S = N ⊕ V**
  The sign bit is always an exclusive or between the negative flag, N, and the two's complement overflow flag, V.
- **Bit 3 – V: Two's Complement Overflow Flag**
  The two's complement overflow flag (V) supports two's complement arithmetic.
- **Bit 2 – N: Negative Flag**
  The negative flag (N) indicates a negative result in an arithmetic or logic operation.
- **Bit 1 – Z: Zero Flag**
  The zero flag (Z) indicates a zero result in an arithmetic or logic operation.
- **Bit 0 – C: Carry Flag**
  The carry flag (C) indicates a carry in an arithmetic or logic operation.

3.15 Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	Reserved	–	–	–	–	–	–	–	–	
+0x01	Reserved	–	–	–	–	–	–	–	–	
+0x02	Reserved	–	–	–	–	–	–	–	–	
+0x03	Reserved	–	–	–	–	–	–	–	–	
+0x04	CCP	CCP[7:0]								15
+0x05	Reserved	–	–	–	–	–	–	–	–	
+0x06	Reserved	–	–	–	–	–	–	–	–	
+0x07	Reserved	–	–	–	–	–	–	–	–	
+0x08	RAMPD	RAMPD[7:0]								15
+0x09	RAMPX	RAMPX[7:0]								15
+0x0A	RAMPY	RAMPY[7:0]								16
+0x0B	RAMPZ	RAMPZ[7:0]								16
+0x0C	EIND	EIND[7:0]								16
+0x0D	SPL	SPL[7:0]								17
+0x0E	SPH	SPH[7:0]								17
+0x0F	SREG	I	T	H	S	V	N	Z	C	17

4. Memories

4.1 Features

- Flash program memory
  - One linear address space
  - In-system programmable
  - Self-programming and boot loader support
  - Application section for application code
  - Application table section for application code or data storage
  - Boot section for application code or bootloader code
  - Separate read/write protection lock bits for all sections
  - Built in fast CRC check of a selectable flash program memory section
- Data memory
  - One linear address space
  - Single-cycle access from CPU
  - SRAM
  - EEPROM
    - Byte and page accessible
    - Optional memory mapping for direct load and store
  - I/O memory
    - Configuration and status registers for all peripherals and modules
    - 16 bit-accessible general purpose registers for global variables or flags
  - External memory support
    - SRAM
    - SDRAM
    - Memory mapped external hardware
  - Bus arbitration
    - Deterministic handling of priority between CPU, DMA controller, and other bus masters
  - Separate buses for SRAM, EEPROM, I/O memory, and external memory access
    - Simultaneous bus access for CPU and DMA controller
- Production signature row memory for factory programmed data
  - ID for each microcontroller device type
  - Serial number for each device
  - Calibration bytes for factory calibrated peripherals
- User signature row
  - One flash page in size
  - Can be read and written from software
  - Content is kept after chip erase

4.2 Overview

This section describes the different memory sections. The AVR architecture has two main memory spaces, the program memory and the data memory. Executable code can reside only in the program memory, while data can be stored in the program memory and the data memory. The data memory includes the internal SRAM, and EEPROM for nonvolatile data storage. All memory spaces are linear and require no memory bank switching. Nonvolatile memory (NVM) spaces can be locked for further write and read/write operations. This prevents unrestricted access to the application software.

A separate memory section contains the fuse bytes. These are used for configuring important system functions, and can only be written by an external programmer.

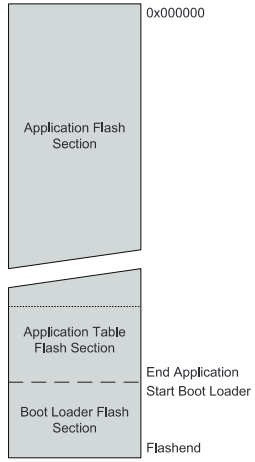
4.3 Flash Program Memory

All XMEGA devices contain on-chip, in-system reprogrammable flash memory for program storage. The flash memory can be accessed for read and write from an external programmer through the PDI or from application software running in the device.

All AVR CPU instructions are 16 or 32 bits wide, and each flash location is 16 bits wide. The flash memory is organized in two main sections, the application section and the boot loader section, as shown in Figure 4-1 on page 21. The sizes of the different sections are fixed, but device-dependent. These two sections have separate lock bits, and can have different levels of protection. The store program memory (SPM) instruction, used to write to the flash from the application software, will only operate when executed from the boot loader section.

The application section contains an application table section with separate lock settings. This enables safe storage of nonvolatile data in the program memory.

Figure 4-1. Flash memory sections.



4.3.1 Application Section

The Application section is the section of the flash that is used for storing the executable application code. The protection level for the application section can be selected by the boot lock bits for this section. The application section can not store any boot loader code since the SPM instruction cannot be executed from the application section.

4.3.2 Application Table Section

The application table section is a part of the application section of the flash memory that can be used for storing data. The size is identical to the boot loader section. The protection level for the application table section can be selected by the boot lock bits for this section. The possibilities for different protection levels on the application section and the application table section enable safe parameter storage in the program memory. If this section is not used for data, application code can reside here.

4.3.3 Boot Loader Section

While the application section is used for storing the application code, the boot loader software must be located in the boot loader section because the SPM instruction can initiate programming when executing from this section. The SPM instruction can access the entire flash, including the boot loader section itself. The protection level for the boot loader section can be selected by the boot loader lock bits. If this section is not used for boot loader software, application code can be stored here.

4.3.4 Production Signature Row

The production signature row is a separate memory section for factory programmed data. It contains calibration data for functions such as oscillators and analog modules. Some of the calibration values will be automatically loaded to the corresponding module or peripheral unit during reset. Other values must be loaded from the signature row and written to the corresponding peripheral registers from software. For details on calibration conditions such as temperature, voltage references, etc., refer to the device datasheet.

The production signature row also contains an ID that identifies each microcontroller device type and a serial number for each manufactured device. The serial number consists of the production lot number, wafer number, and wafer coordinates for the device.

The production signature row cannot be written or erased, but it can be read from application software and external programmers.

4.3.5 User Signature Row

The user signature row is a separate memory section that is fully accessible (read and write) from application software and external programmers. It is one flash page in size, and is meant for static user parameter storage, such as calibration data, custom serial number, identification numbers, random number seeds, etc. This section is not erased by chip erase commands that erase the flash, and requires a dedicated erase command. This ensures parameter storage during multiple program/erase operations and on-chip debug sessions.

4.4 Fuses and Lockbits

The fuses are used to configure important system functions, and can only be written from an external programmer. The application software can read the fuses. The fuses are used to configure reset sources such as brownout detector and watchdog, startup configuration, JTAG enable, and JTAG user ID.

The lock bits are used to set protection levels for the different flash sections (i.e., if read and/or write access should be blocked). Lock bits can be written by external programmers and application software, but only to stricter protection levels. Chip erase is the only way to erase the lock bits. To ensure that flash contents are protected even during chip erase, the lock bits are erased after the rest of the flash memory has been erased.

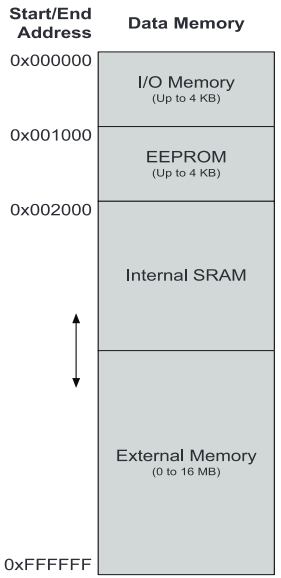
An unprogrammed fuse or lock bit will have the value one, while a programmed fuse or lock bit will have the value zero.

Both fuses and lock bits are reprogrammable like the flash program memory.

4.5 Data Memory

The data memory contains the I/O memory, internal SRAM, optionally memory mapped EEPROM, and external memory, if available. The data memory is organized as one continuous memory section, as shown in Figure 4-2 on page 23.

Figure 4-2. Data memory map.



I/O memory, EEPROM, and SRAM will always have the same start addresses for all XMEGA devices. The address space for external memory will always start at the end of internal SRAM and end at address 0xFFFFFF.

4.6 Internal SRAM

The internal SRAM always starts at hexadecimal address 0x2000. SRAM is accessed by the CPU using the load (LD/LDS/LDD) and store (ST/STS/STD) instructions.

4.7 EEPROM

All XMEGA devices have EEPROM for nonvolatile data storage. It is addressable in a separate data space (default) or memory mapped and accessed in normal data space. The EEPROM supports both byte and page access. Memory mapped EEPROM allows highly efficient EEPROM reading and EEPROM buffer loading. When doing this, EEPROM is accessible using load and store instructions. Memory mapped EEPROM will always start at hexadecimal address 0x1000. However, flushing the buffer and erasing and writing pages must still be done through the NVM controller as for I/O-mapped access.

4.8 I/O Memory

The status and configuration registers for peripherals and modules, including the CPU, are addressable through I/O memory locations. All I/O locations can be accessed by the load (LD/LDS/LDD) and store (ST/STS/STD) instructions, which are used to transfer data between the 32 registers in the register file and the I/O memory. The IN and OUT instructions can address I/O memory locations in the range of 0x00 to 0x3F directly. In the address range 0x00 - 0x1F, single-cycle instructions for manipulation and checking of individual bits are available.

4.8.1 General Purpose I/O Registers

The lowest 16 I/O memory addresses are reserved as general purpose I/O registers. These registers can be used for storing global variables and flags, as they are directly bit-accessible using the SBI, CBI, SBIS, and SBIC instructions.

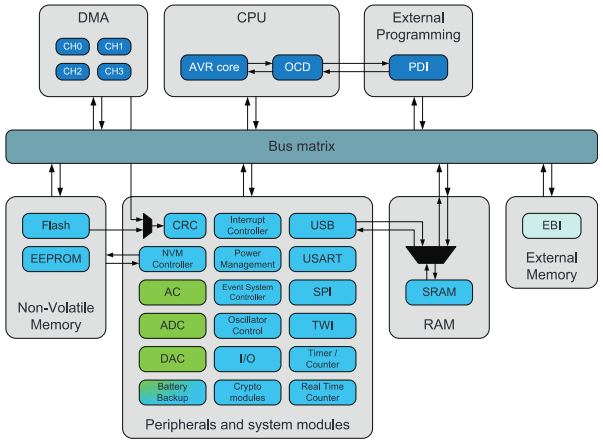
4.9 External Memory

Up to four ports are dedicated to external memory, supporting external SRAM, SDRAM, and memory mapped peripherals such as LCD displays. For details, refer to "EBI – External Bus Interface" on page 319. The external memory address space will always start at the end of internal SRAM.

4.10 Data Memory and Bus Arbitration

Since the data memory is organized as four separate sets of memories, the different bus masters (CPU, DMA controller read and DMA controller write, etc.) can access different memory sections at the same time. See Figure 4-3 on page 24. The USB module acts as a bus master, and is connected directly to internal SRAM through a pseudo-dual-port (PDP) interface.

Figure 4-3. Bus access.



4.10.1 Bus Priority

When several masters request access to the same bus, the bus priority is in the following order (from higher to lower priority):

1. Bus Master with ongoing access.
2. Bus Master with ongoing burst.
    1. Alternating DMA controller read and DMA controller write when they access the same data memory section.
3. Bus Master requesting burst access.
    1. CPU has priority.
4. Bus Master requesting bus access.
    1. CPU has priority.

4.11 Memory Timing

Read and write access to the I/O memory takes one CPU clock cycle. A write to SRAM takes one cycle, and a read from SRAM takes two cycles. For burst read (DMA), new data are available every cycle. EEPROM page load (write) takes one cycle, and three cycles are required for read. For burst read, new data are available every second cycle. External memory has multi-cycle read and write. The number of cycles depends on the type of memory and configuration of the external bus interface. Refer to the instruction summary for more details on instructions and instruction timing.

4.12 Device ID and Revision

Each device has a three-byte device ID. This ID identifies Atmel as the manufacturer of the device and the device type. A separate register contains the revision number of the device.

4.13 JTAG Disable

It is possible to disable the JTAG interface from the application software. This will prevent all external JTAG access to the device until the next device reset or until JTAG is enabled again from the application software. As long as JTAG is disabled, the I/O pins required for JTAG can be used as normal I/O pins.

4.14 I/O Memory Protection

Some features in the device are regarded as critical for safety in some applications. Due to this, it is possible to lock the I/O register related to the clock system, the event system, and the advanced waveform extensions. As long as the lock is enabled, all related I/O registers are locked and they can not be written from the application software. The lock registers themselves are protected by the configuration change protection mechanism. For details, refer to "Configuration Change Protection" on page 13.

5. DMAC - Direct Memory Access Controller

5.1 Features

- Allows high speed data transfers with minimal CPU intervention
  - from data memory to data memory
  - from data memory to peripheral
  - from peripheral to data memory
  - from peripheral to peripheral
- Four DMA channels with separate
  - transfer triggers
  - interrupt vectors
  - addressing modes
- Programmable channel priority
- From 1 byte to 16MB of data in a single transaction
  - Up to 64KB block transfers with repeat
  - 1, 2, 4, or 8 byte burst transfers
- Multiple addressing modes
  - Static
  - Incremental
  - Decremental
- Optional reload of source and destination addresses at the end of each
  - Burst
  - Block
  - Transaction
- Optional interrupt on end of transaction
- Optional connection to CRC generator for CRC on DMA data

5.2 Overview

The four-channel direct memory access (DMA) controller can transfer data between memories and peripherals, and thus off load these tasks from the CPU. It enables high data transfer rates with minimum CPU intervention, and frees up CPU time. The four DMA channels enable up to four independent and parallel transfers.

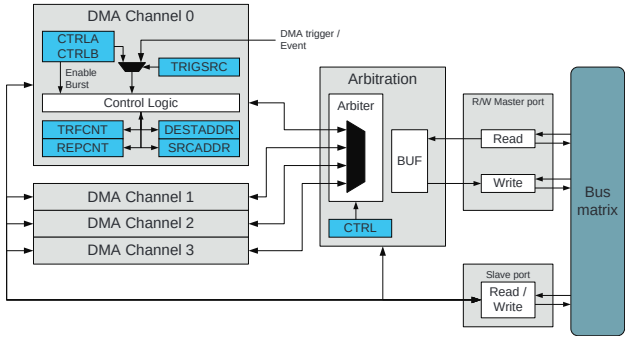
The DMA controller can move data between SRAM and peripherals, between SRAM locations and directly between peripheral registers. With access to all peripherals, the DMA controller can handle automatic transfer of data to/from communication modules. The DMA controller can also read from memory mapped EEPROM.

Data transfers are done in continuous bursts of 1, 2, 4, or 8 bytes. They build block transfers of configurable size from 1 byte to 64KB. A repeat counter can be used to repeat each block transfer for single transactions up to 16MB. Source and destination addressing can be static, incremental or decremental. Automatic reload of source and/or destination addresses can be done after each burst or block transfer, or when a transaction is complete. Application software, peripherals, and events can trigger DMA transfers.

The four DMA channels have individual configuration and control settings. This include source, destination, transfer triggers, and transaction sizes. They have individual interrupt settings. Interrupt requests can be generated when a transaction is complete or when the DMA controller detects an error on a DMA channel.

To allow for continuous transfers, two channels can be interlinked so that the second takes over the transfer when the first is finished, and vice versa.

Figure 5-1. DMA Overview.



5.3 DMA Transaction

A complete DMA read and write operation between memories and/or peripherals is called a DMA transaction. A transaction is done in data blocks, and the size of the transaction (number of bytes to transfer) is selectable from software and controlled by the block size and repeat counter settings. Each block transfer is divided into smaller bursts.

5.3.1 Block Transfer and Repeat

The size of the block transfer is set by the block transfer count register, and can be anything from 1 byte to 64KB.

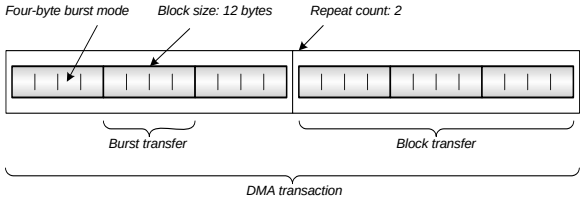
A repeat counter can be enabled to set a number of repeated block transfers before a transaction is complete. The repeat is from 1 to 255, and an unlimited repeat count can be achieved by setting the repeat count to zero.

5.3.2 Burst Transfer

Since the AVR CPU and DMA controller use the same data buses, a block transfer is divided into smaller burst transfers. The burst transfer is selectable to 1, 2, 4, or 8 bytes. This means that if the DMA acquires the data bus and a transfer request is pending, it will occupy the bus until all bytes in the burst are transferred.

A bus arbiter controls when the DMA controller and the AVR CPU can use the bus. The CPU always has priority, and so as long as the CPU requests access to the bus, any pending burst transfer must wait. The CPU requests bus access when it executes an instruction that writes or reads data to SRAM, I/O memory, EEPROM or the external bus interface. For more details on memory access bus arbitration, refer to “Data Memory” on page 22.

Figure 5-2. DMA transaction.



5.4 Transfer Triggers

DMA transfers can be started only when a DMA transfer request is detected. A transfer request can be triggered from software, from an external trigger source (peripheral), or from an event. There are dedicated source trigger selections for each DMA channel. The available trigger sources may vary from device to device, depending on the modules or peripherals that exist in the device. Using a transfer trigger for a module or peripherals that does not exist will have no effect. For a list of all transfer triggers, refer to "TRIGSRC – Trigger Source" on page 62.

By default, a trigger starts a block transfer operation. When the block transfer is complete, the channel is automatically disabled. When enabled again, the channel will wait for the next block transfer trigger. It is possible to select the trigger to start a burst transfer instead of a block transfer. This is called a single-shot transfer, and for each trigger only one burst is transferred. When repeat mode is enabled, the next block transfer does not require a transfer trigger. It will start as soon as the previous block is done.

If the trigger source generates a transfer request during an ongoing transfer, this will be kept pending, and the transfer can start when the ongoing one is done. Only one pending transfer can be kept, and so if the trigger source generates more transfer requests when one is already pending, these will be lost.

5.5 Addressing

The source and destination address for a DMA transfer can either be static or automatically incremented or decremented, with individual selections for source and destination. When address increment or decrement is used, the default behaviour is to update the address after each access. The original source and destination addresses are stored by the DMA controller, and so the source and destination addresses can be individually configured to be reloaded at the following points:

- End of each burst transfer
- End of each block transfer
- End of transaction
- Never reloaded

5.6 Priority Between Channels

If several channels request a data transfer at the same time, a priority scheme is available to determine which channel is allowed to transfer data. Application software can decide whether one or more channels should have a fixed priority or if a round robin scheme should be used. A round robin scheme means that the channel that last transferred data will have the lowest priority.

5.7 Double Buffering

To allow for continuous transfer, two channels can be interlinked so that the second takes over the transfer when the first is finished, and vice versa. This leaves time for the application to process the data transferred by the first channel, prepare fresh data buffers, and set up the channel registers again while the second channel is working. This is referred to as double buffering or chained transfers.

When double buffering is enabled for a channel pair, it is important that the two channels are configured with the same repeat count. The block sizes need not be equal, but for most applications they should be, along with the rest of the channel's operation mode settings.

Note that the double buffering channel pairs are limited to channels 0 and 1 as the first pair and channels 2 and 3 as the second pair. However, it is possible to have one pair operate in double buffered mode while the other is left unused or operating independently.

5.8 Transfer Buffers

To avoid unnecessary bus loading when doing data transfer between memories with different access timing (for example, I/O register and external memory), the DMA controller has a four-byte buffer. Two bytes will be read from the source address and written to this buffer before a write to the destination is started.

5.9 Error detection

The DMA controller can detect erroneous operation. Error conditions are detected individually for each DMA channel, and the error conditions are:

- Write to memory mapped EEPROM locations
- Reading EEPROM when the EEPROM is off (sleep entered)
- DMA controller or a busy channel is disabled in software during a transfer

5.10 Software Reset

Both the DMA controller and a DMA channel can be reset from the user software. When the DMA controller is reset, all registers associated with the DMA controller, including channels, are cleared. A software reset can be done only when the DMA controller is disabled.

When a DMA channel is reset, all registers associated with the DMA channel are cleared. A software reset can be done only when the DMA channel is disabled.

5.11 Protection

In order to ensure safe operation, some of the channel registers are protected during a transaction. When the DMA channel busy flag (CHnBUSY) is set for a channel, the user can modify only the following registers and bits:

- CTRL register
- INTFLAGS register
- TEMP registers
- CHEN, CHRST, TRFREQ, and REPEAT bits of the channel CTRL register
- TRIGSRC register

5.12 Interrupts

The DMA controller can generate interrupts when an error is detected on a DMA channel or when a transaction is complete for a DMA channel. Each DMA channel has a separate interrupt vector, and there are different interrupt flags for error and transaction complete.

If repeat is not enabled, the transaction complete flag is set at the end of the block transfer. If unlimited repeat is enabled, the transaction complete flag is also set at the end of each block transfer.

5.13 Register Description – DMA Controller

5.13.1 CTRL – Control register

Bit	7	6	5	4	3	2	1	0
+0x00	ENABLE	RESET	–	–	DBUFMODE[1:0]		PRIMODE[1:0]	
Read/Write	R/W	R/W	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7 – ENABLE: Enable**
  Setting this bit enables the DMA controller. If the DMA controller is enabled and this bit is written to zero, the ENABLE bit is not cleared before the internal transfer buffer is empty, and the DMA data transfer is aborted.
- **Bit 6 – RESET: Software Reset**
  Writing a one to RESET will be ignored as long as DMA is enabled (ENABLE = 1). This bit can be set only when the DMA controller is disabled (ENABLE = 0).
- **Bit 5:4 – Reserved**
  These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.
- **Bit 3:2 – DBUFMODE[1:0]: Double Buffer Mode**
  These bits enable the double buffer on the different channels according to Table 5-1.

Table 5-1. DMA double buffer settings.

DBUFMODE[1:0]	Group configuration	Description
00	DISABLED	No double buffer enabled
01	CH01	Double buffer enabled on channel0/1
10	CH23	Double buffer enabled on channel2/3
11	CH01CH23	Double buffer enabled on channel0/1 and channel2/3

- **Bit 1:0 – PRIMODE[1:0]: Channel Priority Mode**
  These bits determine the internal channel priority according to Table 5-2

Table 5-2. DMA channel priority settings.

PRIMODE[1:0]	Group configuration	Description
00	RR0123	Round robin
01	CH0RR123	Channel0 > Round robin (channel 1, 2 and 3)
10	CH01RR23	Channel0 > Channel1 > Round robin (channel 2 and 3)
11	CH0123	Channel0 > Channel1 > Channel2 > Channel3

5.13.2 INTFLAGS – Interrupt Status register

Bit	7	6	5	4	3	2	1	0
+0x03	CH3ERRIF	CH2ERRIF	CH1ERRIF	CH0ERRIF	CH3TRNFIF	CH2TRNFIF	CH1TRNFIF	CH0TRNFIF
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 7:4 – CHnERRIF[3:0]: Channel n Error Interrupt Flag**
  If an error condition is detected on DMA channel n, the CHnERRIF flag will be set. Writing a one to this bit location will clear the flag.
- **Bit 3:0 – CHnTRNFIF[3:0]: Channel n Transaction Complete Interrupt Flag**
  When a transaction on channel n has been completed, the CHnTRFIF flag will be set. If unlimited repeat count is enabled, this flag is read as one after each block transfer. Writing a one to this bit location will clear the flag.

5.13.3 STATUS – Status register

Bit	7	6	5	4	3	2	1	0
+0x04	CH3BUSY	CH2BUSY	CH1BUSY	CH0BUSY	CH3PEND	CH2PEND	CH1PEND	CH0PEND
Read/Write	R	R	R	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7:4 – CHnBUSY[3:0]: Channel Busy**
  When channel n starts a DMA transaction, the CHnBUSY flag will be read as one. This flag is automatically cleared when the DMA channel is disabled, when the channel n transaction complete interrupt flag is set, or if the DMA channel n error interrupt flag is set.
- **Bit 3:0 – CHnPEND[3:0]: Channel Pending**
  If a block transfer is pending on DMA channel n, the CHnPEND flag will be read as one. This flag is automatically cleared when the block transfer starts or if the transfer is aborted.

5.13.4 TEMPL – Temporary register Low

Bit	7	6	5	4	3	2	1	0
+0x06	TEMP[7:0]							
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 7:0 – TEMP[7:0]: Temporary register 0**
  This register is used when reading 16- and 24-bit registers in the DMA controller. Byte 1 of the 16/24-bit register is stored here when it is written by the CPU. Byte 1 of the 16/24-bit register is stored when byte 0 is read by the CPU. This register can also be read and written from the user software.
  Reading and writing 16- and 24-bit registers requires special attention. For details, refer to "The combined EIND + Z register." on page 12.

5.13.5 TEMPH – Temporary Register High

Bit	7	6	5	4	3	2	1	0
+0x07	TEMP[15:8]							
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

- **Bit 7:0 – TEMP[15:8]: Temporary Register**
  This register is used when reading and writing 24-bit registers in the DMA controller. Byte 2 of the 24-bit register is stored when it is written by the CPU. Byte 2 of the 24-bit register is stored here when byte 1 is read by the CPU. This register can also be read and written from the user software.
  Reading and writing 24-bit registers requires special attention. For details, refer to "The combined EIND + Z register." on page 12.

5.14 Register Description – DMA Channel

5.14.1 CTRLA – Control register A

Bit	7	6	5	4	3	2	1	0
+0x00	ENABLE	RESET	REPEAT	TRFREQ	–	SINGLE	BURSTLEN[1:0]	
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7 – ENABLE: Channel Enable**
  Setting this bit enables the DMA channel. This bit is automatically cleared when the transaction is completed. If the DMA channel is enabled and this bit is written to zero, the CHEN bit is not cleared until the internal transfer buffer is empty and the DMA transfer is aborted.
- **Bit 6 – RESET: Software Reset**
  Setting this bit will reset the DMA channel. It can only be set when the DMA channel is disabled (CHEN = 0). Writing a one to this bit will be ignored as long as the channel is enabled (CHEN=1). This bit is automatically cleared when reset is completed.
- **Bit 5 – REPEAT: Repeat Mode**
  Setting this bit enables the repeat mode. In repeat mode, this bit is cleared by hardware at the beginning of the last block transfer. The REPCNT register should be configured before setting the REPEAT bit.
- **Bit 4 – TRFREQ: Transfer Request**
  Setting this bit requests a data transfer on the DMA channel. This bit is automatically cleared at the beginning of the data transfer. Writing this bit does not have any effect unless the channel is enabled.
- **Bit 3 – Reserved**
  This bit is unused and reserved for future use. For compatibility with future devices, always write this bit to zero when this register is written.
- **Bit 2 – SINGLE: Single-Shot Data transfer**
  Setting this bit enables the single-shot mode. The channel will then do a burst transfer of BURSTLEN bytes on the transfer trigger. A write to this bit will be ignored while the channel is enabled.
- **Bit 1:0 – BURSTLEN[1:0]: Burst Mode**
  These bits decide the DMA channel burst mode according to Table 5-3 on page 60. These bits cannot be changed if the channel is busy.

Table 5-3. DMA channel burst mode.

BURSTLEN[1:0]	Group configuration	Description
00	1BYTE	1 byte burst mode
01	2BYTE	2 bytes burst mode
10	4BYTE	4 bytes burst mode
11	8BYTE	8 bytes burst mode

Table 5-4. Summary of triggers, transaction complete flag and channel disable according to DMA channel configuration.

REPEAT	SINGLE	REPCNT	Trigger	Flag set after	Channel disabled after
0	0	0	Block	1 block	1 block
0	0	1	Block	1 block	1 block
0	0	n > 1	Block	1 block	1 block
0	1	0	BURSTLEN	1 block	1 block
0	1	1	BURSTLEN	1 block	1 block
0	1	n > 1	BURSTLEN	1 block	1 block
1	0	0	Block	Each block	Each block
1	0	1	Transaction	1 block	1 block
1	0	n > 1	Transaction	n blocks	n blocks
1	1	0	BURSTLEN	Each block	Never
1	1	1	BURSTLEN	1 block	1 block
1	1	n > 1	BURSTLEN	n blocks	n blocks

5.14.2 CTRLB – Control register B

Bit	7	6	5	4	3	2	1	0
+0x01	CHBUSY	CHPEND	ERRIF	TRNIF	ERRINTLVL[1:0]		TRNINTLVL[1:0]	
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

- **Bit 7 – CHBUSY: Channel Busy**
  When the DMA channel starts a DMA transaction, the CHBUSY flag will be read as one. This flag is automatically cleared when the DMA channel is disabled, when the channel transaction complete interrupt flag is set or when the channel error interrupt flag is set.
- **Bit 6 – CHPEND: Channel Pending**
  If a block transfer is pending on the DMA channel, the CHPEND flag will be read as one. This flag is automatically cleared when the transfer starts or if the transfer is aborted.
- **Bit 5 – ERRIF: Error Interrupt Flag**
  If an error condition is detected on the DMA channel, the ERRIF flag will be set and the optional interrupt is generated. Since the DMA channel error interrupt shares the interrupt address with the DMA channel n transaction

complete interrupt, ERRIF will not be cleared when the interrupt vector is executed. This flag is cleared by writing a one to this location.

- **Bit 4 – TRNIF: Channel n Transaction Complete Interrupt Flag**
  When a transaction on the DMA channel has been completed, the TRNIF flag will be set and the optional interrupt is generated. When repeat is not enabled, the transaction is complete and TRNIFR is set after the block transfer. When unlimited repeat is enabled, TRNIF is also set after each block transfer.
  Since the DMA channel transaction n complete interrupt shares the interrupt address with the DMA channel error interrupt, TRNIF will not be cleared when the interrupt vector is executed. This flag is cleared by writing a one to this location.

- **Bit 3:2 – ERRINTLVL[1:0]: Channel Error Interrupt Level**
  These bits enable the interrupt for DMA channel transfer errors and select the interrupt level, as described in "Interrupts and Programmable Multilevel Interrupt Controller" on page 131. The enabled interrupt will trigger for the conditions when ERRIF is set.

- **Bit 1:0 – TRNINTLVL[1:0]: Channel Transaction Complete Interrupt Level**
  These bits enable the interrupt for DMA channel transaction completes and select the interrupt level, as described in "Interrupts and Programmable Multilevel Interrupt Controller" on page 131. The enabled interrupt will trigger for the conditions when TRNIF is set.

5.14.3 ADDRCTRL – Address Control register

Bit	7	6	5	4	3	2	1	0
+0x02	SRCRELOAD[1:0]		SRCDIR[1:0]		DESTRELOAD[1:0]		DESTDIR[1:0]	
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 7:6 – SRCRELOAD[1:0]: Channel Source Address Reload**
  These bits decide the DMA channel source address reload according to Table 5-5 on page 61. A write to these bits is ignored while the channel is busy.

Table 5-5. DMA channel source address reload settings.

SRCRELOAD[1:0]	Group configuration	Description
00	NONE	No reload performed.
01	BLOCK	DMA source address register is reloaded with initial value at end of each block transfer.
10	BURST	DMA source address register is reloaded with initial value at end of each burst transfer.
11	TRANSACTION	DMA source address register is reloaded with initial value at end of each transaction.

- **Bit 5:4 – SRCDIR[1:0]: Channel Source Address Mode**
  These bits decide the DMA channel source address mode according to Table 5-6. These bits cannot be changed if the channel is busy.

Table 5-6. DMA channel source address mode settings.

SRCDIR[1:0]	Group configuration	Description
00	FIXED	Fixed
01	INC	Increment
10	DEC	Decrement
11	–	Reserved

- **Bit 3:2 – DESTRELOAD[1:0]: Channel Destination Address Reload**
  These bits decide the DMA channel destination address reload according to Table 5-7 on page 62. These bits cannot be changed if the channel is busy.

Table 5-7. DMA channel destination address reload settings.

DESTRELOAD[1:0]	Group configuration	Description
00	NONE	No reload performed.
01	BLOCK	DMA channel destination address register is reloaded with initial value at end of each block transfer.
10	BURST	DMA channel destination address register is reloaded with initial value at end of each burst transfer.
11	TRANSACTION	DMA channel destination address register is reloaded with initial value at end of each transaction.

- **Bit 1:0 – DESTDIR[1:0]: Channel Destination Address Mode**
  These bits decide the DMA channel destination address mode according to Table 5-8 on page 62. These bits cannot be changed if the channel is busy.

Table 5-8. DMA channel destination address mode settings.

DESTDIR[1:0]	Group configuration	Description
00	FIXED	Fixed
01	INC	Increment
10	DEC	Decrement
11	–	Reserved

5.14.4 TRIGSRC – Trigger Source

Bit	7	6	5	4	3	2	1	0
+0x03	TRIGSRC[7:0]							
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 7:0 – TRIGSRC[7:0]: Channel Trigger Source Select**
  These bits select which trigger source is used for triggering a transfer on the DMA channel. A zero value means that the trigger source is disabled. For each trigger source, the value to put in the TRIGSRC register is the sum of the module's or peripheral's base value and the offset value for the trigger source in the module or peripheral.

Table 5-9 on page 63 shows the base value for all modules and peripherals. Table 5-10 on page 64 to Table 5-13 on page 64 shows the offset value for the trigger sources in the different modules and peripheral types. For modules or peripherals which do not exist for a device, the transfer trigger does not exist. Refer to the device datasheet for the list of peripherals available.

If the interrupt flag related to the trigger source is cleared or the interrupt level enabled so that an interrupt is triggered, the DMA request will be lost. Since a DMA request can clear the interrupt flag, interrupts can be lost.

Note: For most trigger sources, the request is cleared by accessing a register belonging to the peripheral with the request. Refer to the different peripheral chapters for how requests are generated and cleared.

Table 5-9. DMA trigger source base values for all modules and peripherals.

TRIGSRC base value	Group configuration	Description
0x00	OFF	Software triggers only
0x01	SYS	Event system DMA triggers base value
0x04	AES	AES DMA trigger value
0x10	ADCA	ADCA DMA triggers base value
0x15	DACA	DACA DMA trigger bas
0x20	ADCB	ADCB DMA triggers base value
0x25	DACB	DACB DMA triggers base value
0x40	TCC0	Timer/counter C0 DMA triggers base value
0x46	TCC1	Timer/counter C1 triggers base value
0x4A	SPIC	SPI C DMA triggers value
0x4B	USARTC0	USART C0 DMA triggers base value
0x4E	USARTC1	USART C1 DMA triggers base value
0x60	TCD0	Timer/counter D0 DMA triggers base value
0x66	TCD1	Timer/counter D1 triggers base value
0x6A	SPID	SPI D DMA triggers value
0x6B	USARTD0	USART D0 DMA triggers base value
0x6E	USARTD1	USART D1 DMA triggers base value
0x80	TCE0	Timer/counter E0 DMA triggers base value
0x86	TCE1	Timer/counter E1 triggers base value
0x8A	SPIE	SPI E DMA triggers value
0x8B	USARTE0	USART E0 DMA triggers base value
0x8E	USARTE1	USART E1 DMA triggers base value
0xA0	TCF0	Timer/counter F0 DMA triggers base value
0xA6	TCF1	Timer/counter F1 triggers base value
0xAA	SPIF	SPI F DMA trigger value
0xAB	USARTF0	USART F0 DMA triggers base value
0xAE	USARTF1	USART F1 DMA triggers base value

Table 5-10. DMA trigger source offset values for event system triggers.

TRGSRC offset value	Group configuration	Description
+0x00	CH0	Event channel 0
+0x01	CH1	Event channel 1
+0x02	CH2	Event channel 2

Table 5-11. DMA trigger source offset values for DAC and ADC triggers.

TRGSRC offset value	Group configuration	Description
+0x00	CH0	ADC/DAC channel 0
+0x01	CH1	ADC/DAC channel 1
+0x02	CH2(1)	ADC channel 2
+0x03	CH3	ADC channel 3
+0x04	CH4(2)	ADC channel 0, 1, 2, 3

Notes: 1. For DAC only, channel 0 and 1 exists and can be used as triggers.
2. Channel 4 equals ADC channel 0 to 3 all together.

Table 5-12. DMA trigger source offset values for timer/ counter triggers.

TRGSRC offset value	Group configuration	Description
+0x00	OVF	Overflow/underflow
+0x01	ERR	Error
+0x02	CCA	Compare or capture channel A
+0x03	CCB	Compare or capture channel B
+0x04	CCC(1)	Compare or capture channel C
+0x05	CCD(1)	Compare or capture channel D

Note: 1. CC channel C and D triggers are available only for timer/counters 0.

Table 5-13. DMA trigger source offset values for USART triggers.

TRGSRC offset value	Group configuration	Description
0x00	RXC	Receive complete
0x01	DRE	Data register empty

The group configuration is the “base_offset;” for example, TCC1_CCA for the timer/counter C1 CC channel A the transfer trigger.

5.14.5 TRFCNTL – Channel Block Transfer Count register Low

The TRFCNTH and TRFCNTL register pair represents the 16-bit value TRFCNT. TRFCNT defines the number of bytes in a block transfer. The value of TRFCNT is decremented after each byte read by the DMA channel. When TRFCNT reaches zero, the register is reloaded with the last value written to it.

Bit	7	6	5	4	3	2	1	0
+0x04	TRFCNT[7:0]							
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 7:0 – TRFCNT[7:0]: Channel n Block Transfer Count low byte**
  These bits hold the LSB of the 16-bit block transfer count.
  The default value of this register is 0x1. If a user writes 0x0 to this register and fires a DMA trigger, DMA will be doing 0xFFFF transfers.

5.14.6 TRFCNTH – Channel Block Transfer Count register High

Reading and writing 16-bit values requires special attention. For details, refer to "The combined EIND + Z register." on page 12.

Bit	7	6	5	4	3	2	1	0
+0x05	TRFCNT[15:8]							
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

- **Bit 7:0 – TRFCNT[15:8]: Channel n Block Transfer Count high byte**
  These bits hold the MSB of the 16-bit block transfer count.
  The default value of this register is 0x1. If a user writes 0x0 to this register and fires a DMA trigger, DMA will be doing 0xFFFF transfers.

5.14.7 REPCNT – Repeat Counter register

Bit	7	6	5	4	3	2	1	0
+0x06	REPCNT[7:0]							
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

REPCNT counts how many times a block transfer is performed. For each block transfer, this register will be decremented.

When repeat mode is enabled (see REPEAT bit in "ADDRCTRL – Address Control register" on page 61), this register is used to control when the transaction is complete. The counter is decremented after each block transfer if the DMA has to serve a limited number of repeated block transfers. When repeat mode is enabled, the channel is disabled when REPCNT reaches zero and the last block transfer is completed. Unlimited repeat is achieved by setting this register to zero.

5.14.8 SRCADDR0 – Source Address 0

SRCADDR0, SRCADDR1, and SRCADDR2 represent the 24-bit value SRCADDR, which is the DMA channel source address. SRCADDR2 is the most significant byte in the register. SRCADDR may be automatically incremented or decremented based on settings in the SRCDIR bits in "ADDRCTRL – Address Control register" on page 61.

Bit	7	6	5	4	3	2	1	0
+0x08	SRCADDR[7:0]							
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 7:0 – SRCADDR[7:0]: Channel Source Address Byte 0**
  These bits hold byte 0 of the 24-bit source address.

5.14.9 SRCADDR1 – Channel Source Address 1

Bit	7	6	5	4	3	2	1	0
+0x09	SRCADDR[15:8]							
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

- **Bit 7:0 – SRCADDR[15:8]: Channel Source Address Byte 1**
  These bits hold byte 1 of the 24-bit source address.

5.14.10 SRCADDR2 – Channel Source Address 2

Reading and writing 24-bit values require special attention. For details, refer to "Accessing 24- and 32-bit Registers" on page 13.

Bit	7	6	5	4	3	2	1	0
+0x0A	SRCADDR[23:16]							
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 7:0 – SRCADDR[23:16]: Channel Source Address Byte 2**
  These bits hold byte 2 of the 24-bit source address.

5.14.11 DESTADDR0 – Channel Destination Address 0

DESTADDR0, DESTADDR1, and DESTADDR2 represent the 24-bit value DESTADDR, which is the DMA channel destination address. DESTADDR2 holds the most significant byte in the register. DESTADDR may be automatically incremented or decremented based on settings in the DESTDIR bits in "ADDRCTRL – Address Control register" on page 61.

Bit	7	6	5	4	3	2	1	0
+0x0C	DESTADDR[7:0]							
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 7:0 – DESTADDR[7:0]: Channel Destination Address Byte 0**
  These bits hold byte 0 of the 24-bit source address.

5.14.12 DESTADDR1 – Channel Destination Address 1

Bit	7	6	5	4	3	2	1	0
+0x0D	DESTADDR[15:8]							
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

- **Bit 7:0 – DESTADDR[15:8]: Channel Destination Address Byte 1**
  These bits hold byte 1 of the 24-bit source address.

5.14.13 DESTADDR2 – Channel Destination Address 2

Reading and writing 24-bit values require special attention. For details, refer to "Accessing 24- and 32-bit Registers" on page 13.

Bit	7	6	5	4	3	2	1	0
+0x0E	DESTADDR[23:16]							
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 7:0 – DESTADDR[23:16]: Channel Destination Address Byte 2**
  These bits hold byte 2 of the 24-bit source address.

5.15 Register Summary – DMA Controller

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	CTRL	ENABLE	RESET	–	–	DBUFMODE[1:0]		PRIMODE[1:0]		57
+0x01	Reserved	–	–	–	–	–	–	–	–	
+0x02	Reserved	–	–	–	–	–	–	–	–	
+0x03	INTFLAGS	CH3ERRIF	CH2ERRIF	CH1ERRIF	CH0ERRIF	CH3TRNFIF	CH2TRNFIF	CH1TRNFIF	CH0TRNFIF	58
+0x04	STATUS	CH3BUSY	CH2BUSY	CH1BUSY	CH0BUSY	CH3PEND	CH2PEND	CH1PEND	CH0PEND	58
+0x05	Reserved	–	–	–	–	–	–	–	–	
+0x06	TEMPL	TEMP[7:0]								58
+0x07	TEMPH	TEMP[15:8]								59
+0x10	CH0 Offset	Offset address for DMA Channel 0								
+0x20	CH1 Offset	Offset address for DMA Channel 1								
+0x30	CH2 Offset	Offset address for DMA Channel 2								
+0x40	CH3 Offset	Offset address for DMA Channel 3								

5.16 Register Summary – DMA Channel

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	CTRLA	ENABLE	RESET	REPEAT	TRFREQ	–	SINGLE	BURSTLEN[1:0]		59
+0x01	CTRLB	CHBUSY	CHPEND	ERRIF	TRNIF	ERRINTLVL[1:0]		TRNINTLVL[1:0]		60
+0x02	ADDCTRL	SRCRELOAD[1:0]		SRCDIR[1:0]		DESTRELOAD[1:0]		DESTDIR[1:0]		61
+0x03	TRIGSRC	TRIGSRC[7:0]								62
+0x04	TRFCNTL	TRFCNT[7:0]								65
+0x05	TRFCNTH	TRFCNT[15:8]								65
+0x06	REPCNT	REPCNT[7:0]								65
+0x07	Reserved	–	–	–	–	–	–	–	–	
+0x08	SRCADDR0	SRCADDR[7:0]								65
+0x09	SRCADDR1	SRCADDR[15:8]								66
+0x0A	SRCADDR2	SRCADDR[23:16]								66
+0x0B	Reserved	–	–	–	–	–	–	–	–	
+0x0C	DESTADDR0	DESTADDR[7:0]								66
+0x0D	DESTADDR1	DESTADDR[15:8]								66
+0x0E	DESTADDR2	DESTADDR[23:16]								67
+0x0F	Reserved	–	–	–	–	–	–	–	–	

5.17 Interrupt vector summary

Table 5-14. DMA interrupt vectors and their word offset addresses from the DMA controller interrupt base.

Offset	Source	Interrupt description
0x00	CH0_vect	DMA controller channel 0 interrupt vector
0x02	CH1_vect	DMA controller channel 1 interrupt vector
0x04	CH2_vect	DMA controller channel 2 interrupt vector
0x06	CH3_vect	DMA controller channel 3 interrupt vector

6. Event System

6.1 Features

- System for direct peripheral-to-peripheral communication and signaling
- Peripherals can directly send, receive, and react to peripheral events
  - CPU and DMA controller independent operation
  - 100% predictable signal timing
  - Short and guaranteed response time
- Eight event channels for up to eight different and parallel signal routing and configurations
- Events can be sent and/or used by most peripherals, clock system, and software
- Additional functions include
  - Quadrature decoders
  - Digital filtering of I/O pin state
- Works in active mode and idle sleep mode

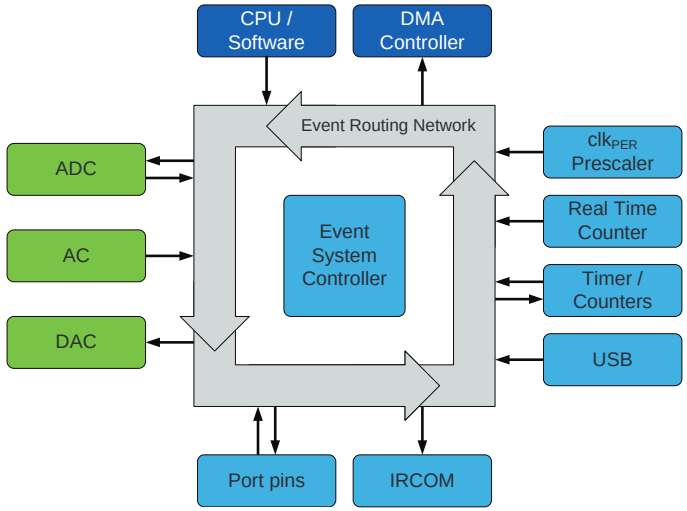
6.2 Overview

The event system enables direct peripheral-to-peripheral communication and signaling. It allows a change in one peripheral's state to automatically trigger actions in other peripherals. It is designed to provide a predictable system for short and predictable response times between peripherals. It allows for autonomous peripheral control and interaction without the use of interrupts, CPU, or DMA controller resources, and is thus a powerful tool for reducing the complexity, size and execution time of application code. It also allows for synchronized timing of actions in several peripheral modules.

A change in a peripheral's state is referred to as an event, and usually corresponds to the peripheral's interrupt conditions. Events can be directly passed to other peripherals using a dedicated routing network called the event routing network. How events are routed and used by the peripherals is configured in software.

Figure 6-1 on page 71 shows a basic diagram of all connected peripherals. The event system can directly connect together analog and digital converters, analog comparators, I/O port pins, the real-time counter, timer/counters, IR communication module (IRCOM), and USB interface. It can also be used to trigger DMA transactions (DMA controller). Events can also be generated from software and the peripheral clock.

Figure 6-1. Event system overview and connected peripherals.



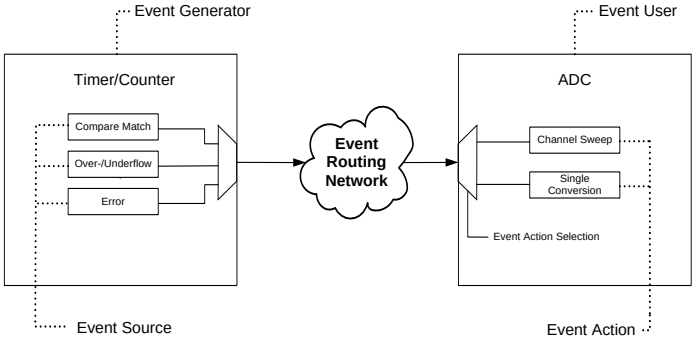
The event routing network consists of eight software-configurable multiplexers that control how events are routed and used. These are called event channels, and allow for up to eight parallel event configurations and routings. The maximum routing latency is two peripheral clock cycles. The event system works in both active mode and idle sleep mode.

6.3 Events

In the context of the event system, an indication that a change of state within a peripheral has occurred is called an event. There are two main types of events: signaling events and data events. Signaling events only indicate a change of state while data events contain additional information about the event.

The peripheral from which the event originates is called the event generator. Within each peripheral (for example, a timer/counter), there can be several event sources, such as a timer compare match or timer overflow. The peripheral using the event is called the event user, and the action that is triggered is called the event action.

Figure 6-2. Example of event source, generator, user, and action.



Events can also be generated manually in software.

6.3.1 Signaling Events

Signaling events are the most basic type of event. A signaling event does not contain any information apart from the indication of a change in a peripheral. Most peripherals can only generate and use signaling events. Unless otherwise stated, all occurrences of the word ”event” are to be understood as meaning signaling events.

6.3.2 Data Events

Data events differ from signaling events in that they contain information that event users can decode to decide event actions based on the receiver information.

Although the event routing network can route all events to all event users, those that are only meant to use signaling events do not have decoding capabilities needed to utilize data events. How event users decode data events is shown in Table 6-1 on page 73.

Event users that can utilize data events can also use signaling events. This is configurable, and is described in the datasheet module for each peripheral.

6.3.3 Peripheral Clock Events

Each event channel includes a peripheral clock prescaler with a range from 1 (no prescaling) to 32768. This enables configurable periodic event generation based on the peripheral clock. It is possible to periodically trigger events in a peripheral or to periodically trigger synchronized events in several peripherals. Since each event channel include a prescaler, different peripherals can receive triggers with different intervals.

6.3.4 Software Events

Events can be generated from software by writing the DATA and STROBE registers. The DATA register must be written first, since writing the STROBE register triggers the operation. The DATA and STROBE registers contain one bit for each event channel. Bit n corresponds to event channel n. It is possible to generate events on several channels at the same time by writing to several bit locations at once.

Software-generated events last for one clock cycle and will overwrite events from other event generators on that event channel during that clock cycle.

Table 6-1 on page 73 shows the different events, how they can be manually generated, and how they are decoded.

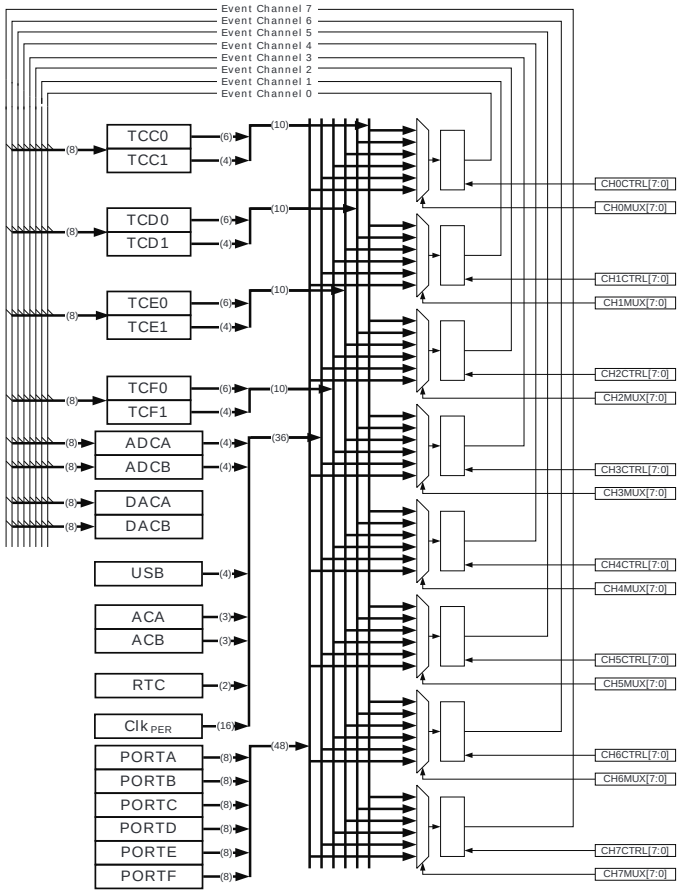
Table 6-1. Quadrature decoder data events.

STROBE	DATA	Data event user	Signaling event user
0	0	No event	No event
0	1	Data event 01	No event
1	0	Data event 02	Signaling event
1	1	Data event 03	Signaling event

6.4 Event Routing Network

The event routing network routes the events between peripherals. It consists of eight multiplexers (CHnMUX), which can each be configured to route any event source to any event users. The output from a multiplexer is referred to as an event channel. For each peripheral, it is selectable if and how incoming events should trigger event actions. Details on configurations can be found in the datasheet for each peripheral. The event routing network is shown in Figure 6-3 on page 74.

Figure 6-3. Event routing network.



Eight multiplexers means that it is possible to route up to eight events at the same time. It is also possible to route one event through several multiplexers.

Not all XMEGA devices contain all peripherals. This only means that a peripheral is not available for generating or using events. The network configuration itself is compatible between all devices.

6.5 Event Timing

An event normally lasts for one peripheral clock cycle, but some event sources, such as a low level on an I/O pin, will generate events continuously. Details on this are described in the datasheet for each peripheral, but unless otherwise stated, an event lasts for one peripheral clock cycle.

It takes a maximum of two peripheral clock cycles from when an event is generated until the event actions in other peripherals are triggered. This ensures short and 100% predictable response times, independent of CPU or DMA controller load or software revisions.

6.6 Filtering

Each event channel includes a digital filter. When this is enabled, an event must be sampled with the same value for a configurable number of system clock cycles before it is accepted. This is primarily intended for pin change events.

6.7 Quadrature Decoder

The event system includes three quadrature decoders (QDECs), which enable the device to decode quadrature input on I/O pins and send data events that a timer/counter can decode to count up, count down, or index/reset. Table 6-2 on page 75 summarizes which quadrature decoder data events are available, how they are decoded, and how they can be generated. The QDECs and related features and control and status registers are available for event channels 0, 2, and 4.

Table 6-2. Quadrature decoder data events.

STROBE	DATA	Data event user	Signaling event user
0	0	No event	No event
0	1	Index/reset	No event
1	0	Count down	Signaling event
1	1	Count up	Signaling event

6.7.1 Quadrature Operation

A quadrature signal is characterized by having two square waves that are phase shifted 90 degrees relative to each other. Rotational movement can be measured by counting the edges of the two waveforms. The phase relationship between the two square waves determines the direction of rotation.

Figure 6-4. Quadrature signals from a rotary encoder.

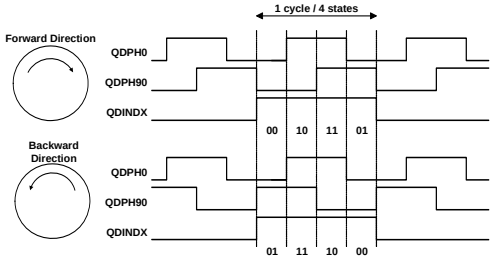


Figure 6-4 shows typical quadrature signals from a rotary encoder. The signals QDPH0 and QDPH90 are the two quadrature signals. When QDPH90 leads QDPH0, the rotation is defined as positive or forward. When QDPH0 leads

QDPH90, the rotation is defined as negative or reverse. The concatenation of the two phase signals is called the quadrature state or the phase state.

In order to know the absolute rotary displacement, a third index signal (QINDX) can be used. This gives an indication once per revolution.

6.7.2 QDEC Setup

For a full QDEC setup, the following is required:

- Two or three I/O port pins for quadrature signal input
- Two event system channels for quadrature decoding
- One timer/counter for up, down, and optional index count

The following procedure should be used for QDEC setup:

1. Choose two successive pins on a port as QDEC phase inputs.
2. Set the pin direction for QDPH0 and QDPH90 as input.
3. Set the pin configuration for QDPH0 and QDPH90 to low level sense.
4. Select the QDPH0 pin as a multiplexer input for an event channel, n.
5. Enable quadrature decoding and digital filtering in the event channel.
6. Optional:
    1. Set up a QDEC index (QINDX).
    2. Select a third pin for QINDX input.
    3. Set the pin direction for QINDX as input.
    4. Set the pin configuration for QINDX to sense both edges.
    5. Select QINDX as a multiplexer input for event channel n+1
    6. Set the quadrature index enable bit in event channel n.
    7. Select the index recognition mode for event channel n.
7. Set quadrature decoding as the event action for a timer/counter.
8. Select event channel n as the event source for the timer/counter.
- Set the period register of the timer/counter to ('line count' * 4 - 1), the line count of the quadrature encoder.
- Enable the timer/counter without clock prescaling.

The angle of a quadrature encoder attached to QDPH0, QDPH90 (and QINDX) can now be read directly from the timer/counter count register. If the count register is different from BOTTOM when the index is recognized, the timer/counter error flag is set. Similarly, the error flag is set if the position counter passes BOTTOM without the recognition of the index.

6.8 Register Description

6.8.1 CHnMUX – Event Channel n Multiplexer register

Bit	7	6	5	4	3	2	1	0
	CHnMUX[7:0]							
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 7:0 – CHnMUX[7:0]: Channel Multiplexer
These bits select the event source according to Table 6-3. This table is valid for all XMEGA devices regardless of whether the peripheral is present or not. Selecting event sources from peripherals that are not present will give the same result as when this register is zero. When this register is zero, no events are routed through. Manually generated events will override CHnMUX and be routed to the event channel even if this register is zero.

Table 6-3. CHnMUX[7:0] bit settings.

CHnMUX[7:4]	CHnMUX[3:0]				Group configuration	Event source
0000	0	0	0	0		None (manually generated events only)
0000	0	0	0	1		(Reserved)
0000	0	0	1	X		(Reserved)
0000	0	1	X	X		(Reserved)
0000	1	0	0	0	RTC_OVF/RTC32_OVF	RTC overflow / RTC32 overflow
0000	1	0	0	1	RTC_CMP	RTC compare match
0000	1	0	1	0		USB start of frame on CH0 (see Table 6-4 on page 78) USB error on CH1 (see Table 6-4 on page 78) USB overflow on CH2 (see Table 6-4 on page 78) USB setup on CH3 (see Table 6-4 on page 78)
0000	1	0	1	X		(Reserved)
0000	1	1	X	X		(Reserved)
0001	0	0	0	0	ACA_CH0	ACA channel 0
0001	0	0	0	1	ACA_CH1	ACA channel 1
0001	0	0	1	0	ACA_WIN	ACA window
0001	0	0	1	1	ACB_CH0	ACB channel 0
0001	0	1	0	0	ACB_CH1	ACB channel 1
0001	0	1	0	1	ACB_WIN	ACB window
0001	0	1	1	X		(Reserved)
0001	1	X	X	X		(Reserved)
0010	0	0	n		ADCA_CHn	ADCA channel n (n =0, 1, 2 or 3)
0010	0	1	n		ADCB_CHn	ADCB channel n (n=0, 1, 2 or 3)
0010	1	X	X	X		(Reserved)
0011	X	X	X	X		(Reserved)
0100	X	X	X	X		(Reserved)

CHnMUX[7:4]	CHnMUX[3:0]	Group configuration	Event source
0101	0 n	PORTA_PINn(1)	PORTA pin n (n= 0, 1, 2 ... or 7)
0101	1 n	PORTB_PINn(1)	PORTB pin n (n= 0, 1, 2 ... or 7)
0110	0 n	PORTC_PINn(1)	PORTC pin n (n= 0, 1, 2 ... or 7)
0110	1 n	PORTD_PINn(1)	PORTD pin n (n= 0, 1, 2 ... or 7)
0111	0 n	PORTE_PINn(1)	PORTE pin n (n= 0, 1, 2 ... or 7)
0111	1 n	PORTF_PINn(1)	PORTF pin n (n= 0, 1, 2 ... or 7)
1000	M	PRESCALER_M	Clk_{PER} divide by 2^M (M=0 to 15)
1001	X X X X		(Reserved)
1010	X X X X		(Reserved)
1011	X X X X		(Reserved)
1100	0 E	See Table 6-4	Timer/counter C0 event type E
1100	1 E	See Table 6-4	Timer/counter C1 event type E
1101	0 E	See Table 6-4	Timer/counter D0 event type E
1101	1 E	See Table 6-4	Timer/counter D1 event type E
1110	0 E	See Table 6-4	Timer/counter E0 event type E
1110	1 E	See Table 6-4	Timer/counter E1 event type E
1111	0 E	See Table 6-4	Timer/counter F0 event type E
1111	1 E	See Table 6-4	Timer/counter F1 event type E

Notes: 1. The description of how the ports generate events is described in "Port Event" on page 145.
2. The different USB events can be selected for only event channel, 0 to 3.

Table 6-4. Timer/counter events.

T/C event E			Group configuration	Event type
0	0	0	TCxn_OVF	Over/Underflow (x = C, D, E or F) (n= 0 or 1)
0	0	1	TCxn_ERR	Error (x = C, D, E or F) (n= 0 or 1)
0	1	X		(Reserved)
1	0	0	TCxn_CCA	Capture or compare A (x = C, D, E or F) (n= 0 or 1)
1	0	1	TCxn_CCB	Capture or compare B (x = C, D, E or F) (n= 0 or 1)
1	1	0	TCxn_CCC	Capture or compare C (x = C, D, E or F) (n= 0)
1	1	1	TCxn_CCD	Capture or compare D (x = C, D, E or F) (n= 0)

6.8.2 CHnCTRL – Event Channel n Control register

Bit	7	6	5	4	3	2	1	0
	–	QDIRM[1:0]		QDIEN	QDEN	DIGFILT[2:0]		
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – Reserved
This bit is reserved and will always be read as zero. For compatibility with future devices, always write this bit to zero when this register is written.

Bit 6:5 – QDIRM[1:0]: Quadrature Decode Index Recognition Mode
These bits determine the quadrature state for the QDPH0 and QDPH90 signals, where a valid index signal is recognized and the counter index data event is given according to Table 6-5 on page 79. These bits should only be set when a quadrature encoder with a connected index signal is used.These bits are available only for CH0CTRL, CH2CTRL, and CH4CTRL.

Table 6-5. QDIRM bit settings..

QDIRM[1:0]		Index recognition state
0	0	{QDPH0, QDPH90} = 0b00
0	1	{QDPH0, QDPH90} = 0b01
1	0	{QDPH0, QDPH90} = 0b10
1	1	{QDPH0, QDPH90} = 0b11

Bit 4 – QDIEN: Quadrature Decode Index Enable
When this bit is set, the event channel will be used as a QDEC index source, and the index data event will be enabled.
This bit is available only for CH0CTRL, CH2CTRL, and CH4CTRL.

Bit 3 – QDEN: Quadrature Decode Enable
Setting this bit enables QDEC operation.
This bit is available only for CH0CTRL, CH2CTRL, and CH4CTRL.

Bit 2:0 – DIGFILT[2:0]: Digital Filter Coefficient
These bits define the length of digital filtering used. Events will be passed through to the event channel only when the event source has been active and sampled with the same level for the number of peripheral clock cycles defined by DIGFILT.

Table 6-6. Digital filter coefficient values .

DIGFILT[2:0]	Group configuration	Description
000	1SAMPLE	One sample
001	2SAMPLES	Two samples
010	3SAMPLES	Three samples
011	4SAMPLES	Four samples
100	5SAMPLES	Five samples
101	6SAMPLES	Six samples
110	7SAMPLES	Seven samples
111	8SAMPLES	Eight samples

6.8.3 STROBE – Strobe register

If the STROBE register location is written, each event channel will be set according to the STROBE[n] and corresponding DATA[n] bit settings, if any are unequal to zero.

A single event lasting for one peripheral clock cycle will be generated.

Bit	7	6	5	4	3	2	1	0
+0x10	STROBE[7:0]							
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

6.8.4 DATA – Data register

This register contains the data value when manually generating a data event. This register must be written before the STROBE register. For details, See "STROBE – Strobe register" on page 80.

Bit	7	6	5	4	3	2	1	0
+0x11	DATA[7:0]							
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

6.9 Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	CH0MUX	CH0MUX[7:0]								77
+0x01	CH1MUX	CH1MUX[7:0]								77
+0x02	CH2MUX	CH2MUX[7:0]								77
+0x03	CH3MUX	CH3MUX[7:0]								77
+0x04	CH4MUX	CH4MUX[7:0]								77
+0x05	CH5MUX	CH5MUX[7:0]								77
+0x06	CH6MUX	CH6MUX[7:0]								77
+0x07	CH7MUX	CH7MUX[7:0]								77
+0x08	CH0CTRL	–	QDIRM[1:0]		QDIEN	QDEN	DIGFILT[2:0]			78
+0x09	CH1CTRL	–	–	–	–	–	DIGFILT[2:0]			78
+0x0A	CH2CTRL	–	QDIRM[1:0]		QDIEN	QDEN	DIGFILT[2:0]			78
+0x0B	CH3CTRL	–	–	–	–	–	DIGFILT[2:0]			78
+0x0C	CH4CTRL	–	QDIRM[1:0]		QDIEN	QDEN	DIGFILT[2:0]			78
+0x0D	CH5CTRL	–	–	–	–	–	DIGFILT[2:0]			78
+0x0E	CH6CTRL	–	–	–	–	–	DIGFILT[2:0]			78
+0x0F	CH7CTRL	–	–	–	–	–	DIGFILT[2:0]			78
+0x10	STROBE	STROBE[7:0]								80
+0x11	DATA	DATA[7:0]								80

7. System Clock and Clock Options

7.1 Features

- Fast start-up time
- Safe run-time clock switching
- Internal oscillators:
  - 32MHz run-time calibrated oscillator
  - 2MHz run-time calibrated oscillator
  - 32.768kHz calibrated oscillator
  - 32kHz ultra low power (ULP) oscillator with 1kHz output
- External clock options
  - 0.4MHz - 16MHz crystal oscillator
  - 32.768kHz crystal oscillator
  - External clock
- PLL with 20MHz - 128MHz output frequency
  - Internal and external clock options and 1x to 31x multiplication
  - Lock detector
- Clock prescalers with 1x to 2048x division
- Fast peripheral clocks running at 2 and 4 times the CPU clock
- Automatic run-time calibration of internal oscillators
- External oscillator and PLL lock failure detection with optional non-maskable interrupt

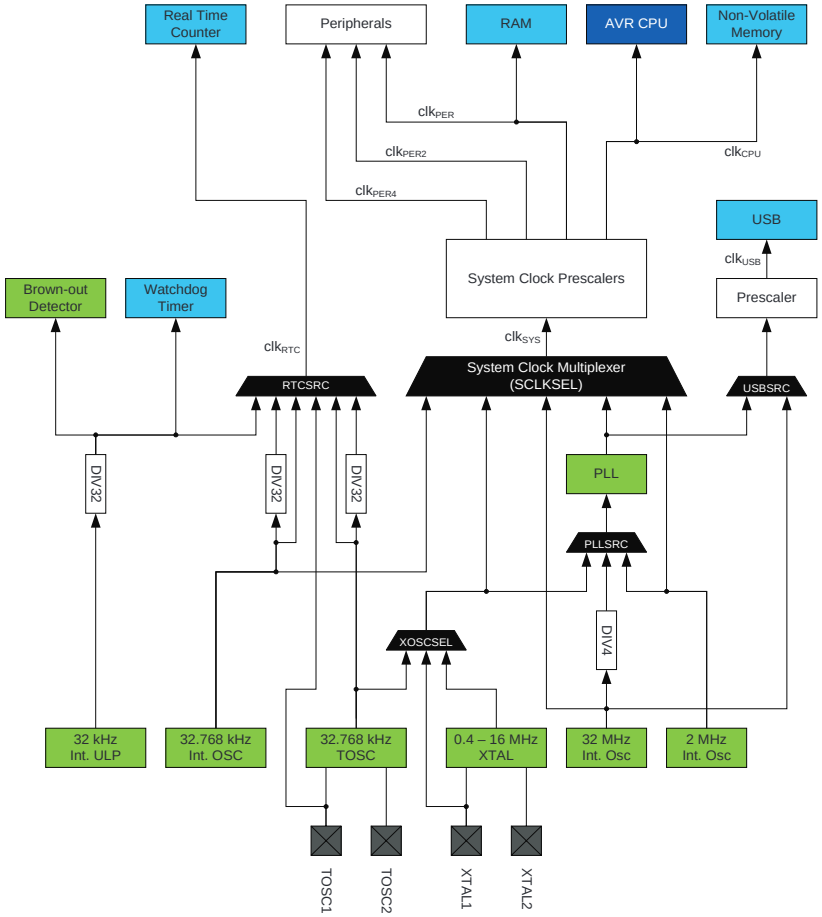
7.2 Overview

XMEGA devices have a flexible clock system supporting a large number of clock sources. It incorporates both accurate internal oscillators and external crystal oscillator and resonator support. A high-frequency phase locked loop (PLL) and clock prescalers can be used to generate a wide range of clock frequencies. A calibration feature (DFLL) is available, and can be used for automatic run-time calibration of the internal oscillators to remove frequency drift over voltage and temperature. An oscillator failure monitor can be enabled to issue a non-maskable interrupt and switch to the internal oscillator if the external oscillator or PLL fails.

When a reset occurs, all clock sources except the 32kHz ultra low power oscillator are disabled. After reset, the device will always start up running from the 2MHz internal oscillator. During normal operation, the system clock source and prescalers can be changed from software at any time.

Figure 7-1 on page 83 presents the principal clock system in the XMEGA family of devices. Not all of the clocks need to be active at a given time. The clocks for the CPU and peripherals can be stopped using sleep modes and power reduction registers, as described in "Power Management and Sleep Modes" on page 103.

Figure 7-1. The clock system, clock sources, and clock distribution.



7.3 Clock Distribution

Figure 7-1 on page 83 presents the principal clock distribution system used in XMEGA devices.

7.3.1 System Clock - Clk_{SYS}

The system clock is the output from the main system clock selection. This is fed into the prescalers that are used to generate all internal clocks except the asynchronous and USB clocks.

7.3.2 CPU Clock - Clk_{CPU}

The CPU clock is routed to the CPU and nonvolatile memory. Halting the CPU clock inhibits the CPU from executing instructions.

7.3.3 Peripheral Clock - Clk_{PER}

The majority of peripherals and system modules use the peripheral clock. This includes the DMA controller, event system, interrupt controller, external bus interface and RAM. This clock is always synchronous to the CPU clock, but may run even when the CPU clock is turned off.

7.3.4 Peripheral 2x/4x Clocks - Clk_{PER2}/Clk_{PER4}

Modules that can run at two or four times the CPU clock frequency can use the peripheral 2x and peripheral 4x clocks.

7.3.5 Asynchronous Clock - Clk_{RTC}

The asynchronous clock allows the real-time counter (RTC) to be clocked directly from an external 32.768kHz crystal oscillator or the 32 times prescaled output from the internal 32.768kHz oscillator or ULP oscillator. The dedicated clock domain allows operation of this peripheral even when the device is in sleep mode and the rest of the clocks are stopped.

7.3.6 USB Clock - Clk_{USB}

The USB device module requires a 12MHz or 48MHz clock. It has a separate clock source selection in order to avoid system clock source limitations when USB is used.

7.4 Clock Sources

The clock sources are divided in two main groups: internal oscillators and external clock sources. Most of the clock sources can be directly enabled and disabled from software, while others are automatically enabled or disabled, depending on peripheral settings. After reset, the device starts up running from the 2MHz internal oscillator. The other clock sources, DFLLs and PLL, are turned off by default.

7.4.1 Internal Oscillators

The internal oscillators do not require any external components to run. For details on characteristics and accuracy of the internal oscillators, refer to the device datasheet.

7.4.1.1 32kHz Ultra Low Power Oscillator

This oscillator provides an approximate 32kHz clock. The 32kHz ultra low power (ULP) internal oscillator is a very low power clock source, and it is not designed for high accuracy.The oscillator employs a built-in prescaler that provides a 1kHz output. The oscillator is automatically enabled/disabled when it is used as clock source for any part of the device. This oscillator can be selected as the clock source for the RTC.

7.4.1.2 32.768kHz Calibrated Oscillator

This oscillator provides an approximate 32.768kHz clock. It is calibrated during production to provide a default frequency close to its nominal frequency. The calibration register can also be written from software for run-time calibration of the oscillator frequency. The oscillator employs a built-in prescaler, which provides both a 32.768kHz output and a 1.024kHz output.

7.4.1.3 32MHz Run-time Calibrated Oscillator

The 32MHz run-time calibrated internal oscillator is a high-frequency oscillator. It is calibrated during production to provide a default frequency close to its nominal frequency. A digital frequency looked loop (DFLL) can be enabled for automatic run-time calibration of the oscillator to compensate for temperature and voltage drift and optimize the oscillator accuracy. This oscillator can also be adjusted and calibrated to any frequency between 30MHz and 55MHz. The production signature row contains 48 MHz calibration values intended used when the oscillator is used a full-speed USB clock source.

7.4.1.4 2MHz Run-time Calibrated Oscillator

The 2MHz run-time calibrated internal oscillator is the default system clock source after reset. It is calibrated during production to provide a default frequency close to its nominal frequency. A DFLL can be enabled for automatic run-time calibration of the oscillator to compensate for temperature and voltage drift and optimize the oscillator accuracy.

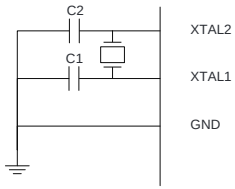
7.4.2 External Clock Sources

The XTAL1 and XTAL2 pins can be used to drive an external oscillator, either a quartz crystal or a ceramic resonator. XTAL1 can be used as input for an external clock signal. The TOSC1 and TOSC2 pins is dedicated to driving a 32.768kHz crystal oscillator.

7.4.2.1 0.4MHz - 16MHz Crystal Oscillator

This oscillator can operate in four different modes optimized for different frequency ranges, all within 0.4MHz - 16MHz. Figure 7-2 shows a typical connection of a crystal oscillator or resonator.

Figure 7-2. Crystal oscillator connection.

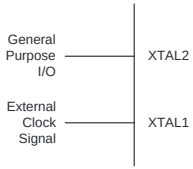


Two capacitors, C1 and C2, may be added to match the required load capacitance for the connected crystal.

7.4.2.2 External Clock Input

To drive the device from an external clock source, XTAL1 must be driven as shown in Figure 7-3 on page 85. In this mode, XTAL2 can be used as a general I/O pin.

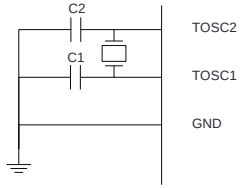
Figure 7-3. External clock drive configuration.



7.4.2.3 32.768kHz Crystal Oscillator

A 32.768kHz crystal oscillator can be connected between the TOSC1 and TOSC2 pins and enables a dedicated low frequency oscillator input circuit. A typical connection is shown in Figure 7-4 on page 86. A low power mode with reduced voltage swing on TOSC2 is available. This oscillator can be used as a clock source for the system clock and RTC, and as the DFLL reference clock.

Figure 7-4. 32.768kHz crystal oscillator connection.



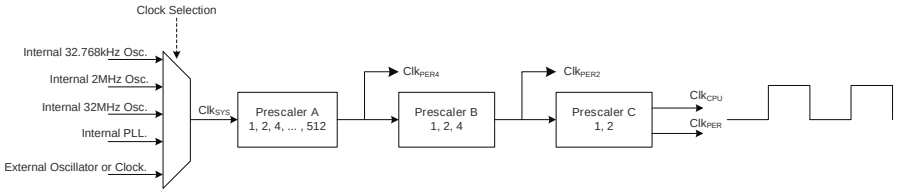
Two capacitors, C1 and C2, may be added to match the required load capacitance for the connected crystal. For details on recommended TOSC characteristics and capacitor load, refer to device datasheets.

7.5 System Clock Selection and Prescalers

All the calibrated internal oscillators, the external clock sources (XOSC), and the PLL output can be used as the system clock source. The system clock source is selectable from software, and can be changed during normal operation. Built-in hardware protection prevents unsafe clock switching. It is not possible to select a non-stable or disabled oscillator as the clock source, or to disable the oscillator currently used as the system clock source. Each oscillator option has a status flag that can be read from software to check that the oscillator is ready.

The system clock is fed into a prescaler block that can divide the clock signal by a factor from 1 to 2048 before it is routed to the CPU and peripherals. The prescaler settings can be changed from software during normal operation. The first stage, prescaler A, can divide by a factor of from 1 to 512. Then, prescalers B and C can be individually configured to either pass the clock through or combine divide it by a factor from 1 to 4. The prescaler guarantees that derived clocks are always in phase, and that no glitches or intermediate frequencies occur when changing the prescaler setting. The prescaler settings are updated in accordance with the rising edge of the slowest clock.

Figure 7-5. System clock selection and prescalers.



Prescaler A divides the system clock, and the resulting clock is clk_{PER4}. Prescalers B and C can be enabled to divide the clock speed further to enable peripheral modules to run at twice or four times the CPU clock frequency. If Prescalers B and C are not used, all the clocks will run at the same frequency as the output from Prescaler A.

The system clock selection and prescaler registers are protected by the configuration change protection mechanism, employing a timed write procedure for changing the system clock and prescaler settings. For details, refer to "Configuration Change Protection" on page 13.

7.6 PLL with 1x-31x Multiplication Factor

The built-in phase locked loop (PLL) can be used to generate a high-frequency system clock. The PLL has a user-selectable multiplication factor of from 1 to 31. The output frequency, f_{OUT}, is given by the input frequency, f_{IN}, multiplied by the multiplication factor, PLL_FAC.

$$f_{OUT} = f_{IN} \cdot PLL\_FAC$$

Four different clock sources can be chosen as input to the PLL:

- 2MHz internal oscillator
- 32MHz internal oscillator divided by 4
- 0.4MHz - 16MHz crystal oscillator
- External clock

To enable the PLL, the following procedure must be followed:

1. Enable reference clock source.

2. Set the multiplication factor and select the clock reference for the PLL.

3. Wait until the clock reference source is stable.

4. Enable the PLL.

Hardware ensures that the PLL configuration cannot be changed when the PLL is in use. The PLL must be disabled before a new configuration can be written.

It is not possible to use the PLL before the selected clock source is stable and the PLL has locked.

The reference clock source cannot be disabled while the PLL is running.

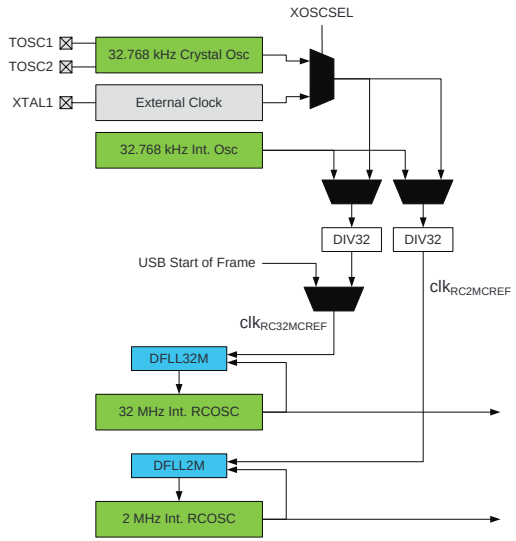
7.7 DFLL 2MHz and DFLL 32MHz

Two built-in digital frequency locked loops (DFLLs) can be used to improve the accuracy of the 2MHz and 32MHz internal oscillators. The DFLL compares the oscillator frequency with a more accurate reference clock to do automatic run-time calibration of the oscillator and compensate for temperature and voltage drift. The choices for the reference clock sources are:

- 32.768kHz calibrated internal oscillator
- 32.768kHz crystal oscillator connected to the TOSC pins
- External clock
- USB start of frame

The DFLLs divide the oscillator reference clock by 32 to use a 1.024kHz reference. The reference clock is individually selected for each DFLL, as shown on Figure 7-6 on page 88.

Figure 7-6. DFLL reference clock selection.



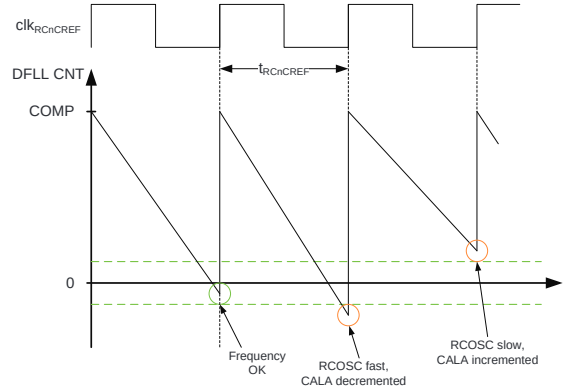
The ideal counter value representing the frequency ratio between the internal oscillator and a 1.024kHz reference clock is loaded into the DFLL oscillator compare register (COMP) during reset. For the 32MHz oscillator, this register can be written from software to make the oscillator run at a different frequency or when the ratio between the reference clock and the oscillator is different (for example when the USB start of frame is used). The 48MHz calibration values must be read from the production signature row and written to the 32MHz CAL register before the DFLL is enabled with USB SOF as reference source.

The value that should be written to the COMP register is given by the following formula:

$$COMP \approx hex\left(\frac{f_{OSC}}{f_{RCnCREF}}\right)$$

When the DFLL is enabled, it controls the ratio between the reference clock frequency and the oscillator frequency. If the internal oscillator runs too fast or too slow, the DFLL will decrement or increment its calibration register value by one to adjust the oscillator frequency. The oscillator is considered running too fast or too slow when the error is more than a half calibration step size.

Figure 7-7. Automatic run-time calibration.



The DFLL will stop when entering a sleep mode where the oscillators are stopped. After wake up, the DFLL will continue with the calibration value found before entering sleep. The reset value of the DFLL calibration register can be read from the production signature row.

When the DFLL is disabled, the DFLL calibration register can be written from software for manual run-time calibration of the oscillator.

7.8 PLL and External Clock Source Failure Monitor

A built-in failure monitor is available for the PLL and external clock source. If the failure monitor is enabled for the PLL and/or the external clock source, and this clock source fails (the PLL looses lock or the external clock source stops) while being used as the system clock, the device will:

- Switch to run the system clock from the 2MHz internal oscillator
- Reset the oscillator control register and system clock selection register to their default values
- Set the failure detection interrupt flag for the failing clock source (PLL or external clock)
- Issue a non-maskable interrupt (NMI)

If the PLL or external clock source fails when not being used for the system clock, it is automatically disabled, and the system clock will continue to operate normally. No NMI is issued. The failure monitor is meant for external clock sources above 32kHz. It cannot be used for slower external clocks.

When the failure monitor is enabled, it will not be disabled until the next reset.

The failure monitor is stopped in all sleep modes where the PLL or external clock source are stopped. During wake up from sleep, it is automatically restarted.

The PLL and external clock source failure monitor settings are protected by the configuration change protection mechanism, employing a timed write procedure for changing the settings. For details, refer to “Configuration Change Protection” on page 13.

7.9 Register Description – Clock

7.9.1 CTRL – Control register

Bit	7	6	5	4	3	2	1	0
+0x00	–	–	–	–	–	SCLKSEL[2:0]		
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:3 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 2:0 – SCLKSEL[2:0]: System Clock Selection
These bits are used to select the source for the system clock. See Table 7-1 on page 90 for the different selections. Changing the system clock source will take two clock cycles on the old clock source and two more clock cycles on the new clock source. These bits are protected by the configuration change protection mechanism. For details, refer to "Configuration Change Protection" on page 13.

SCLKSEL cannot be changed if the new clock source is not stable. The old clock can not be disabled until the clock switching is completed.

Table 7-1. System clock selection.

SCLKSEL[2:0]	Group configuration	Description
000	RC2MHZ	2MHz internal oscillator
001	RC32MHZ	32MHz internal oscillator
010	RC32KHZ	32.768kHz internal oscillator
011	XOSC	External oscillator or clock
100	PLL	Phase locked loop
101	–	Reserved
110	–	Reserved
111	–	Reserved

7.9.2 PSCTRL – Prescaler register

This register is protected by the configuration change protection mechanism. For details, refer to "Configuration Change Protection" on page 13.

Bit	7	6	5	4	3	2	1	0
+0x01	–	PSADIV[4:0]					PSBCDIV	
Read/Write	R	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 – Reserved
This bit is unused and reserved for future use. For compatibility with future devices, always write this bit to zero when this register is written.

Bit 6:2 – PSADIV[4:0]: Prescaler A Division Factor
These bits define the division ratio of the clock prescaler A according to Table 7-2 on page 91. These bits can be written at run-time to change the frequency of the Clk_{PER4} clock relative to the system clock, Clk_{SYS}.

Table 7-2. Prescaler A division factor.

PSADIV[4:0]	Group configuration	Description
00000	1	No division
00001	2	Divide by 2
00011	4	Divide by 4
00101	8	Divide by 8
00111	16	Divide by 16
01001	32	Divide by 32
01011	64	Divide by 64
01101	128	Divide by 128
01111	256	Divide by 256
10001	512	Divide by 512
10101		Reserved
10111		Reserved
11001		Reserved
11011		Reserved
11101		Reserved
11111		Reserved

Bit 1:0 – PSBCDIV: Prescaler B and C Division Factors
These bits define the division ratio of the clock prescalers B and C according to Table 7-3 on page 91. Prescaler B will set the clock frequency for the Clk_{PER2} clock relative to the Clk_{PER4} clock. Prescaler C will set the clock frequency for the Clk_{PER} and Clk_{CPU} clocks relative to the Clk_{PER2} clock. Refer to Figure 7-5 on page 86 fore more details.

Table 7-3. Prescaler B and C division factors.

PSBCDIV[1:0]	Group configuration	Prescaler B division	Prescaler C division
00	1_1	No division	No division
01	1_2	No division	Divide by 2
10	4_1	Divide by 4	No division
11	2_2	Divide by 2	Divide by 2

7.9.3 LOCK – Lock register

Bit	7	6	5	4	3	2	1	0
+0x02	–	–	–	–	–	–	–	LOCK
Read/Write	R	R	R	R	R	R	R	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:1 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 0 – LOCK: Clock System Lock
When this bit is written to one, the CTRL and PSCTRL registers cannot be changed, and the system clock selection and prescaler settings are protected against all further updates until after the next reset. This bit is protected by the configuration change protection mechanism. For details, refer to "Configuration Change Protection" on page 13.

The LOCK bit can be cleared only by a reset.

7.9.4 RTCCTRL – RTC Control register

Bit	7	6	5	4	3	2	1	0
+0x03	–	–	–	–	RTCSRC[2:0]			RTCEN
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:4 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 3:1 – RTCSRC[2:0]: RTC Clock Source
These bits select the clock source for the real-time counter according to Table 7-4 on page 92.

Table 7-4. RTC clock source selection(1).

RTCSRC[2:0]	Group configuration	Description
000	ULP	1kHz from 32kHz internal ULP oscillator
001	TOSC	1.024kHz from 32.768kHz crystal oscillator on TOSC
010	RCOSC	1.024kHz from 32.768kHz internal oscillator(2)
011	–	Reserved
100	–	Reserved
101	TOSC32	32.768kHz from 32.768kHz crystal oscillator on TOSC
110	RCOSC32	32.768kHz from 32.768kHz internal oscillator
111	EXTCLK	External clock from TOSC1(2)

Notes: 1. This table is not applicable for RTC32
2. Not available on devices with Battery Backup System

Bit 0 – RTCEN: RTC Clock Source Enable
Setting the RTCEN bit enables the selected RTC clock source for the real-time counter.

7.9.5 USBCTRL – USB Control register

Bit	7	6	5	4	3	2	1	0
+0x04	–	–	USBPSDIV[2:0]			USBSRC[1:0]		USBSEN
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

Bit 7:6 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 5:3 – USBPSDIV[2:0]: USB Prescaler Division Factor
These bits define the division ratio of the USB clock prescaler according to Table 7-5 on page 93. These bits are locked as long as the USB clock source is enabled.

Table 7-5. USB prescaler division factor.

USBPSDIV[2:0]	Group configuration	Description
000	1	No division
001	2	Divide by 2
010	4	Divide by 4
011	8	Divide by 8
100	16	Divide by 16
101	32	Divide by 32
110	–	Reserved
111	–	Reserved

Bit 2:1 – USBSRC[1:0]: USB Clock Source
These bits select the clock source for the USB module according to Table 7-6 on page 93.

Table 7-6. USB clock source.

USBSRC[1:0]	Group configuration	Description
00	PLL	PLL
01	RC32M	32MHz internal oscillator(1)

Note: 1. The 32MHz internal oscillator must be calibrated to 48MHz before selecting this as source for the USB device module. Refer to "DFLL 2MHz and DFLL 32MHz" on page 87.

Bit 0 – USBSEN: USB Clock Source Enable
Setting this bit enables the selected clock source for the USB device module.

7.10 Register Description – Oscillator

7.10.1 CTRL – Oscillator Control register

Bit	7	6	5	4	3	2	1	0
+0x00	–	–	–	PLLEN	XOSCEN	RC32KEN	RC32MEN	RC2MEN
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	1

Bit 7:5 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 4 – PLLEN: PLL Enable
Setting this bit enables the PLL. Before the PLL is enabled, it must be configured with the desired multiplication factor and clock source. See "STATUS – Oscillator Status register" on page 94.

Bit 3 – XOSCEN: External Oscillator Enable
Setting this bit enables the selected external clock source. Refer to "XOSCCTRL – XOSC Control register" on page 95 for details on how to select the external clock source. The external clock source should be allowed time to stabilize before it is selected as the source for the system clock. See "STATUS – Oscillator Status register" on page 94.

Bit 2 – RC32KEN: 32.768kHz Internal Oscillator Enable
Setting this bit enables the 32.768kHz internal oscillator. The oscillator must be stable before it is selected as the source for the system clock. See "STATUS – Oscillator Status register" on page 94.

Bit 1 – RC32MEN: 32MHz Internal Oscillator Enable
Setting this bit will enable the 32MHz internal oscillator. The oscillator must be stable before it is selected as the source for the system clock. See "STATUS – Oscillator Status register" on page 94.

Bit 0 – RC2MEN: 2MHz Internal Oscillator Enable
Setting this bit enables the 2MHz internal oscillator. The oscillator must be stable before it is selected as the source for the system clock. See "STATUS – Oscillator Status register" on page 94.
By default, the 2MHz internal oscillator is enabled and this bit is set.

7.10.2 STATUS – Oscillator Status register

Bit	7	6	5	4	3	2	1	0
+0x01	–	–	–	PLLRDY	XOSCRDY	RC32KRDY	RC32MRDY	RC2MRDY
Read/Write	R	R	R	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

Bit 7:5 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 4 – PLLRDY: PLL Ready
This flag is set when the PLL has locked on the selected frequency and is ready to be used as the system clock source.

Bit 3 – XOSCRDY: External Clock Source Ready
This flag is set when the external clock source is stable and is ready to be used as the system clock source.

Bit 2 – RC32KRDY: 32.768kHz Internal Oscillator Ready
This flag is set when the 32.768kHz internal oscillator is stable and is ready to be used as the system clock source.

Bit 1 – RC32MRDY: 32MHz Internal Oscillator Ready
This flag is set when the 32MHz internal oscillator is stable and is ready to be used as the system clock source.

Bit 0 – RC2MRDY: 2MHz Internal Oscillator Ready
This flag is set when the 2MHz internal oscillator is stable and is ready to be used as the system clock source.

7.10.3 XOSCCTRL – XOSC Control register

Bit	7	6	5	4	3	2	1	0
+0x02	FRQRANGE[1:0]		X32KLPM	XOSCPWR	XOSCSEL[3:0]			
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 7:6 – FRQRANGE[1:0]: 0.4 - 16MHz Crystal Oscillator Frequency Range Select
These bits select the frequency range for the connected crystal oscillator according to Table 7-7 on page 95.

Table 7-7. 16MHz crystal oscillator frequency range selection.

FRQRANGE[1:0]	Group configuration	Typical frequency range	Recommended range for capacitors C1 and C2 (pF)
00	04TO2	0.4MHz - 2MHz	100-300
01	2TO9	2MHz - 9MHz	10-40
10	9TO12	9MHz - 12MHz	10-40
11	12TO16	12MHz - 16MHz	10-30

Note: Refer to Electrical characteristics section in device datasheet to retrieve the best setting for a given frequency.

Bit 5 – X32KLPM: Crystal Oscillator 32.768kHz Low Power Mode
Setting this bit enables the low power mode for the 32.768kHz crystal oscillator. This will reduce the swing on the TOSC2 pin.

Bit 4 – XOSCPWR: Crystal Oscillator Drive
Setting this bit will increase the current in the 0.4MHz - 16MHz crystal oscillator and increase the swing on the XTAL2 pin. This allows for driving crystals with higher load or higher frequency than specified by the FRQRANGE bits.

Bit 3:0 – XOSCSEL[3:0]: Crystal Oscillator Selection
These bits select the type and start-up time for the crystal or resonator that is connected to the XTAL or TOSC pins. See Table 7-8 on page 96 for crystal selections. If an external clock or external oscillator is selected as the source for the system clock, see "CTRL – Oscillator Control register" on page 94. This configuration cannot be changed.

Table 7-8. External oscillator selection and start-up time.

XOSCSEL[3:0]	Group configuration	Selected clock source	Start-up time
0000	EXTCLK[3]	External Clock	6 CLK
0010	32KHZ[3]	32.768kHz TOSC	16K CLK
0011	XTAL_256CLK[1]	0.4MHz - 16MHz XTAL	256 CLK
0111	XTAL_1KCLK[2]	0.4MHz - 16MHz XTAL	1K CLK
1011	XTAL_16KCLK	0.4MHz - 16MHz XTAL	16K CLK

Notes:
1. This option should be used only when frequency stability at startup is not important for the application. The option is not suitable for crystals.
2. This option is intended for use with ceramic resonators. It can also be used when the frequency stability at startup is not important for the application.
3. When the external oscillator is used as the reference for a DFLL, only EXTCLK and 32KHZ can be selected.

7.10.4 XOSCFAIL – XOSC Failure Detection register

Bit	7	6	5	4	3	2	1	0
+0x03	–	–	–	–	PLLFDIF	PLLFDEN	XOSCFDIF	XOSCFDEN
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:4 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 3 – PLLFDIF: PLL Fault Detection Flag
If PLL failure detection is enabled, PLLFDIF is set when the PLL looses lock. Writing logic one to this location will clear PLLFDIF.

Bit 2 – PLLFDEN: PLL Fault Detection Enable
Setting this bit will enable PLL failure detection. A non-maskable interrupt will be issued when PLLFDIF is set.
This bit is protected by the configuration change protection mechanism. Refer to "Configuration Change Protection" on page 13 for details.

Bit 1 – XOSCFDIF: Failure Detection Interrupt Flag
If the external clock source oscillator failure monitor is enabled, XOSCFDIF is set when a failure is detected. Writing logic one to this location will clear XOSCFDIF.

Bit 0 – XOSCFDEN: Failure Detection Enable
Setting this bit will enable the failure detection monitor, and a non-maskable interrupt will be issued when XOSCFDIF is set.
This bit is protected by the configuration change protection mechanism. Refer to "Configuration Change Protection" on page 13 for details. Once enabled, failure detection can only be disabled by a reset.

7.10.5 RC32KCAL – 32kHz Oscillator Calibration register

Bit	7	6	5	4	3	2	1	0
+0x04	RC32KCAL[7:0]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	x	x	x	x	x	x	x	x

Bit 7:0 – RC32KCAL[7:0]: 32.768kHz Internal Oscillator Calibration bits
This register is used to calibrate the 32.768kHz internal oscillator. A factory-calibrated value is loaded from the signature row of the device and written to this register during reset, giving an oscillator frequency close to 32.768kHz. The register can also be written from software to calibrate the oscillator frequency during normal operation.

7.10.6 PLLCTRL – PLL Control register

Bit	7	6	5	4	3	2	1	0
+0x05	PLLSRC[1:0]		PLLDIV	PLLFAC[4:0]				
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 7:6 – PLLSRC[1:0]: Clock Source
The PLLSRC bits select the input source for the PLL according to Table 7-9 on page 100.

Table 7-9. PLL clock source.

PLLSRC[1:0]	Group configuration	PLL input source
00	RC2M	2MHz internal oscillator
01	–	Reserved
10	RC32M	32MHz internal oscillator
11	XOSC	External clock source[1]

Notes:
1. The 32.768kHz TOSC cannot be selected as the source for the PLL. An external clock must be a minimum 0.4MHz to be used as the source clock.

Bit 5 – PLLDIV: PLL Divided Output Enable
Setting this bit will divide the output from the PLL by 2.

Bit 4:0 – PLLFAC[4:0]: Multiplication Factor
These bits select the multiplication factor for the PLL. The multiplication factor can be in the range of from 1x to 31x.

7.10.7 DFLLCTRL – DFLL Control register

Bit	7	6	5	4	3	2	1	0
+0x06	–	–	–	–	–	RC32MCREF[1:0]		RC2MCREF
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:3 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 2:1 – RC32MCREF[1:0]: 32MHz Oscillator Calibration Reference
These bits are used to select the calibration source for the 32MHz DFLL according to the Table 7-10 on page 101. These bits will select only which calibration source to use for the DFLL. In addition, the actual clock source that is selected must enabled and configured for the calibration to function.

Table 7-10. 32MHz oscillator reference selection.

RC32MCREF[1:0]	Group configuration	Description
00	RC32K	32.768kHz internal oscillator
01	XOSC32	32.768kHz crystal oscillator on TOSC
10	USBSOF	USB start of frame
11	–	Reserved

Bit 0 – RC2MCREF: 2MHz Oscillator Calibration Reference
This bit is used to select the calibration source for the 2MHz DFLL. By default, this bit is zero and the 32.768kHz internal oscillator is selected. If this bit is set to one, the 32.768kHz crystal oscillator on TOSC is selected as the reference. This bit will select only which calibration source to use for the DFLL. In addition, the actual clock source that is selected must enabled and configured for the calibration to function.

7.11 Register Description – DFLL32M/DFLL2M

7.11.1 CTRL – DFLL Control register

Bit	7	6	5	4	3	2	1	0
+0x00	–	–	–	–	–	–	–	ENABLE
Read/Write	R	R	R	R	R	R	R	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:1 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 0 – ENABLE: DFLL Enable
Setting this bit enables the DFLL and auto-calibration of the internal oscillator. The reference clock must be enabled and stable before the DFLL is enabled.
After disabling the DFLL, the reference clock can not be disabled before the ENABLE bit is read as zero.

7.11.2 CALA – DFLL Calibration Register A

The CALA and CALB registers hold the 13-bit DFLL calibration value that is used for automatic run-time calibration of the internal oscillator. When the DFLL is disabled, the calibration registers can be written by software for manual run-time calibration of the oscillator. The oscillators will also be calibrated according to the calibration value in these registers when the DFLL is disabled.

Bit	7	6	5	4	3	2	1	0
+0x02	–	CALA[6:0]						
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	x	x	x	x	x	x	x

Bit 7 – Reserved
This bit is unused and reserved for future use. For compatibility with future devices, always write this bit to zero when this register is written.

Bit 6:0 – CALA[6:0]: DFLL Calibration Bits
These bits hold the part of the oscillator calibration value that is used for automatic runtime calibration. A factory-calibrated value is loaded from the signature row of the device and written to this register during reset, giving an oscillator frequency approximate to the nominal frequency for the oscillator. The bits cannot be written when the DFLL is enabled.

7.11.3 CALB – DFLL Calibration register B

Bit	7	6	5	4	3	2	1	0
+0x03	–	–	CALB[5:0]					
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	x	x	x	x	x	x

Bit 7:6 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 5:0 – CALB[5:0]: DFLL Calibration bits
These bits hold the part of the oscillator calibration value that is used to select the oscillator frequency. A factory-calibrated value is loaded from the signature row of the device and written to this register during reset, giving an oscillator frequency approximate to the nominal frequency for the oscillator. These bits are not changed during automatic run-time calibration of the oscillator. The bits cannot be written when the DFLL is enabled. When calibrating to a frequency different from the default, the CALA bits should be set to a middle value to maximize the range for the DFLL.

7.11.4 COMP1 – DFLL Compare register 1

The COMP1 and COMP2 register pair represent the frequency ratio between the oscillator and the reference clock. The initial value for these registers is the ratio between the internal oscillator frequency and a 1.024kHz reference.

The initial value for these registers is the ratio between the internal oscillator frequency and a 1.024kHz reference; 0x7A12 for 32 MHz DFLL.

Bit	7	6	5	4	3	2	1	0
+0x05	COMP[7:0]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	–	–	–	–	–	–	–	–

Bit 7:0 – COMP1[7:0]: Compare Byte 1
These bits hold byte 1 of the 16-bit compare register.

7.11.5 COMP2 – DFLL Compare register 2

Bit	7	6	5	4	3	2	1	0
+0x06	COMP[15:8]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	–	–	–	–	–	–	–	–

Bit 7:0 – COMP2[15:8]: Compare Byte 2
These bits hold byte 2 of the 16-bit compare register.

Table 7-11. Nominal DFLL32M COMP values for different output frequencies.

Oscillator frequency (MHz)	COMP value ($Clk_{RCnCREF}$ = 1.024kHz)
30.0	0x7270
32.0	0x7A12
34.0	0x81B3
36.0	0x8954
38.0	0x90F5
40.0	0x9896
42.0	0xA037
44.0	0xA7D8
46.0	0xAF79
48.0	0xB71B
50.0	0xBEBC
52.0	0xC65D
54.0	0xCDFE

7.12 Register summary – Clock

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	CTRL	–	–	–	–	–	SCLKSEL[2:0]			90
+0x01	PSCTRL	–	PSADIV[4:0]					PSBCDIV[1:0]		90
+0x02	LOCK	–	–	–	–	–	–	–	LOCK	92
+0x03	RTCCTRL	–	–	–	–	RTCSRC[2:0]			RTCEN	92
+0x04	USBSCTRL	–	USBPSDIV[2:0]			USBSRC[1:0]		USBSEN	USBPSDIV[2:0]	92
+0x05	Reserved	–	–	–	–	–	–	–	–	
+0x06	Reserved	–	–	–	–	–	–	–	–	
+0x07	Reserved	–	–	–	–	–	–	–	–	

7.13 Register summary – Oscillator

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	CTRL	–	–	–	PLLEN	XOSCEN	RC32KEN	R32MEN	RC2MEN	94
+0x01	STATUS	–	–	–	PLLRDY	XOSCRDY	RC32KRDY	R32MRDY	RC2MRDY	94
+0x02	XOSCCTRL	FRQRANGE[1:0]		X32KLPM	XOSCPWR	XOSCSEL[3:0]				95
+0x03	XOSCFAIL	–	–	–	–	PLLFDIF	PLLFDEN	XOSCFDIF	XOSCFDEN	96
+0x04	RC32KCAL	RC32KCAL[7:0]								97
+0x05	PLLCTRL	PLLSRC[1:0]		PLLDIV	PLLFAC[4:0]					97
+0x06	DFLLCTRL	–	–	–	–	–	RC32MCREF[1:0]		RC2MCREF	97
+0x07	Reserved	–	–	–	–	–	–	–	–	

7.14 Register summary – DFLL32M/DFLL2M

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	CTRL	–	–	–	–	–	–	–	ENABLE	98
+0x01	Reserved	–	–	–	–	–	–	–	–	
+0x02	CALA	–	CALA[6:0]							98
+0x03	CALB	–	–	CALB[5:0]						99
+0x04	Reserved	–	–	–	–	–	–	–	–	
+0x05	COMP1	COMP[7:0]								99
+0x06	COMP2	COMP[15:8]								99
+0x07	Reserved	–	–	–	–	–	–	–	–	

7.15 Oscillator failure interrupt vector summary

Table 7-12. Oscillator failure interrupt vector and its word offset address PLL and external oscillator failure interrupt base.

Offset	Source	Interrupt Description
0x00	OSCF_vect	PLL and external oscillator failure interrupt vector (NMI)

12. Interrupts and Programmable Multilevel Interrupt Controller

12.1 Features

- Short and predictable interrupt response time
- Separate interrupt configuration and vector address for each interrupt
- Programmable multilevel interrupt controller
  - Interrupt prioritizing according to level and vector address
  - Three selectable interrupt levels for all interrupts: low, medium and high
  - Selectable, round-robin priority scheme within low-level interrupts
  - Non-maskable interrupts for critical functions
- Interrupt vectors optionally placed in the application section or the boot loader section

12.2 Overview

Interrupts signal a change of state in peripherals, and this can be used to alter program execution. Peripherals can have one or more interrupts, and all are individually enabled and configured. When an interrupt is enabled and configured, it will generate an interrupt request when the interrupt condition is present. The programmable multilevel interrupt controller (PMIC) controls the handling and prioritizing of interrupt requests. When an interrupt request is acknowledged by the PMIC, the program counter is set to point to the interrupt vector, and the interrupt handler can be executed.

All peripherals can select between three different priority levels for their interrupts: low, medium, and high. Interrupts are prioritized according to their level and their interrupt vector address. Medium-level interrupts will interrupt low-level interrupt handlers. High-level interrupts will interrupt both medium- and low-level interrupt handlers. Within each level, the interrupt priority is decided from the interrupt vector address, where the lowest interrupt vector address has the highest interrupt priority. Low-level interrupts have an optional round-robin scheduling scheme to ensure that all interrupts are serviced within a certain amount of time.

Non-maskable interrupts (NMI) are also supported, and can be used for system critical functions.

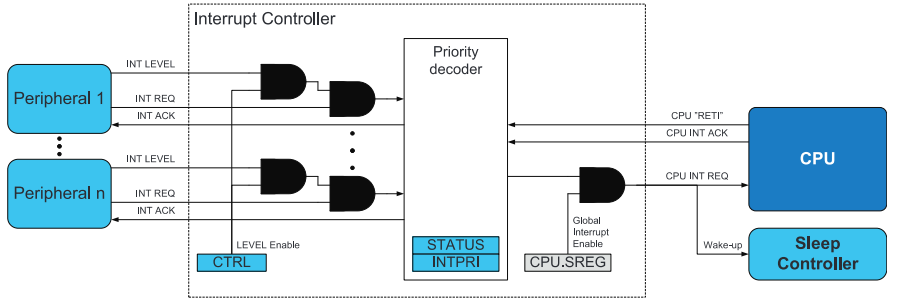
12.3 Operation

Interrupts must be globally enabled for any interrupts to be generated. This is done by setting the global interrupt enable (I) bit in the CPU status register. The I bit will not be cleared when an interrupt is acknowledged. Each interrupt level must also be enabled before interrupts with the corresponding level can be generated.

When an interrupt is enabled and the interrupt condition is present, the PMIC will receive the interrupt request. Based on the interrupt level and interrupt priority of any ongoing interrupts, the interrupt is either acknowledged or kept pending until it has priority. When the interrupt request is acknowledged, the program counter is updated to point to the interrupt vector. The interrupt vector is normally a jump to the interrupt handler; the software routine that handles the interrupt. After returning from the interrupt handler, program execution continues from where it was before the interrupt occurred. One instruction is always executed before any pending interrupt is served.

The PMIC status register contains state information that ensures that the PMIC returns to the correct interrupt level when the RETI (interrupt return) instruction is executed at the end of an interrupt handler. Returning from an interrupt will return the PMIC to the state it had before entering the interrupt. The status register (SREG) is not saved automatically upon an interrupt request. The RET (subroutine return) instruction cannot be used when returning from the interrupt handler routine, as this will not return the PMIC to its correct state.

Figure 12-1. Interrupt controller overview



12.4 Interrupts

All interrupts and the reset vector each have a separate program vector address in the program memory space. The lowest address in the program memory space is the reset vector. All interrupts are assigned individual control bits for enabling and setting the interrupt level, and this is set in the control registers for each peripheral that can generate interrupts. Details on each interrupt are described in the peripheral where the interrupt is available.

All interrupts have an interrupt flag associated with it. When the interrupt condition is present, the interrupt flag will be set, even if the corresponding interrupt is not enabled. For most interrupts, the interrupt flag is automatically cleared when executing the interrupt vector. Writing a logical one to the interrupt flag will also clear the flag. Some interrupt flags are not cleared when executing the interrupt vector, and some are cleared automatically when an associated register is accessed (read or written). This is described for each individual interrupt flag.

If an interrupt condition occurs while another, higher priority interrupt is executing or pending, the interrupt flag will be set and remembered until the interrupt has priority. If an interrupt condition occurs while the corresponding interrupt is not enabled, the interrupt flag will be set and remembered until the interrupt is enabled or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while global interrupts are disabled, the corresponding interrupt flag will be set and remembered until global interrupts are enabled. All pending interrupts are then executed according to their order of priority.

Interrupts can be blocked when executing code from a locked section; e.g., when the boot lock bits are programmed. This feature improves software security. Refer to “Memory Programming” on page 407 for details on lock bit settings.

Interrupts are automatically disabled for up to four CPU clock cycles when the configuration change protection register is written with the correct signature. Refer to “Configuration Change Protection” on page 13 for more details.

12.4.1 NMI – Non-Maskable Interrupts

Which interrupts represent NMI and which represent regular interrupts cannot be selected. Non-maskable interrupts must be enabled before they can be used. Refer to the device datasheet for NMI present on each device.

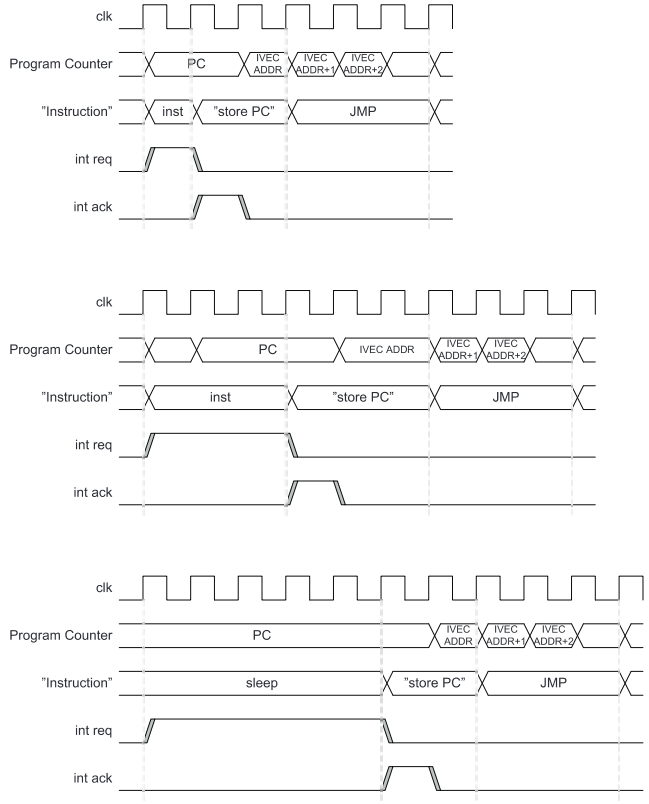
An NMI will be executed regardless of the setting of the I bit, and it will never change the I bit. No other interrupts can interrupt a NMI handler. If more than one NMI is requested at the same time, priority is static according to the interrupt vector address, where the lowest address has highest priority.

12.4.2 Interrupt Response Time

The interrupt response time for all the enabled interrupts is three CPU clock cycles, minimum; one cycle to finish the ongoing instruction and two cycles to store the program counter to the stack. After the program counter is pushed on the stack, the program vector for the interrupt is executed. The jump to the interrupt handler takes three clock cycles.

If an interrupt occurs during execution of a multicycle instruction, this instruction is completed before the interrupt is served. See Figure 12-2 on page 133 for more details.

Figure 12-2. Interrupt execution of a multi cycle instruction.



If an interrupt occurs when the device is in sleep mode, the interrupt execution response time is increased by five clock cycles. In addition, the response time is increased by the start-up time from the selected sleep mode.

A return from an interrupt handling routine takes four to five clock cycles, depending on the size of the program counter. During these clock cycles, the program counter is popped from the stack and the stack pointer is incremented.

12.5 Interrupt level

The interrupt level is independently selected for each interrupt source. For any interrupt request, the PMIC also receives the interrupt level for the interrupt. The interrupt levels and their corresponding bit values for the interrupt level configuration of all interrupts is shown in Table 12-1.

Table 12-1. Interrupt levels.

Interrupt level configuration	Group configuration	Description
00	OFF	Interrupt disabled.
01	LO	Low-level interrupt
10	MED	Medium-level interrupt
11	HI	High-level interrupt

The interrupt level of an interrupt request is compared against the current level and status of the interrupt controller. An interrupt request of a higher level will interrupt any ongoing interrupt handler from a lower level interrupt. When returning from the higher level interrupt handler, the execution of the lower level interrupt handler will continue.

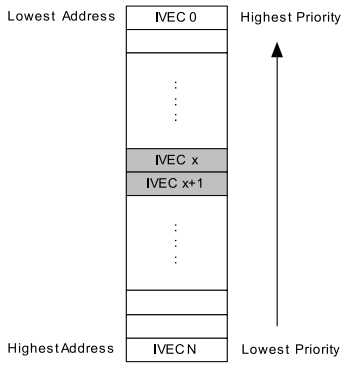
12.6 Interrupt priority

Within each interrupt level, all interrupts have a priority. When several interrupt requests are pending, the order in which interrupts are acknowledged is decided both by the level and the priority of the interrupt request. Interrupts can be organized in a static or dynamic (round-robin) priority scheme. High- and medium-level interrupts and the NMI will always have static priority. For low-level interrupts, static or dynamic priority scheduling can be selected.

12.6.1 Static priority

Interrupt vectors (IVEC) are located at fixed addresses. For static priority, the interrupt vector address decides the priority within one interrupt level, where the lowest interrupt vector address has the highest priority. Refer to the device datasheet for the interrupt vector table with the base address for all modules and peripherals with interrupt capability. Refer to the interrupt vector summary of each module and peripheral in this manual for a list of interrupts and their corresponding offset address within the different modules and peripherals.

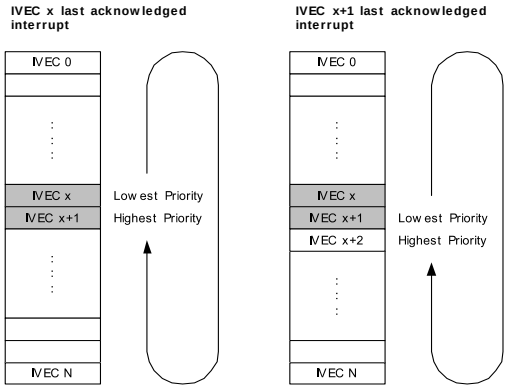
Figure 12-3. Static priority.



12.6.2 Round-robin Scheduling

To avoid the possible starvation problem for low-level interrupts with static priority, where some interrupts might never be served, the PMIC offers round-robin scheduling for low-level interrupts. When round-robin scheduling is enabled, the interrupt vector address for the last acknowledged low-level interrupt will have the lowest priority the next time one or more interrupts from the low level is requested.

Figure 12-4. Round-robin scheduling.



12.7 Interrupt vector locations

Table 12-2 on page 135 shows reset and Interrupt vectors placement for the various combinations of BOOTRST and IVSEL settings. If the program never enables an interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations. This is also the case if the Reset Vector is in the Application section while the Interrupt Vectors are in the Boot section or vice versa.

Table 12-2. Reset and interrupt vectors placement.

BOOTRST	IVSEL	Reset address	Interrupt vectors start address
1	0	0x0000	0x0002
1	1	0x0000	Boot Reset Address + 0x0002
0	0	Boot Reset Address	0x0002
0	1	Boot Reset Address	Boot Reset Address + 0x0002

12.8 Register description

12.8.1 STATUS – Status register

Bit	7	6	5	4	3	2	1	0
+0x00	NMIEX	–	–	–	–	HILVLEX	MEDLVLEX	LOLVLEX
Read/Write	R	R	R	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7 – NMIEX: Non-Maskable Interrupt Executing**
  This flag is set if a non-maskable interrupt is executing. The flag will be cleared when returning (RETI) from the interrupt handler.
- **Bit 6:3 – Reserved**
  These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.
- **Bit 2 – HILVLEX: High-level Interrupt Executing**
  This flag is set when a high-level interrupt is executing or when the interrupt handler has been interrupted by an NMI. The flag will be cleared when returning (RETI) from the interrupt handler.
- **Bit 1 – MEDLVLEX: Medium-level Interrupt Executing**
  This flag is set when a medium-level interrupt is executing or when the interrupt handler has been interrupted by an interrupt from higher level or an NMI. The flag will be cleared when returning (RETI) from the interrupt handler.
- **Bit 0 – LOLVLEX: Low-level Interrupt Executing**
  This flag is set when a low-level interrupt is executing or when the interrupt handler has been interrupted by an interrupt from higher level or an NMI. The flag will be cleared when returning (RETI) from the interrupt handler.

12.8.2 INTPRI – Interrupt priority register

Bit	7	6	5	4	3	2	1	0
+0x01	INTPRI[7:0]							
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 7:0 – INTPRI: Interrupt Priority**
  When round-robin scheduling is enabled, this register stores the interrupt vector of the last acknowledged low-level interrupt. The stored interrupt vector will have the lowest priority the next time one or more low-level interrupts are pending. The register is accessible from software to change the priority queue. This register is not reinitialized to its initial value if round-robing scheduling is disabled, and so if default static priority is needed, the register must be written to zero.

12.8.3 CTRL – Control register

Bit	7	6	5	4	3	2	1	0
+0x02	RREN	IVSEL	–	–	–	HILVLEN	MEDLVLEN	LOLVLEN
Read/Write	R/W	R/W	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7 – RREN: Round-robin Scheduling Enable**
  When the RREN bit is set, the round-robin scheduling scheme is enabled for low-level interrupts. When this bit is cleared, the priority is static according to interrupt vector address, where the lowest address has the highest priority.
- **Bit 6 – IVSEL: Interrupt Vector Select**
  When the IVSEL bit is cleared (zero), the interrupt vectors are placed at the start of the application section in flash. When this bit is set (one), the interrupt vectors are placed in the beginning of the boot section of the flash. Refer to the device datasheet for the absolute address.
  This bit is protected by the configuration change protection mechanism. Refer to "Configuration Change Protection" on page 13 for details.
- **Bit 5:3 – Reserved**
  These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.
- **Bit 2 – HILVLEN: High-level Interrupt Enable**(1)
  When this bit is set, all high-level interrupts are enabled. If this bit is cleared, high-level interrupt requests will be ignored.
- **Bit 1 – MEDLVLEN: Medium-level Interrupt Enable**(1)
  When this bit is set, all medium-level interrupts are enabled. If this bit is cleared, medium-level interrupt requests will be ignored.
- **Bit 0 – LOLVLEN: Low-level Interrupt Enable**(1)
  When this bit is set, all low-level interrupts are enabled. If this bit is cleared, low-level interrupt requests will be ignored.

Note: 1. Ignoring interrupts will be effective one cycle after the bit is cleared.

12.9 Register summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	STATUS	NMIEX	–	–	–	–	HILVLEX	MEDLVLEX	LOLVLEX	136
+0x01	INTPRI	INTPRI[7:0]								136
+0x02	CTRL	RREN	IVSEL	–	–	–	HILVLEN	MEDLVLEN	LOLVLEN	137

13. I/O Ports

13.1 Features

- General purpose input and output pins with individual configuration
- Output driver with configurable driver and pull settings:
  - Totem-pole
  - Wired-AND
  - Wired-OR
  - Bus-keeper
  - Inverted I/O
- Input with synchronous and/or asynchronous sensing with interrupts and events
  - Sense both edges
  - Sense rising edges
  - Sense falling edges
  - Sense low level
- Optional pull-up and pull-down resistor on input and Wired-OR/AND configurations
- Optional slew rate control
- Asynchronous pin change sensing that can wake the device from all sleep modes
- Two port interrupts with pin masking per I/O port
- Efficient and safe access to port pins
  - Hardware read-modify-write through dedicated toggle/clear/set registers
  - Configuration of multiple pins in a single operation
  - Mapping of port registers into bit-accessible I/O memory space
- Peripheral clocks output on port pin
- Real-time counter clock output to port pin
- Event channels can be output on port pin
- Remapping of digital peripheral pin functions
  - Selectable USART, SPI, and timer/counter input/output pin locations

13.2 Overview

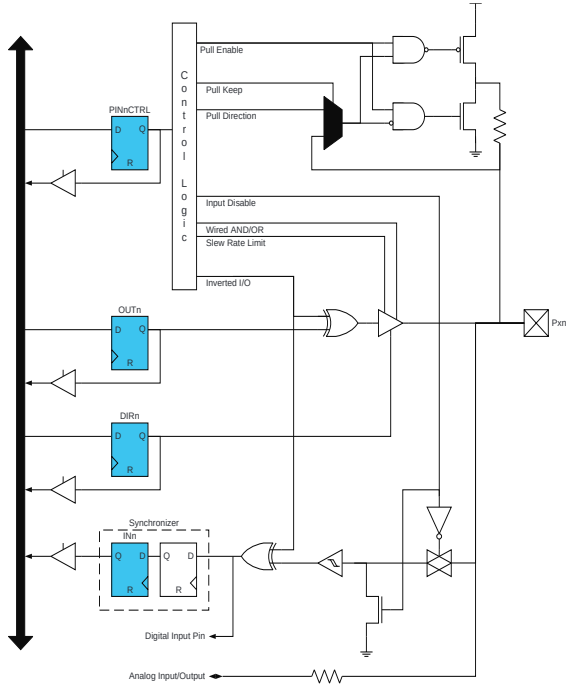
AVR XMEGA microcontrollers have flexible general purpose I/O ports. One port consists of up to eight port pins: pin 0 to 7. Each port pin can be configured as input or output with configurable driver and pull settings. They also implement synchronous and asynchronous input sensing with interrupts and events for selectable pin change conditions. Asynchronous pin-change sensing means that a pin change can wake the device from all sleep modes, included the modes where no clocks are running.

All functions are individual and configurable per pin, but several pins can be configured in a single operation. The pins have hardware read-modify-write (RMW) functionality for safe and correct change of drive value and/or pull resistor configuration. The direction of one port pin can be changed without unintentionally changing the direction of any other pin.

The port pin configuration also controls input and output selection of other device functions. It is possible to have both the peripheral clock and the real-time clock output to a port pin, and available for external use. The same applies to events from the event system that can be used to synchronize and control external functions. Other digital peripherals, such as USART, SPI, and timer/counters, can be remapped to selectable pin locations in order to optimize pin-out versus application needs.

Figure 13-1 on page 140 shows the I/O pin functionality and the registers that are available for controlling a pin.

Figure 13-1. General I/O pin functionality.



13.3 I/O Pin Use and Configuration

Each port has one data direction (DIR) register and one data output value (OUT) register that are used for port pin control. The data input value (IN) register is used for reading the port pins. In addition, each pin has a pin configuration (PINnCTRL) register for additional pin configuration.

Direction of the pin is decided by the DIRn bit in the DIR register. If DIRn is written to one, pin n is configured as an output pin. If DIRn is written to zero, pin n is configured as an input pin.

When direction is set as output, the OUTn bit in OUT is used to set the value of the pin. If OUTn is written to one, pin n is driven high. If OUTn is written to zero, pin n is driven low.

The IN register is used for reading pin values. A pin value can always be read regardless of whether the pin is configured as input or output, except if digital input is disabled.

The I/O pins are tri-stated when a reset condition becomes active, even if no clocks are running.

The pin n configuration (PINnCTRL) register is used for additional I/O pin configuration. A pin can be set in a totem-pole, wired-AND, or wired-OR configuration. It is also possible to enable inverted input and output for a pin.

A totem-pole output has four possible pull configurations: totem-pole (push-pull), pull-down, pull-up, and bus-keeper. The bus-keeper is active in both directions. This is to avoid oscillation when disabling the output. The totem-pole

configurations with pull-up and pull-down have active resistors only when the pin is set as input. This feature eliminates unnecessary power consumption. For wired-AND and wired-OR configuration, the optional pull-up and pull-down resistors are active in both input and output directions.

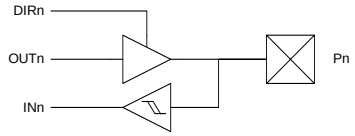
Since pull configuration is configured through the pin configuration register, all intermediate port states during switching of the pin direction and pin values are avoided.

The I/O pin configurations are summarized with simplified schematics in Figure 13-2 on page 141 to Figure 13-7 on page 143.

13.3.1 Totem-pole

In the totem-pole (push-pull) configuration, the pin is driven low or high according to the corresponding bit setting in the OUT register. In this configuration, there is no current limitation for sink or source other than what the pin is capable of. If the pin is configured for input, the pin will float if no external pull resistor is connected.

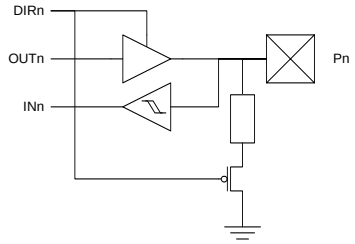
Figure 13-2. I/O pin configuration - Totem-pole (push-pull).



13.3.1.1 Totem-pole with Pull-down

In this mode, the configuration is the same as for totem-pole mode, expect the pin is configured with an internal pull-down resistor when set as input.

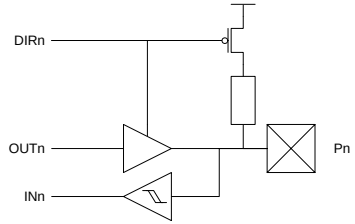
Figure 13-3. I/O pin configuration - Totem-pole with pull-down (on input).



13.3.1.2 Totem-pole with Pull-up

In this mode, the configuration is as for totem-pole, expect the pin is configured with internal pull-up when set as input.

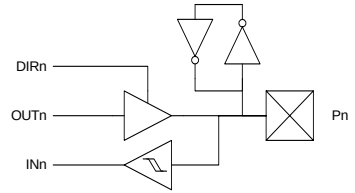
Figure 13-4. I/O pin configuration - Totem-pole with pull-up (on input).



13.3.2 Bus-keeper

In the bus-keeper configuration, it provides a weak bus-keeper that will keep the pin at its logic level when the pin is no longer driven to high or low. If the last level on the pin/bus was 1, the bus-keeper configuration will use the internal pull resistor to keep the bus high. If the last logic level on the pin/bus was 0, the bus-keeper will use the internal pull resistor to keep the bus low.

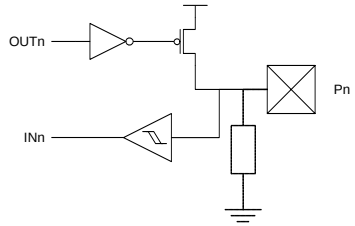
Figure 13-5. I/O pin configuration - Totem-pole with bus-keeper.



13.3.3 Wired-OR

In the wired-OR configuration, the pin will be driven high when the corresponding bits in the OUT and DIR registers are written to one. When the OUT register is set to zero, the pin is released, allowing the pin to be pulled low with the internal or an external pull-resistor. If internal pull-down is used, this is also active if the pin is set as input.

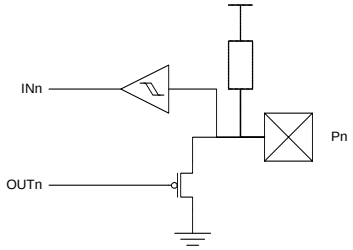
Figure 13-6. Output configuration - Wired-OR with optional pull-down.



13.3.4 Wired-AND

In the wired-AND configuration, the pin will be driven low when the corresponding bits in the OUT and DIR registers are written to zero. When the OUT register is set to one, the pin is released allowing the pin to be pulled high with the internal or an external pull-resistor. If internal pull-up is used, this is also active if the pin is set as input.

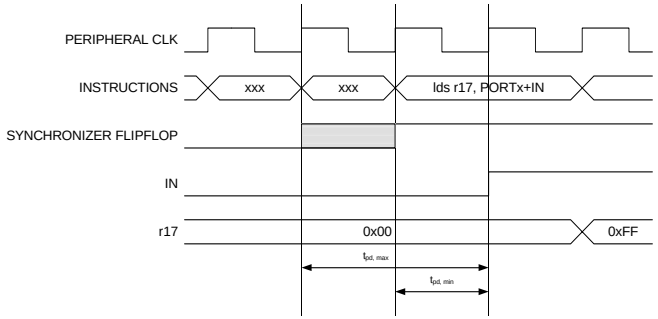
Figure 13-7. Output configuration - Wired-AND with optional pull-up.



13.4 Reading the Pin Value

Independent of the pin data direction, the pin value can be read from the IN register, as shown in Figure 13-1 on page 140. If the digital input is disabled, the pin value cannot be read. The IN register bit and the preceding flip-flop constitute a synchronizer. The synchronizer introduces a delay on the internal signal line. Figure 13-8 on page 143 shows a timing diagram of the synchronization when reading an externally applied pin value. The maximum and minimum propagation delays are denoted as $t_{pd,max}$ and $t_{pd,min}$, respectively.

Figure 13-8. Synchronization when reading a pin value.

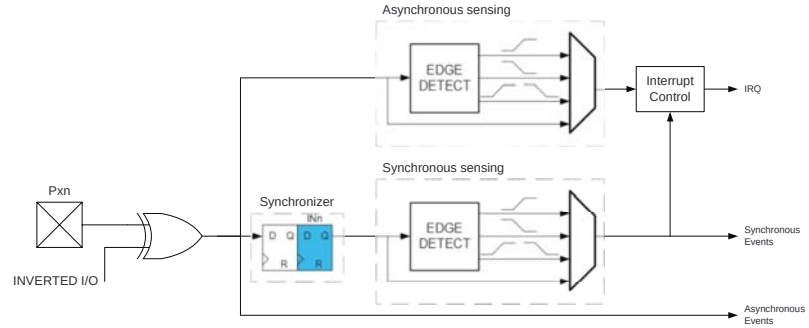


13.5 Input Sense Configuration

Input sensing is used to detect an edge or level on the I/O pin input. The different sense configurations that are available for each pin are detection of a rising edge, falling edge, or any edge or detection of a low level. High level can be detected by using the inverted input configuration. Input sensing can be used to trigger interrupt requests (IREQ) or events when there is a change on the pin.

The I/O pins support synchronous and asynchronous input sensing. Synchronous sensing requires the presence of the peripheral clock, while asynchronous sensing does not require any clock.

Figure 13-9. Input sensing.



13.6 Port Interrupt

Each port has two interrupt vectors, and it is configurable which pins on the port will trigger each interrupt. Port interrupts must be enabled before they can be used. Which sense configurations can be used to generate interrupts is dependent on whether synchronous or asynchronous input sensing is available for the selected pin.

For synchronous sensing, all sense configurations can be used to generate interrupts. For edge detection, the changed pin value must be sampled once by the peripheral clock for an interrupt request to be generated.

For asynchronous sensing, only port pin 2 on each port has full asynchronous sense support. This means that for edge detection, pin 2 will detect and latch any edge and it will always trigger an interrupt request. The other port pins have limited asynchronous sense support. This means that for edge detection, the changed value must be held until the device wakes up and a clock is present. If the pin value returns to its initial value before the end of the device wake-up time, the device will still wake up, but no interrupt request will be generated.

A low level can always be detected by all pins, regardless of a peripheral clock being present or not. If a pin is configured for low-level sensing, the interrupt will trigger as long as the pin is held low. In active mode, the low level must be held until the completion of the currently executing instruction for an interrupt to be generated. In all sleep modes, the low level must be kept until the end of the device wake-up time for an interrupt to be generated. If the low level disappears before the end of the wake-up time, the device will still wake up, but no interrupt will be generated.

Table 13-1, Table 13-2, and Table 13-3 on page 145 summarize when interrupts can be triggered for the various input sense configurations.

Table 13-1. Synchronous sense support.

Sense settings	Supported	Interrupt description
Rising edge	Yes	Always triggered
Falling edge	Yes	Always triggered
Any edge	Yes	Always triggered
Low level	Yes	Pin level must be kept unchanged during wake up

Table 13-2. Full asynchronous sense support.

Sense settings	Supported	Interrupt description
Rising edge	Yes	Always triggered
Falling edge	Yes	Always triggered
Both edges	Yes	Always triggered
Low level	Yes	Pin level must be kept unchanged during wake up

Table 13-3. Limited asynchronous sense support.

Sense settings	Supported	Interrupt description
Rising edge	No	-
Falling edge	No	-
Any edge	Yes	Pin value must be kept unchanged during wake up
Low level	Yes	Pin level must be kept unchanged during wake up

13.7 Port Event

Port pins can generate an event when there is a change on the pin. The sense configurations decide the conditions for each pin to generate events. Event generation requires the presence of a peripheral clock, and asynchronous event generation is not possible. For edge sensing, the changed pin value must be sampled once by the peripheral clock for an event to be generated.

For level sensing, a low-level pin value will not generate events, and a high-level pin value will continuously generate events. For events to be generated on a low level, the pin configuration must be set to inverted I/O.

Table 13-4. Event sense support.

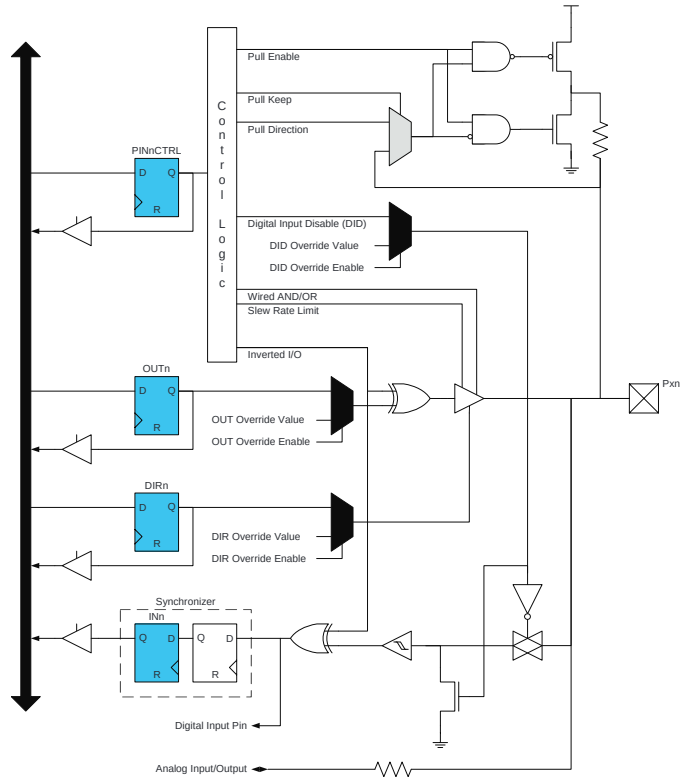
Sense settings	Signal event	Data event
Rising edge	Rising edge	Pin value
Falling edge	Falling edge	Pin value
Both edge	Any edge	Pin value
Low level	Pin value	Pin value

13.8 Alternate Port Functions

Most port pins have alternate pin functions in addition to being a general purpose I/O pin. When an alternate function is enabled, it might override the normal port pin function or pin value. This happens when other peripherals that require pins are enabled or configured to use pins. If and how a peripheral will override and use pins is described in the section for that peripheral.

The port override signals and related logic (grey) are shown in Figure 13-10 on page 146. These signals are not accessible from software, but are internal signals between the overriding peripheral and the port pin.

Figure 13-10. Port override signals and related logic.



13.9 Slew Rate Control

Slew rate control can be enabled for all I/O pins individually. Enabling the slew rate limiter will typically increase the rise/fall time by 50% to 150%, depending on operating conditions and load. For information about the characteristics of the slew rate limiter, please refer to the device datasheet.

13.10 Clock and Event Output

It is possible to output the peripheral clock and any of the event channels to the port pins (using EVCTRL register). This can be used to clock, control, and synchronize external functions and hardware to internal device timing. The output port pin is selectable. If an event occurs, it remains visible on the port pin as long as the event lasts; normally one peripheral clock cycle.

13.11 Multi-pin configuration

The multi-pin configuration function is used to configure multiple port pins using a single write operation to only one of the port pin configuration registers. A mask register decides which port pin is configured when one port pin register is written, while avoiding several pins being written the same way during identical write operations.

13.12 Virtual Ports

Virtual port registers allow the port registers to be mapped virtually in the bit-accessible I/O memory space. When this is done, writing to the virtual port register will be the same as writing to the real port register. This enables the use of I/O memory-specific instructions, such as bit-manipulation instructions, on a port register that normally resides in the extended I/O memory space. There are four virtual ports, and so four ports can be mapped at the same time.

13.13 Register Descriptions – Ports

13.13.1 DIR – Data Direction register

Bit	7	6	5	4	3	2	1	0
+0x00	DIR[7:0]							
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 7:0 – DIR[7:0]: Data Direction
This register sets the data direction for the individual pins of the port. If DIRn is written to one, pin n is configured as an output pin. If DIRn is written to zero, pin n is configured as an input pin.

13.13.2 DIRSET – Data Direction Set register

Bit	7	6	5	4	3	2	1	0
+0x01	DIRSET[7:0]							
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 7:0 – DIRSET[7:0]: Port Data Direction Set
This register can be used instead of a read-modify-write to set individual pins as output. Writing a one to a bit will set the corresponding bit in the DIR register. Reading this register will return the value of the DIR register.

13.13.3 DIRCLR – Data Direction Clear register

Bit	7	6	5	4	3	2	1	0
+0x02	DIRCLR[7:0]							
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 7:0 – DIRCLR[7:0]: Port Data Direction Clear
This register can be used instead of a read-modify-write to set individual pins as input. Writing a one to a bit will clear the corresponding bit in the DIR register. Reading this register will return the value of the DIR register.

13.13.4 DIRTGL – Data Direction Toggle register

Bit	7	6	5	4	3	2	1	0
+0x03	DIRTGL[7:0]							
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 7:0 – DIRTGL[7:0]: Port Data Direction Toggle
This register can be used instead of a read-modify-write to toggle the direction of individual pins. Writing a one to a bit will toggle the corresponding bit in the DIR register. Reading this register will return the value of the DIR register.

13.13.5 OUT – Data Output Value register

Bit	7	6	5	4	3	2	1	0
+0x04	OUT[7:0]							
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 7:0 – OUT[7:0]: Port Data Output value
This register sets the data output value for the individual pins of the port. If OUTn is written to one, pin n is driven high. If OUTn is written to zero, pin n is driven low. For this setting to have any effect, the pin direction must be set as output.

13.13.6 OUTSET – Data Output Value Set register

Bit	7	6	5	4	3	2	1	0
+0x05	OUTSET[7:0]							
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 7:0 – OUTSET[7:0]: Data Output Value Set
This register can be used instead of a read-modify-write to set the output value of individual pins to one. Writing a one to a bit will set the corresponding bit in the OUT register. Reading this register will return the value in the OUT register.

13.13.7 OUTCLR – Data Output Value Clear Register

Bit	7	6	5	4	3	2	1	0
+0x06	OUTCLR[7:0]							
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 7:0 – OUTCLR[7:0]: Data Output Value Clear
This register can be used instead of a read-modify-write to set the output value of individual pins to zero. Writing a one to a bit will clear the corresponding bit in the OUT register. Reading this register will return the value in the OUT register.

13.13.8 OUTTGL – Data Output Value Toggle register

Bit	7	6	5	4	3	2	1	0
+0x07	OUTTGL[7:0]							
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 7:0 – OUTTGL[7:0]: Port Data Output Value Toggle
This register can be used instead of a read-modify-write to toggle the output value of individual pins. Writing a one to a bit will toggle the corresponding bit in the OUT register. Reading this register will return the value in the OUT register.

13.13.9 IN – Data Input Value register

Bit	7	6	5	4	3	2	1	0
+0x08	IN[7:0]							
Read/Write	R	R	R	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

Bit 7:0 – IN[7:0]: Data Input Value
This register shows the value present on the pins if the digital input driver is enabled. INn shows the value of pin n of the port. The input is not sampled and cannot be read if the digital input buffers are disabled.

13.13.10 INTCTRL – Interrupt Control register

Bit	7	6	5	4	3	2	1	0
+0x09	–	–	–	–	INT1LVL[1:0]		INT0LVL[1:0]	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:4 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 3:2/1:0 – INTnLVL[1:0]: Interrupt n Level
These bits enable port interrupt n and select the interrupt level as described in "Interrupts and Programmable Multilevel Interrupt Controller" on page 131.

13.13.11 INT0MASK – Interrupt 0 Mask register

Bit	7	6	5	4	3	2	1	0
+0x0A	INT0MSK[7:0]							
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 7:0 – INT0MSK[7:0]: Interrupt 0 Mask Bits
These bits are used to mask which pins can be used as sources for port interrupt 0. If INT0MASKn is written to one, pin n is used as source for port interrupt 0.The input sense configuration for each pin is decided by the PINnCTRL registers.

13.13.12 INT1MASK – Interrupt 1 Mask register

Bit	7	6	5	4	3	2	1	0
+0x0B	INT1MSK[7:0]							
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 7:0 – INT1MASK[7:0]: Interrupt 1 Mask Bits
These bits are used to mask which pins can be used as sources for port interrupt 1. If INT1MASKn is written to one, pin n is used as source for port interrupt 1.The input sense configuration for each pin is decided by the PINnCTRL registers.

13.13.13 INTFLAGS – Interrupt Flag register

Bit	7	6	5	4	3	2	1	0
+0x0C	–	–	–	–	–	–	INT1IF	INT0IF
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 unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 1:0 – INTnIF: Interrupt n Flag
The INTnIF flag is set when a pin change/state matches the pin's input sense configuration, and the pin is set as source for port interrupt n. Writing a one to this flag's bit location will clear the flag. For enabling and executing the interrupt, refer to the interrupt level description.

13.13.14 REMAP – Pin Remap register

The pin remap functionality is available for PORTC - PORTF only.

Bit	7	6	5	4	3	2	1	0
+0x0E	–	–	SPI	USART0	TC0D	TC0C	TC0B	TC0A
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

Bit 7:6 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 5 – SPI: SPI Remap
Setting this bit to one will swap the pin locations of the SCK and MOSI pins to have pin compatibility between SPI and USART when the USART is operating as a SPI master.

Bit 4 – USART0: USART0 Remap
Setting this bit to one will move the pin location of USART0 from Px[3:0] to Px[7:4].

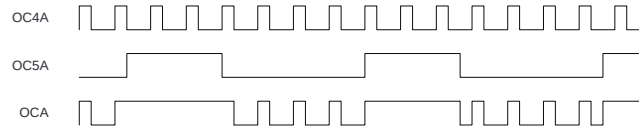
Bit 3 – TC0D: Timer/Counter 0 Output Compare D
Setting this bit will move the location of OC0D from Px3 to Px7.

Bit 2 – TC0C: Timer/Counter 0 Output Compare C
Setting this bit will move the location of OC0C from Px2 to Px6.

Bit 1 – TC0B: Timer/Counter 0 Output Compare B
Setting this bit will move the location of OC0B from Px1 to Px5. If this bit is set and PWM from both timer/counter 0 and timer/counter 1 is enabled, the resulting PWM will be an OR-modulation between the two PWM outputs.

Bit 0 – TC0A: Timer/Counter 0 Output Compare A
Setting this bit will move the location of OC0A from Px0 to Px4. If this bit is set and PWM from both timer/counter 0 and timer/counter 1 is enabled, the resulting PWM will be an OR-modulation between the two PWM outputs. See Figure 13-11.

Figure 13-11.I/O timer/counter.



13.13.15 PINnCTRL – Pin n Configuration register

Bit	7	6	5	4	3	2	1	0
	SRLEN	INVEN	OPC[2:0]			ISC[2:0]		
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 – SRLEN: Slew Rate Limit Enable
Setting this bit will enable slew rate limiting on pin n.

Bit 6 – INVEN: Inverted I/O Enable
Setting this bit will enable inverted output and input data on pin n.

Bit 5:3 – OPC: Output and Pull Configuration
These bits set the output/pull configuration on pin n according to Table 13-5 on page 152.

Table 13-5. Output/pull configuration.

OPC[2:0]	Group configuration	Description	
		Output configuration	Pull configuration
000	TOTEM	Totem-pole	(N/A)
001	BUSKEEPER	Totem-pole	Bus-keeper
010	PULLDOWN	Totem-pole	Pull-down (on input)
011	PULLUP	Totem-pole	Pull-up (on input)
100	WIREDOR	Wired-OR	(N/A)
101	WIREDAND	Wired-AND	(N/A)
110	WIREDORPULL	Wired-OR	Pull-down
111	WIREDANDPULL	Wired-AND	Pull-up

Bit 2:0 – ISC[2:0]: Input/Sense Configuration
These bits set the input and sense configuration on pin n according to Table 13-6. The sense configuration decides how the pin can trigger port interrupts and events. If the input buffer is disabled, the input cannot be read in the IN register.

Table 13-6. Input/sense configuration.

ISC[2:0]	Group configuration	Description
000	BOTHEDGES	Sense both edges
001	RISING	Sense rising edge
010	FALLING	Sense falling edge
011	LEVEL	Sense low level(1)
100		Reserved
101		Reserved
110		Reserved
111	INTPUT_DISABLE	Digital input buffer disabled(2)

Notes: 1. A low-level pin value will not generate events, and a high-level pin value will continuously generate events.
2. Only PORTA - PORTF support the input buffer disable option. If the pin is used for analog functionality, such as AC or ADC, it is recommended to configure the pin to INPUT_DISABLE.

13.14 Register Descriptions – Port Configuration

13.14.1 MPCMASK – Multi-pin Configuration Mask register

Bit	7	6	5	4	3	2	1	0
+0x00	MPCMASK[7:0]							
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 7:0 – MPCMASK[7:0]: Multi-pin Configuration Mask
The MPCMASK register enables configuration of several pins of a port at the same time. Writing a one to bit n makes pin n part of the multi-pin configuration. When one or more bits in the MPCMASK register is set, writing any of the PINnCTRL registers will update only the PINnCTRL registers matching the mask in the MPCMASK register for that port. The MPCMASK register is automatically cleared after any PINnCTRL register is written.

13.14.2 VPCTRLA – Virtual Port-map Control register A

Bit	7	6	5	4	3	2	1	0
+0x02	VP1MAP[3:0]				VP0MAP[3:0]			
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 7:4 – VP1MAP: Virtual Port 1 Mapping
These bits decide which ports should be mapped to Virtual Port 1. The registers DIR, OUT, IN, and INTFLAGS will be mapped. Accessing the virtual port registers is equal to accessing the actual port registers. See Table 13-7 on page 154 for configuration.

Bit 3:0 – VP0MAP: Virtual Port 0 Mapping
These bits decide which ports should be mapped to Virtual Port 0. The registers DIR, OUT, IN, and INTFLAGS will be mapped. Accessing the virtual port registers is equal to accessing the actual port registers. See Table 13-7 on page 154 for configuration.

13.14.3 VPCTRLB – Virtual Port-map Control register B

Bit	7	6	5	4	3	2	1	0
+0x03	VP3MAP[3:0]				VP2MAP[3:0]			
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 7:4 – VP3MAP: Virtual Port 3 Mapping
These bits decide which ports should be mapped to Virtual Port 3. The registers DIR, OUT, IN, and INTFLAGS will be mapped. Accessing the virtual port registers is equal to accessing the actual port registers. See Table 13-7 on page 154 for configuration.

Bit 3:0 – VP2MAP: Virtual Port 2 Mapping
These bits decide which ports should be mapped to Virtual Port 2. The registers DIR, OUT, IN, and INTFLAGS will be mapped. Accessing the virtual port registers is equal to accessing the actual port registers. See Table 13-7 on page 154 for configuration.

Table 13-7. Virtual port mapping.

VPnMAP[3:0]	Group configuration	Description
0000	PORTA	PORTA mapped to Virtual Port n
0001	PORTB	PORTB mapped to Virtual Port n
0010	PORTC	PORTC mapped to Virtual Port n
0011	PORTD	PORTD mapped to Virtual Port n
0100	PORTE	PORTE mapped to Virtual Port n
0101	PORTF	PORTF mapped to Virtual Port n
0110	PORTG	PORTG mapped to Virtual Port n
0111	PORTH	PORTH mapped to Virtual Port n
1000	PORTJ	PORTJ mapped to Virtual Port n
1001	PORTK	PORTK mapped to Virtual Port n
1010	PORTL	PORTL mapped to Virtual Port n
1011	PORTM	PORTM mapped to Virtual Port n
1100	PORTN	PORTN mapped to Virtual Port n
1101	PORTP	PORTP mapped to Virtual Port n
1110	PORTQ	PORTQ mapped to Virtual Port n
1111	PORTR	PORTR mapped to Virtual Port n

13.14.4 CLKEVOUT – Clock and Event Out register

Bit	7	6	5	4	3	2	1	0
+0x04	CLKEVPIN	RTCOUT	EVOUT[1:0]		CLKOUTSEL[1:0]		CLKOUT[1:0]	
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 – CLKEVPIN: Clock and Event Output Pin Select
Setting this pin enables output of clock and event pins on port pin 4 instead of port pin 7.

Bit 6 – RTCOUT: RTC Clock Output Enable
Setting this bit enables output of the RTC clock source on PORTC pin 6.

Bit 5:4 – EVOUT[1:0]: Event Output Port
These bits decide which port event channel 0 from the event system will be output to. Pin 7 on the selected port is the default used, and the CLKOUT bits must be set differently from those of EVOUT. The port pin must be configured as output for the event to be available on the pin.
Table 13-8 on page 155 shows the possible configurations.

Table 13-8. Event output pin selection.

EVOUT[1:0]	Group configuration	Description
00	OFF	Event output disabled
01	PC	Event channel 0 output on PORTC
10	PD	Event channel 0 output on PORTD
11	PE	Event channel 0 output on PORTE

Bits 3:2 – CLKOUTSEL[1:0]: Clock Output Select
These bits are used to select which of the peripheral clocks will be output to the port pin if CLKOUT is configured.

Table 13-9. Event output clock selection.

CLKOUTSEL[1:0]	Group configuration	Description
00	CLK1X	CLK_{PER} output to pin
01	CLK2X	CLK_{PER2} output to pin
10	CLK4X	CLK_{PER4} output to pin

Bit 1:0 – CLKOUT[1:0]: Clock Output Port
These bits decide which port the peripheral clock will be output to. Pin 7 on the selected port is the default used. The CLKOUT setting will override the EVOUT setting. Thus, if both are enabled on the same port pin, the peripheral clock will be visible. The port pin must be configured as output for the clock to be available on the pin.
Table 13-10 on page 155 shows the possible configurations.

Table 13-10. Clock output port configurations.

CLKOUT[1:0]	Group configuration	Description
00	OFF	Clock output disabled
01	PC	Clock output on PORTC
10	PD	Clock output on PORTD
11	PE	Clock output on PORTE

13.14.5 EBIOUT – EBI Output register

Bit	7	6	5	4	3	2	1	0
+0x05	–	–	–	–	EBIADROUT[1:0]		EBICSOUT[1:0]	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:4 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 3:2 – EBIADROUT[1:0]: EBI Address Output
The maximum configuration of the external bus interface (EBI) requires up to 32 dedicated pins. For devices with only 24 EBI pins available, eight additional pins can be enabled and placed on alternate pin locations in order to get a full 32-pin EBI. The port pins must be configured as output for signals to be available on the pins. These bits are available on devices with only three ports dedicated for the EBI interface. The selections are valid only if the EBI is configured to operate in four-port mode.

Table 13-11. EBI address output port selection.

EBIADROUT[1:0]	Group configuration	Description
00	PF	EBI port 3 address output on PORTF pins 0 to 7
01	PE	EBI port 3 address output on PORTE pins 0 to 7
10	PFH	EBI port 3 address output on PORTF pins 4 to 7
11	PEH	EBI port 3 address output on PORTE pins 4 to 7

Table 13-12. EBI address output

EBIADROUT	SDRAM	SRAM or SRAM LPC (with SDRAM on CS3)	SRAM NOALE or ALE1
00 or 01	4'h0, A[11:8]	A[23:16]	A[15:8]
10 or 11	A[11:8]	[19:16]	–

Bit 1:0 – EBICSOUT[1:0]: EBI Chip Select Output
These bits decide which port the EBI chip select signals will be output to. The pins must be configured as output pins for signals to be available on the pins. Refer to "Register Description – EBI" on page 329 for chip select configuration.

Table 13-13. EBI chip select port selection.

EBICSOUT[1:0]	Group configuration	Description
00	PH	EBI chip select output to PORTH pin 4 to 7
01	PL	EBI chip select output to PORTL pin 4 to 7
10	PF	EBI chip select output to PORTF pin 4 to 7
11	PE	EBI chip select output to PORTE pin 4 to 7

13.14.6 EVCTRL – Event Control register

Bit	7	6	5	4	3	2	1	0
+0x06	–	–	–	–	–	EVOUTSEL[2:0]		
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:3 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 2:0 – EVOUTSEL[2:0]: Event Channel Output Selection
These bits define which channel from the event system is output to the port pin. Table 13-14 on page 157 shows the available selections.

Table 13-14. Event channel output selection.

EVOUTSEL[2:0]	Group configuration	Description
000	0	Event channel 0 output to pin
001	1	Event channel 1 output to pin
010	2	Event channel 2 output to pin
011	3	Event channel 3 output to pin
100	4	Event channel 4 output to pin
101	5	Event channel 5 output to pin
110	6	Event channel 6 output to pin
111	7	Event channel 7 output to pin

13.15 Register Descriptions – Virtual Port

13.15.1 DIR – Data Direction register

Bit	7	6	5	4	3	2	1	0
+0x00	DIR[7:0]							
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 7:0 – DIR[7:0]: Data Direction
This register sets the data direction for the individual pins in the port mapped by VPCTRLA, virtual port-map control register A or VPCTRLB, virtual port-map control register B. When a port is mapped as virtual, accessing this register is identical to accessing the actual DIR register for the port.

13.15.2 OUT – Data Output Value register

Bit	7	6	5	4	3	2	1	0
+0x01	OUT[7:0]							
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 7:0 – OUT[7:0]: Data Output value
This register sets the data output value for the individual pins in the port mapped by VPCTRLA, virtual port-map control register A or VPCTRLB, virtual port-map control register B. When a port is mapped as virtual, accessing this register is identical to accessing the actual OUT register for the port.

13.15.3 IN – Data Input Value register

Bit	7	6	5	4	3	2	1	0
+0x02	IN[7:0]							
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 7:0 – IN[7:0]: Data Input Value
This register shows the value present on the pins if the digital input buffer is enabled. The configuration of VPCTRLA, virtual port-map control register A or VPCTRLB, virtual port-map control register A, decides the value in the register. When a port is mapped as virtual, accessing this register is identical to accessing the actual IN register for the port.

13.15.4 INTFLAGS – Interrupt Flag register

Bit	7	6	5	4	3	2	1	0
+0x03	–	–	–	–	–	–	INT1IF	INT0IF
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 unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 1:0 – INTnIF: Interrupt n Flag

The INTnIF flag is set when a pin change/state matches the pin's input sense configuration, and the pin is set as source for port interrupt n. Writing a one to this flag's bit location will clear the flag. For enabling and executing the interrupt, refer to the interrupt level description. The configuration of VPCTRLA, virtual port-map control register A, or VPCTRLB, Virtual Port-map Control Register B,, decides which flags are mapped. When a port is mapped as virtual, accessing this register is identical to accessing the actual INTFLAGS register for the port.

13.16 Register summary – Ports

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	DIR	DIR[7:0]								148
+0x01	DIRSET	DIRSET[7:0]								148
+0x02	DIRCLR	DIRCLR[7:0]								148
+0x03	DIRTGL	DIRTGL[7:0]								148
+0x04	OUT	OUT[7:0]								149
+0x05	OUTSET	OUTSET[7:0]								149
+0x06	OUTCLR	OUTCLR[7:0]								149
+0x07	OUTTGL	OUTTGL[7:0]								149
+0x08	IN	IN[7:0]								150
+0x09	INTCTRL	–	–	–	–	INT1LVL[1:0]		INT0LVL[1:0]		150
+0x0A	INT0MASK	INT0MSK[7:0]								150
+0x0B	INT1MASK	INT1MSK[7:0]								150
+0x0C	INTFLAGS	–	–	–	–	–	–	INT1IF	INT0IF	151
+0x0D	Reserved	–	–	–	–	–	–	–	–	
+0x0E	REMAP	–	–	SPI	USART0	TC0D	TC0C	TC0B	TC0A	151
+0x0F	Reserved	–	–	–	–	–	–	–	–	
+0x10	PIN0CTRL	SRLEN	INVEN	OPC[2:0]			ISC[2:0]			152
+0x11	PIN1CTRL	SRLEN	INVEN	OPC[2:0]			ISC[2:0]			152
+0x12	PIN2CTRL	SRLEN	INVEN	OPC[2:0]			ISC[2:0]			152
+0x13	PIN3CTRL	SRLEN	INVEN	OPC[2:0]			ISC[2:0]			152
+0x14	PIN4CTRL	SRLEN	INVEN	OPC[2:0]			ISC[2:0]			152
+0x15	PIN5CTRL	SRLEN	INVEN	OPC[2:0]			ISC[2:0]			152
+0x16	PIN6CTRL	SRLEN	INVEN	OPC[2:0]			ISC[2:0]			152
+0x17	PIN7CTRL	SRLEN	INVEN	OPC[2:0]			ISC[2:0]			152
+0x18	Reserved	–	–	–	–	–	–	–	–	
+0x19	Reserved	–	–	–	–	–	–	–	–	
+0x1A	Reserved	–	–	–	–	–	–	–	–	
+0x1B	Reserved	–	–	–	–	–	–	–	–	
+0x1C	Reserved	–	–	–	–	–	–	–	–	
+0x1D	Reserved	–	–	–	–	–	–	–	–	
+0x1E	Reserved	–	–	–	–	–	–	–	–	
+0x1F	Reserved	–	–	–	–	–	–	–	–	

13.17 Register summary – Port Configuration

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	bit 0	Page
+0x00	MPCMASK	MPCMASK[7:0]								153
+0x01	Reserved	–	–	–	–	–	–	–	–	
+0x02	VPCTRLA	VP1MAP[3:0]				VP0MAP[3:0]				153
+0x03	VPCTRLB	VP3MAP[3:0]				VP2MAP[3:0]				154
+0x04	CLKEVOUT	CLKEVPIN	RTCOUT	EVOUT[1:0]		CLKOUTSEL		CLKOUT[1:0]		154
+0x05	EBIOUT	–	–	–	–	EBIADROUT[1:0]		EBICSOUT[1:0]		156
+0x06	EVCTRL	–	–	–	–	–	EVCTRL[2:0]			157

13.18 Register summary – Virtual Ports

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	bit 0	Page
+0x00	DIR	DIR[7:0]								158
+0x01	OUT	OUT[7:0]								158
+0x02	IN	IN[7:0]								158
+0x03	INTFLAGS	–	–	–	–	–	–	INT1IF	INT0IF	158

13.19 Interrupt vector summary – Ports

Table 13-15. Port interrupt vectors and their word offset address.

Offset	Source	Interrupt description
0x00	INT0_vect	Port interrupt vector 0 offset
0x02	INT1_vect	Port interrupt vector 1 offset

14. TC0/1 – 16-bit Timer/Counter Type 0 and 1

14.1 Features

- 16-bit timer/counter
- 32-bit timer/counter support by cascading two timer/counters
- Up to four compare or capture (CC) channels
  - Four CC channels for timer/counters of type 0
  - Two CC channels for timer/counters of type 1
- Double buffered timer period setting
- Double buffered capture or compare channels
- Waveform generation:
  - Frequency generation
  - Single-slope pulse width modulation
  - Dual-slope pulse width modulation
- Input capture:
  - Input capture with noise cancelling
  - Frequency capture
  - Pulse width capture
  - 32-bit input capture
- Timer overflow and error interrupts/events
- One compare match or input capture interrupt/event per CC channel
- Can be used with event system for:
  - Quadrature decoding
  - Count and direction control
  - Capture
- Can be used with DMA and to trigger DMA transactions
- High-resolution extension
  - Increases frequency and waveform resolution by 4x (2-bit) or 8x (3-bit)
- Advanced waveform extension:
  - Low- and high-side output with programmable dead-time insertion (DTI)
  - Event controlled fault protection for safe disabling of drivers

14.2 Overview

Atmel AVR XMEGA devices have a set of flexible, 16-bit timer/counters (TC). Their capabilities include accurate program execution timing, frequency and waveform generation, and input capture with time and frequency measurement of digital signals. Two timer/counters can be cascaded to create a 32-bit timer/counter with optional 32-bit capture.

A timer/counter consists of a base counter and a set of compare or capture (CC) channels. The base counter can be used to count clock cycles or events. It has direction control and period setting that can be used for timing. The CC channels can be used together with the base counter to do compare match control, frequency generation, and pulse width waveform modulation, as well as various input capture operations. A timer/counter can be configured for either capture or compare functions, but cannot perform both at the same time.

A timer/counter can be clocked and timed from the peripheral clock with optional prescaling or from the event system. The event system can also be used for direction control and capture trigger or to synchronize operations.

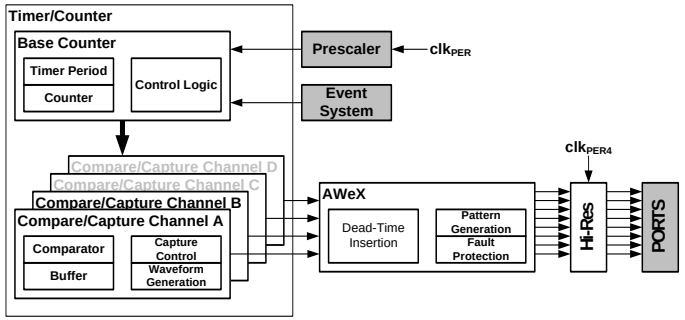
There are two differences between timer/counter type 0 and type 1. Timer/counter 0 has four CC channels, and timer/counter 1 has two CC channels. All information related to CC channels 3 and 4 is valid only for timer/counter 0. Only Timer/Counter 0 has the split mode feature that split it into 2 8-bit Timer/Counters with four compare channels each.

Some timer/counters have extensions to enable more specialized waveform and frequency generation. The advanced waveform extension (AWeX) is intended for motor control and other power control applications. It enables low- and high-

side output with dead-time insertion, as well as fault protection for disabling and shutting down external drivers. It can also generate a synchronized bit pattern across the port pins. The high-resolution (hi-res) extension can be used to increase the waveform output resolution by four or eight times by using an internal clock source running up to four times faster than the peripheral clock.

A block diagram of the 16-bit timer/counter with extensions and closely related peripheral modules (in grey) is shown in Figure 14-1 on page 163.

Figure 14-1. 16-bit timer/counter and closely related peripherals.



14.2.1 Definitions

The following definitions are used throughout the documentation:

Table 14-1. Timer/counter definitions.

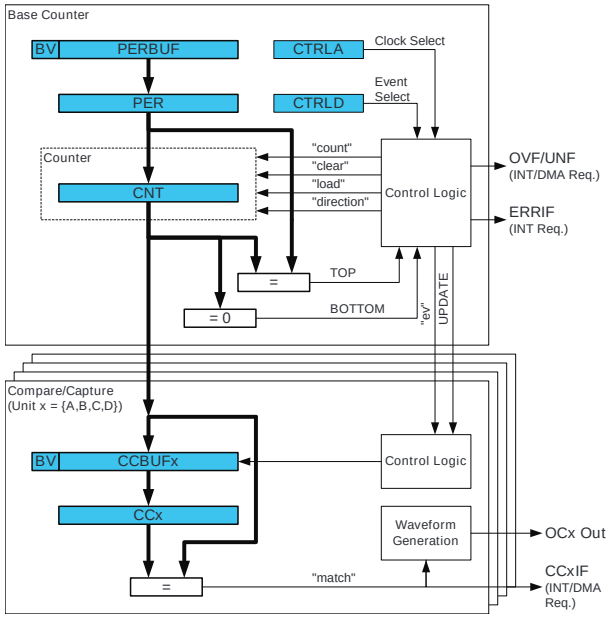
Name	Description
BOTTOM	The counter reaches BOTTOM when it becomes zero.
MAX	The counter reaches MAXimum when it becomes all ones.
TOP	The counter reaches TOP when it becomes equal to the highest value in the count sequence. The TOP value can be equal to the period (PER) or the compare channel A (CCA) register setting. This is selected by the waveform generator mode.
UPDATE	The timer/counter signals an update when it reaches BOTTOM or TOP, depending on the waveform generator mode.

In general, the term “timer” is used when the timer/counter clock control is handled by an internal source, and the term “counter” is used when the clock control is handled externally (e.g. counting external events). When used for compare operations, the CC channels are referred to as “compare channels.” When used for capture operations, the CC channels are referred to as “capture channels.”

14.3 Block Diagram

Figure 14-2 on page 164 shows a detailed block diagram of the timer/counter without the extensions.

Figure 14-2. Timer/counter block diagram.



The counter register (CNT), period registers with buffer (PER and PERBUF), and compare and capture registers with buffers (CCx and CCxBUF) are 16-bit registers. All buffer register have a buffer valid (BV) flag that indicates when the buffer contains a new value.

During normal operation, the counter value is continuously compared to zero and the period (PER) value to determine whether the counter has reached TOP or BOTTOM.

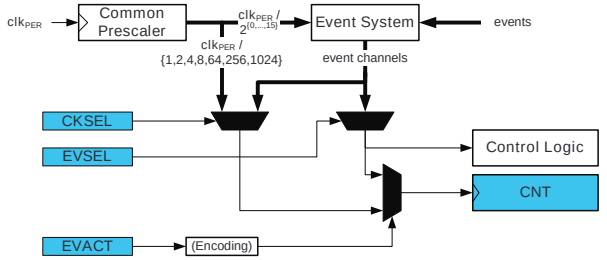
The counter value is also compared to the CCx registers. These comparisons can be used to generate interrupt requests, request DMA transactions or generate events for the event system. The waveform generator modes use these comparisons to set the waveform period or pulse width.

A prescaled peripheral clock and events from the event system can be used to control the counter. The event system is also used as a source to the input capture. Combined with the quadrature decoding functionality in the event system (QDEC), the timer/counter can be used for quadrature decoding.

14.4 Clock and Event Sources

The timer/counter can be clocked from the peripheral clock (clk_{PER}) or the event system, and Figure 14-3 shows the clock and event selection.

Figure 14-3. Clock and event selection.



The peripheral clock is fed into a common prescaler (common for all timer/counters in a device). Prescaler outputs from 1 to 1/1024 are directly available for selection by the timer/counter. In addition, the whole range of prescaling from 1 to 2^{15} times is available through the event system.

Clock selection (CLKSEL) selects one of the prescaler outputs directly or an event channel as the counter (CNT) input. This is referred to as normal operation of the counter. For details, refer to "Normal Operation" on page 166. By using the event system, any event source, such as an external clock signal on any I/O pin, may be used as the clock input.

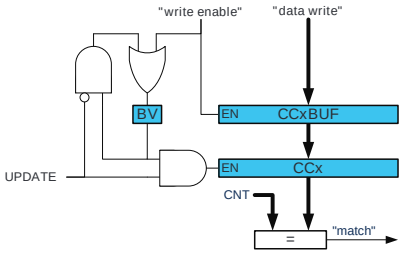
In addition, the timer/counter can be controlled via the event system. The event selection (EVSEL) and event action (EVACT) settings are used to trigger an event action from one or more events. This is referred to as event action controlled operation of the counter. For details, refer to "Event Action Controlled Operation" on page 167. When event action controlled operation is used, the clock selection must be set to use an event channel as the counter input.

By default, no clock input is selected and the timer/counter is not running.

14.5 Double Buffering

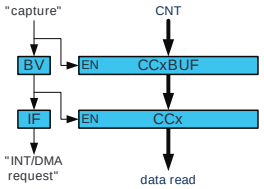
The period register and the CC registers are all double buffered. Each buffer register has a buffer valid (BV) flag, which indicates that the buffer register contains a valid, i.e. new, value that can be copied into the corresponding period or CC register. When the period register and CC channels are used for a compare operation, the buffer valid flag is set when data is written to the buffer register and cleared on an UPDATE condition. This is shown for a compare register in Figure 14-4 on page 166.

Figure 14-4. Period and compare double buffering.



When the CC channels are used for a capture operation, a similar double buffering mechanism is used, but in this case the buffer valid flag is set on the capture event, as shown in Figure 14-5. For capture, the buffer register and the corresponding CCx register act like a FIFO. When the CC register is empty or read, any content in the buffer register is passed to the CC register. The buffer valid flag is passed to set the CCx interrupt flag (IF) and generate the optional interrupt.

Figure 14-5. Capture double buffering.



Both the CCx and CCxBUF registers are available as an I/O register. This allows initialization and bypassing of the buffer register and the double buffering function.

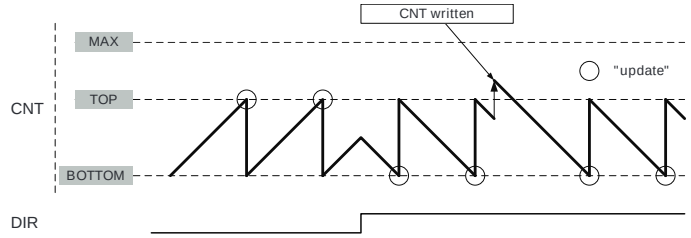
14.6 Counter Operation

Depending on the mode of operation, the counter is cleared, reloaded, incremented, or decremented at each timer/counter clock input.

14.6.1 Normal Operation

In normal operation, the counter will count in the direction set by the direction (DIR) bit for each clock until it reaches TOP or BOTTOM. When up-counting and TOP is reached, the counter will be set to zero when the next clock is given. When down-counting, the counter is reloaded with the period register value when BOTTOM is reached.

Figure 14-6. Normal operation.



As shown in Figure 14-6, it is possible to change the counter value when the counter is running. The write access has higher priority than count, clear, or reload, and will be immediate. The direction of the counter can also be changed during normal operation.

Normal operation must be used when using the counter as timer base for the capture channels.

14.6.2 Event Action Controlled Operation

The event selection and event action settings can be used to control the counter from the event system. For the counter, the following event actions can be selected:

- Event system controlled up/down counting
  - Event n will be used as count enable
  - Event n+1 will be used to select between up (1) and down (0). The pin configuration must be set to low level sensing
- Event system controlled quadrature decode counting

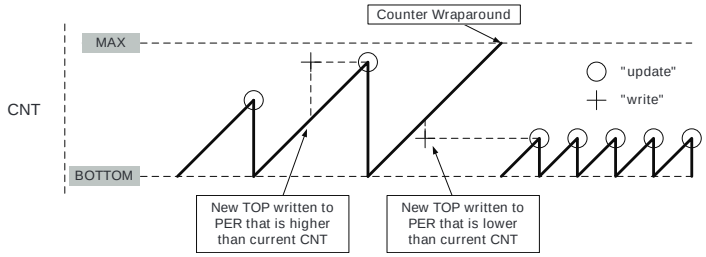
14.6.3 32-bit Operation

Two timer/counters can be used together to enable 32-bit counter operation. By using two timer/counters, the overflow event from one timer/counter (least-significant timer) can be routed via the event system and used as the clock input for another timer/counter (most-significant timer).

14.6.4 Changing the Period

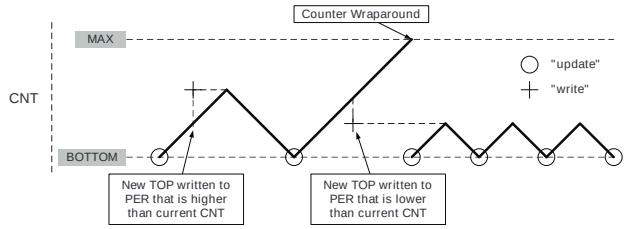
The counter period is changed by writing a new TOP value to the period register. If double buffering is not used, any period update is immediate, as shown in Figure 14-7 on page 167.

Figure 14-7. Changing the period without buffering.



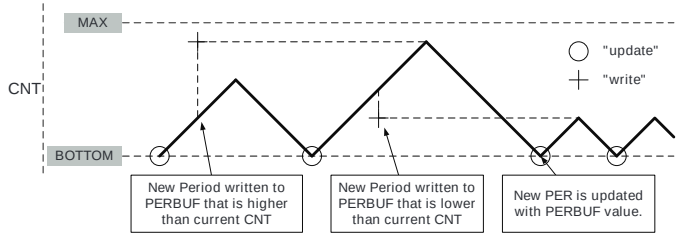
A counter wraparound can occur in any mode of operation when up-counting without buffering, as shown in Figure 14-8. This due to the fact that CNT and PER are continuously compared, and if a new TOP value that is lower than current CNT is written to PER, it will wrap before a compare match happen.

Figure 14-8. Unbuffered dual-slope operation.



When double buffering is used, the buffer can be written at any time and still maintain correct operation. The period register is always updated on the UPDATE condition, as shown for dual-slope operation in Figure 14-9. This prevents wraparound and the generation of odd waveforms.

Figure 14-9. Changing the period using buffering.

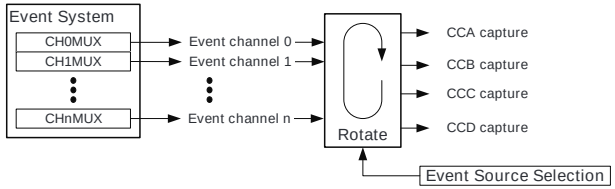


14.7 Capture Channel

The CC channels can be used as capture channels to capture external events and give them a timestamp. To use capture, the counter must be set for normal operation.

Events are used to trigger the capture; i.e., any events from the event system, including pin change from any pin, can trigger a capture operation. The event source select setting selects which event channel will trigger CC channel A. The subsequent event channels then trigger events on subsequent CC channels, if configured. For example, setting the event source select to event channel 2 results in CC channel A being triggered by event channel 2, CC channel B triggered by event channel 3, and so on.

Figure 14-10.Event source selection for capture operation.



The event action setting in the timer/counter will determine the type of capture that is done.

The CC channels must be enabled individually before capture can be done. When the capture condition occur, the timer/counter will time-stamp the event by copying the current CNT value in the count register into the enabled CC channel register.

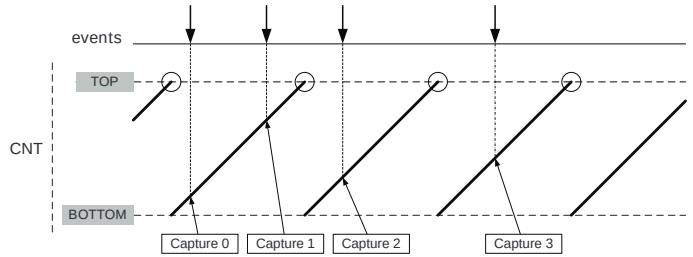
When an I/O pin is used as an event source for the capture, the pin must be configured for edge sensing. For details on sense configuration on I/O pins, refer to "Input Sense Configuration" on page 143. If the period register value is lower than 0x8000, the polarity of the I/O pin edge will be stored in the most-significant bit (msb) of the capture register. If the msb of the capture register is zero, a falling edge generated the capture. If the msb is one, a rising edge generated the capture.

14.7.1 Input Capture

Selecting the input capture event action makes the enabled capture channel perform an input capture on an event. The interrupt flags will be set and indicate that there is a valid capture result in the corresponding CC register. At the same time, the buffer valid flags indicate valid data in the buffer registers.

The counter will continuously count from BOTTOM to TOP, and then restart at BOTTOM, as shown in Figure 14-11. The figure also shows four capture events for one capture channel.

Figure 14-11.Input capture timing.



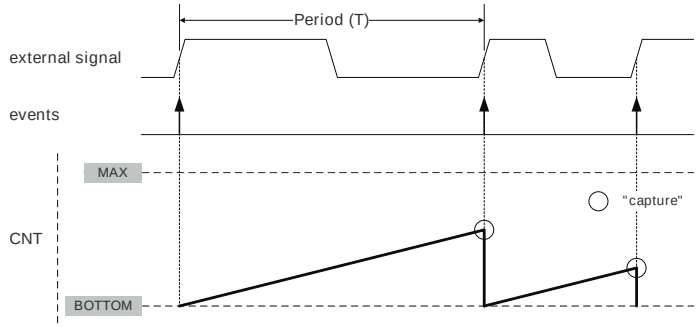
14.7.2 Frequency Capture

Selecting the frequency capture event action makes the enabled capture channel perform an input capture and restart on positive edge events. This enables the timer/counter to measure the period or frequency of a signal directly. The capture result will be the time (T) from the previous timer/counter restart until the event occurred. This can be used to calculate the frequency (f) of the signal:

$$f = \frac{1}{T}$$

Figure 14-12 on page 170 shows an example where the period of an external signal is measured twice.

Figure 14-12.Frequency capture of an external signal.

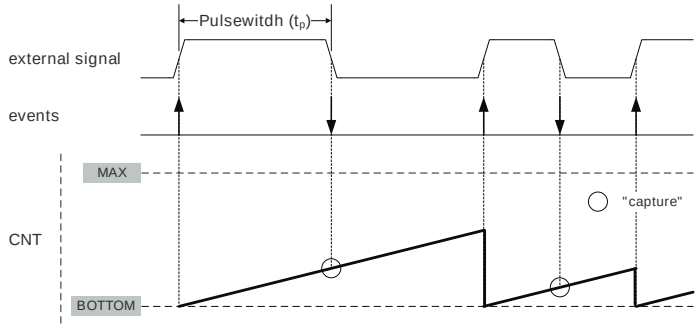


Since all capture channels use the same counter (CNT), only one capture channel must be enabled at a time. If two capture channels are used with different sources, the counter will be restarted on positive edge events from both input sources, and the result will have no meaning.

14.7.3 Pulse Width Capture

Selecting the pulse width measure event action makes the enabled compare channel perform the input capture action on falling edge events and the restart action on rising edge events. The counter will then restart on positive edge events, and the input capture will be performed on the negative edge event. The event source must be an I/O pin, and the sense configuration for the pin must be set to generate an event on both edges. Figure 14-13 on page 170 shows and example where the pulse width is measured twice for an external signal.

Figure 14-13.Pulse width capture of an external signal.



14.7.4 32-bit Input Capture

Two timer/counters can be used together to enable true 32-bit input capture. In a typical 32-bit input capture setup, the overflow event of the least-significant timer is connected via the event system and used as the clock input for the most-significant timer.

The most-significant timer will be updated one peripheral clock period after an overflow occurs for the least-significant timer. To compensate for this, the capture event for the most-significant timer must be equally delayed by setting the event delay bit for this timer.

14.7.5 Capture Overflow

The timer/counter can detect buffer overflow of the input capture channels. When both the buffer valid flag and the capture interrupt flag are set and a new capture event is detected, there is nowhere to store the new timestamp. If a buffer overflow is detected, the new value is rejected, the error interrupt flag is set, and the optional interrupt is generated.

14.8 Compare Channel

Each compare channel continuously compares the counter value (CNT) with the CCx register. If CNT equals CCx, the comparator signals a match. The match will set the CC channel's interrupt flag at the next timer clock cycle, and the event and optional interrupt are generated.

The compare buffer register provides double buffer capability equivalent to that for the period buffer. The double buffering synchronizes the update of the CCx register with the buffer value to either the TOP or BOTTOM of the counting sequence according to the UPDATE condition. The synchronization prevents the occurrence of odd-length, non-symmetrical pulses for glitch-free output.

14.8.1 Waveform Generation

The compare channels can be used for waveform generation on the corresponding port pins. To make the waveform visible on the connected port pin, the following requirements must be fulfilled:

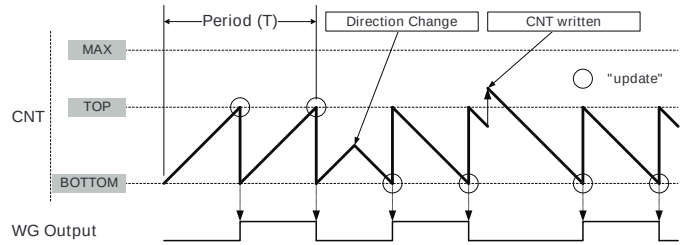
1. A waveform generation mode must be selected.
2. Event actions must be disabled.
3. The CC channels used must be enabled. This will override the corresponding port pin output register.
4. The direction for the associated port pin must be set to output.

Inverted waveform output is achieved by setting the invert output bit for the port pin.

14.8.2 Frequency (FRQ) Waveform Generation

For frequency generation the period time (T) is controlled by the CCA register instead of PER. The waveform generation (WG) output is toggled on each compare match between the CNT and CCA registers, as shown in Figure 14-14 on page 172.

Figure 14-14.Frequency waveform generation.



The waveform frequency (f_{FRQ}) is defined by the following equation:

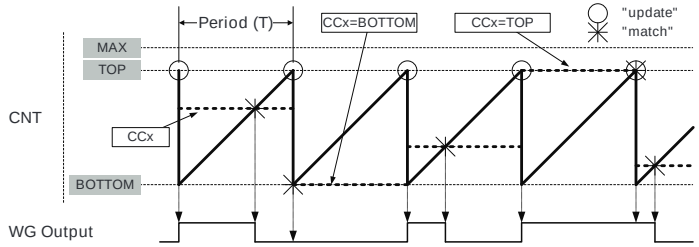
$$f_{FRQ} = \frac{fclk_{PER}}{2N(CCA + 1)}$$

where N represents the prescaler divider used. The waveform generated will have a maximum frequency of half of the peripheral clock frequency ($fclk_{PER}$) when CCA is set to zero (0x0000) and no prescaling is used. This also applies when using the hi-res extension, since this increases the resolution and not the frequency.

14.8.3 Single-slope PWM Generation

For single-slope PWM generation, the period (T) is controlled by PER, while CCx registers control the duty cycle of the WG output. Figure 14-15 shows how the counter counts from BOTTOM to TOP and then restarts from BOTTOM. The waveform generator (WG) output is set on the compare match between the CNT and CCx registers and cleared at TOP.

Figure 14-15.Single-slope pulse width modulation.



The PER register defines the PWM resolution. The minimum resolution is 2 bits (PER=0x0003), and the maximum resolution is 16 bits (PER=MAX).

The following equation calculate the exact resolution for single-slope PWM (R_{PWM_SS}):

$$R_{PWM\_SS} = \frac{\log(PER + 1)}{\log(2)}$$

The single-slope PWM frequency (f_{PWM_SS}) depends on the period setting (PER) and the peripheral clock frequency ($fclk_{PER}$), and can be calculated by the following equation:

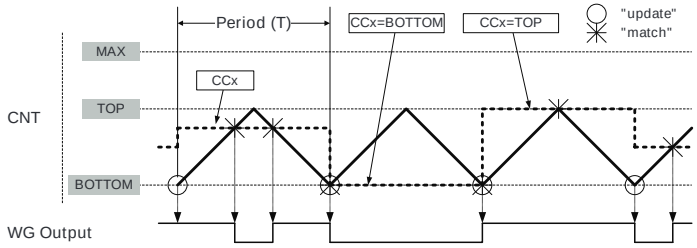
$$f_{PWM\_SS} = \frac{fclk_{PER}}{N(PER + 1)}$$

where N represents the prescaler divider used.

14.8.4 Dual-slope PWM

For dual-slope PWM generation, the period (T) is controlled by PER, while CCx registers control the duty cycle of the WG output. Figure 14-16 shows how for dual-slope PWM the counter counts repeatedly from BOTTOM to TOP and then from TOP to BOTTOM. The waveform generator output is set on BOTTOM, cleared on compare match when up-counting, and set on compare match when down-counting.

Figure 14-16.Dual-slope pulse width modulation.



Using dual-slope PWM results in a lower maximum operation frequency compared to the single-slope PWM operation.

The period register (PER) defines the PWM resolution. The minimum resolution is 2 bits (PER=0x0003), and the maximum resolution is 16 bits (PER=MAX).

The following equation calculate the exact resolution for dual-slope PWM (R_{PWM_DS}):

$$R_{PWM\_DS} = \frac{\log(PER + 1)}{\log(2)}$$

The PWM frequency depends on the period setting (PER) and the peripheral clock frequency ($fclk_{PER}$), and can be calculated by the following equation:

$$f_{PWM\_DS} = \frac{fclk_{PER}}{2NPER}$$

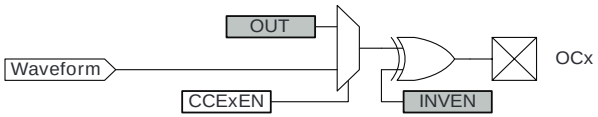
N represents the prescaler divider used.

14.8.5 Port Override for Waveform Generation

To make the waveform generation available on the port pins, the corresponding port pin direction must be set as output. The timer/counter will override the port pin values when the CC channel is enabled (CCENx) and a waveform generation mode is selected.

Figure 14-17 on page 174 shows the port override for a timer/counter. The timer/counter CC channel will override the port pin output value (OUT) on the corresponding port pin. Enabling inverted I/O on the port pin (INVEN) inverts the corresponding WG output.

Figure 14-17.Port override for timer/counter 0 and 1.



14.9 Interrupts and events

The timer/counter can generate both interrupts and events. The counter can generate an interrupt on overflow/underflow, and each CC channel has a separate interrupt that is used for compare or capture. In addition, an error interrupt can be generated if any of the CC channels is used for capture and a buffer overflow condition occurs on a capture channel.

Events will be generated for all conditions that can generate interrupts. For details on event generation and available events, refer to "Event System" on page 70.

14.10 DMA Support

The interrupt flags can be used to trigger DMA transactions. Table 14-2 on page 174 lists the transfer triggers available from the timer/counter and the DMA action that will clear the transfer trigger. For more details on using DMA, refer to "DMAC - Direct Memory Access Controller" on page 53.

Table 14-2. DMA request sources.

Request	Acknowledge	Comment
OVFIF/UNFIF	DMA controller writes to CNT DMA controller writes to PER DMA controller writes to PERBUF DMA controller writes to DTHSBUF or DTLSBUF in AWex in Pattern generation mode	
ERRIF	N/A	
CCxIF	DMA controller access of CCx DMA controller access of CCxBUF	Input capture operation Output compare operation

14.11 Timer/Counter Commands

A set of commands can be given to the timer/counter by software to immediately change the state of the module. These commands give direct control of the UPDATE, RESTART, and RESET signals.

An update command has the same effect as when an update condition occurs. The update command is ignored if the lock update bit is set.

The software can force a restart of the current waveform period by issuing a restart command. In this case the counter, direction, and all compare outputs are set to zero.

A reset command will set all timer/counter registers to their initial values. A reset can be given only when the timer/counter is not running (OFF).

14.12 Register Description

14.12.1 CTRLA – Control register A

Bit	7	6	5	4	3	2	1	0
+0x00	–	–	–	–	CLKSEL[3:0]			
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:4 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 3:0 – CLKSEL[3:0]: Clock Select
These bits select the clock source for the timer/counter according to Table 14-3.

CLKSEL=0001 must be set to ensure a correct output from the waveform generator when the hi-res extension is enabled.

Table 14-3. Clock select options.

CLKSEL[3:0]	Group configuration	Description
0000	OFF	None (i.e, timer/counter in OFF state)
0001	DIV1	Prescaler: Clk
0010	DIV2	Prescaler: Clk/2
0011	DIV4	Prescaler: Clk/4
0100	DIV8	Prescaler: Clk/8
0101	DIV64	Prescaler: Clk/64
0110	DIV256	Prescaler: Clk/256
0111	DIV1024	Prescaler: Clk/1024
1nnn	EVCHn	Event channel n, n= [0,...,7]

14.12.2 CTRLB – Control register B

Bit	7	6	5	4	3	2	1	0
+0x01	CCDEN	CCCEN	CCBEN	CCAEN	–	WGMODE[2:0]		
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:4 – CCxEN: Compare or Capture Enable
Setting these bits in the FRQ or PWM waveform generation mode of operation will override the port output register for the corresponding OCn output pin.

When input capture operation is selected, the CCxEN bits enable the capture operation for the corresponding CC channel.

Bit 3 – Reserved
This bit is unused and reserved for future use. For compatibility with future devices, always write this bit to zero when this register is written.

Bit 2:0 – WGMODE[2:0]: Waveform Generation Mode
These bits select the waveform generation mode, and control the counting sequence of the counter, TOP value, UPDATE condition, interrupt/event condition, and type of waveform that is generated according to Table 14-4 on page 176.

No waveform generation is performed in the normal mode of operation. For all other modes, the result from the waveform generator will only be directed to the port pins if the corresponding CCxEN bit has been set to enable this. The port pin direction must be set as output

Table 14-4. Timer waveform generation mode.

WGMODE[2:0]	Group configuration	Mode of operation	Top	Update	OVFIF/Event
000	NORMAL	Normal	PER	TOP	TOP
001	FRQ	Frequency	CCA	TOP	TOP
010		Reserved	–	–	–
011	SINGLESLOPE	Single-slope PWM	PER	BOTTOM	BOTTOM
100		Reserved	–	–	–
101	DSTOP	Dual-slope PWM	PER	BOTTOM	TOP
110	DSBOTH	Dual-slope PWM	PER	BOTTOM	TOP and BOTTOM
111	DSBOTTOM	Dual-slope PWM	PER	BOTTOM	BOTTOM

14.12.3 CTRLC – Control register C

Bit	7	6	5	4	3	2	1	0
+0x02	–	–	–	–	CMPD	CMPC	CMPB	CMPA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:4 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 3:0 – CMPx: Compare Output Value x
These bits allow direct access to the waveform generator's output compare value when the timer/counter is set in the OFF state. This is used to set or clear the WG output value when the timer/counter is not running.

14.12.4 CTRLD – Control register D

Bit	7	6	5	4	3	2	1	0
+0x03	EVACT[2:0]			EVDLY	EVSEL[3:0]			
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 7:5 – EVACT[2:0]: Event Action
These bits define the event action the timer will perform on an event according to Table 14-5 on page 177.

The EVSEL setting will decide which event source or sources have control in this case.

Table 14-5. Timer event action selection.

EVACT[2:0]	Group configuration	Event action
000	OFF	None
001	CAPT	Input capture
010	UPDOWN	Externally controlled up/ down count
011	QDEC	Quadrature decode
100	RESTART	Restart waveform period
101	FRQ	Frequency capture
110	PW	Pulse width capture
111		Reserved

Selecting any of the capture event actions changes the behavior of the CCx registers and related status and control bits to be used for capture. The error status flag (ERRIF) will indicate a buffer overflow in this configuration. See "Event Action Controlled Operation" on page 167 for further details.

Bit 4 – EVDLY: Timer Delay Event
When this bit is set, the selected event source is delayed by one peripheral clock cycle. This is intended for 32-bit input capture operation. Adding the event delay is necessary to compensate for the carry propagation delay when cascading two counters via the event system.

Bit 3:0 – EVSEL[3:0]:Timer Event Source Select
These bits select the event channel source for the timer/counter. For the selected event channel to have any effect, the event action bits (EVACT) must be set according to Table 14-6 on page 177. When the event action is set to a capture operation, the selected event channel n will be the event channel source for CC channel A, and event channel (n+1)%8, (n+2)%8, and (n+3)%8 will be the event channel source for CC channel B, C, and D.

Table 14-6. Timer event source selection.

EVSEL[3:0]	Group configuration	Event source
0000	OFF	None
0001		Reserved
0010		Reserved
0011		Reserved
0100		Reserved
0101		Reserved
0110		Reserved
0111		Reserved
1nnn	CHn	Event channel n, n={0,...,7}

14.12.5 CTRLE – Control register E

Bit	7	6	5	4	3	2	1	0
+0x04	–	–	–	–	–	–	BYTEM[1:0]	
Read/Write	R	R	R	R	R	R	R	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:2 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 1:0 – BYTEM[1:0]: Byte Mode
These bits select the timer/counter operation mode according to Table 14-7 on page 178.

Table 14-7. Clock select.

BYTEM[1:0]	Group configuration	Description
00	NORMAL	Timer/counter is set to normal mode (timer/counter type 0)
01	BYTEMODE	Upper byte of the counter (CNTH) will be set to zero after each counter clock cycle
10	SPLITMODE	Timer/counter 0 is split into two 8-bit timer/counters (timer/counter type 2)
11		Reserved

14.12.6 INTCTRLA – Interrupt Enable register A

Bit	7	6	5	4	3	2	1	0
+0x06	–	–	–	–	ERRINTLVL[1:0]		OVFINTLVL[1:0]	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:4 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 3:2 – ERRINTLVL[1:0]:Timer Error Interrupt Level
These bits enable the timer error interrupt and select the interrupt level as described in "Interrupts and Programmable Multilevel Interrupt Controller" on page 131.

Bit 1:0 – OVFINTLVL[1:0]:Timer Overflow/Underflow Interrupt Level
These bits enable the timer overflow/underflow interrupt and select the interrupt level as described in "Interrupts and Programmable Multilevel Interrupt Controller" on page 131.

14.12.7 INTCTRLB – Interrupt Enable register B

Bit	7	6	5	4	3	2	1	0
+0x07	CCDINTLVL[1:0]		CCCINTLVL[1:0]		CCBINTLVL[1:0]		CCAINTLVL[1:0]	
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 7:0 – CCxINTLVL[7:0] - Compare or Capture x Interrupt Level:
These bits enable the timer compare or capture interrupt for channel x and select the interrupt level as described in "Interrupts and Programmable Multilevel Interrupt Controller" on page 131.

14.12.8 CTRLFCLR/CTRLFSET – Control register F Clear/Set

This register is mapped into two I/O memory locations, one for clearing (CTRLxCLR) and one for setting the register bits (CTRLxSET) when written. Both memory locations will give the same result when read.

The individual status bit can be set by writing a one to its bit location in CTRLxSET, and cleared by writing a one to its bit location in CTRLxCLR. This allows each bit to be set or cleared without use of a read-modify-write operation on a single register.

Bit	7	6	5	4	3	2	1	0
+0x08	–	–	–	QDECINDX	CMD[1:0]		LUPD	DIR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
+0x09	–	–	–	QDECINDX	CMD[1:0]		LUPD	DIR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:3 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 4 – QDECINDX: QDEC Index Flag
This bit indicates that a QDEC index is observed. The flag is cleared when counting up or down from zero. Normally this bit is controlled in hardware by the event actions, but this bit can also be changed from software.

Bit 3:2 – CMD[1:0]: Command
These bits can be used for software control of update, restart, and reset of the timer/counter. The command bits are always read as zero.

Table 14-8. Command selections.

CMD	Group configuration	Command action
00	NONE	None
01	UPDATE	Force update
10	RESTART	Force restart
11	RESET	Force hard reset (ignored if T/C is not in OFF state)

Bit 1 – LUPD: Lock Update
When this bit is set, no update of the buffered registers is performed, even though an UPDATE condition has occurred. Locking the update ensures that all buffers, including DTI buffers, are valid before an update is performed.

This bit has no effect when input capture operation is enabled.

Bit 0 – DIR: Counter Direction
When zero, this bit indicates that the counter is counting up (incrementing). A one indicates that the counter is in the down-counting (decrementing) state.

Normally this bit is controlled in hardware by the waveform generation mode or by event actions, but this bit can also be changed from software.

14.12.9 CTRLGCLR/CTRLGSET – Control register G Clear/Set

Bit	7	6	5	4	3	2	1	0
+0x0A/ +0x0B	–	–	–	CCDBV	CCCBV	CCBBV	CCABV	PERBV
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Refer to "CTRLFCLR/CTRLFSET – Control register F Clear/Set" on page 179 for information on how to access this type of status register.

Bit 7:5 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 4:1 – CCxBV: Compare or Capture x Buffer Valid
These bits are set when a new value is written to the corresponding CCxBUF register. These bits are automatically cleared on an UPDATE condition.

Note that when input capture operation is used, this bit is set on a capture event and cleared if the corresponding CCxIF is cleared.

Bit 0 – PERBV: Period Buffer Valid
This bit is set when a new value is written to the PERBUF register. This bit is automatically cleared on an UPDATE condition.

14.12.10 INTFLAGS – Interrupt Flag register

Bit	7	6	5	4	3	2	1	0
+0x0C	CCDIF	CCCIF	CCBIF	CCAIF	–	–	ERRIF	OVFIF
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:4 – CCxIF: Compare or Capture Channel x Interrupt Flag
The compare or capture interrupt flag (CCxIF) is set on a compare match or on an input capture event on the corresponding CC channel.

For all modes of operation except for capture, the CCxIF will be set when a compare match occurs between the count register (CNT) and the corresponding compare register (CCx). The CCxIF is automatically cleared when the corresponding interrupt vector is executed.

For input capture operation, the CCxIF will be set if the corresponding compare buffer contains valid data (i.e., when CCxBV is set). The flag will be cleared when the CCx register is read. Executing the interrupt vector in this mode of operation will not clear the flag.

The flag can also be cleared by writing a one to its bit location.

The CCxIF can be used for requesting a DMA transfer. A DMA read or write access of the corresponding CCx or CCxBUF will then clear the CCxIF and release the request.

Bit 3:2 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 1 – ERRIF: Error Interrupt Flag

This flag is set on multiple occasions, depending on the mode of operation.

In the FRQ or PWM waveform generation mode of operation, ERRIF is set on a fault detect condition from the fault protection feature in the AWeX extention. For timer/counters which do not have the AWeX extention available, this flag is never set in FRQ or PWM waveform generation mode.

For capture operation, ERRIF is set if a buffer overflow occurs on any of the CC channels.

For event controlled QDEC operation, ERRIF is set when an incorrect index signal is given.

This flag is automatically cleared when the corresponding interrupt vector is executed. The flag can also be cleared by writing a one to this location.

Bit 0 – OVFIF: Overflow/Underflow Interrupt Flag
This flag is set either on a TOP (overflow) or BOTTOM (underflow) condition, depending on the WGMODE setting. OVFIF is automatically cleared when the corresponding interrupt vector is executed. The flag can also be cleared by writing a one to its bit location.

OVFIF can also be used for requesting a DMA transfer. A DMA write access of CNT, PER, or PERBUF will then clear the OVFIF bit.

14.12.11 TEMP – Temporary bits for 16-bit Access

The TEMP register is used for single-cycle, 16-bit access to the 16-bit timer/counter registers by the CPU. The DMA controller has a separate temporary storage register. There is one common TEMP register for all the 16-bit Timer/counter registers.

For more details, refer to "The combined EIND + Z register." on page 12.

Bit	7	6	5	4	3	2	1	0
+0x0F	TEMP[7:0]							
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

14.12.12 CNTL – Counter register Low

The CNTH and CNTL register pair represents the 16-bit value, CNT. CNT contains the 16-bit counter value in the timer/counter. CPU and DMA write access has priority over count, clear, or reload of the counter.

For more details on reading and writing 16-bit registers, refer to "The combined EIND + Z register." on page 12.

Bit	7	6	5	4	3	2	1	0
+0x20	CNT[7:0]							
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 7:0 – CNT[7:0]: Counter low byte
These bits hold the LSB of the 16-bit counter register.

14.12.13 CNTH – Counter register High

Bit	7	6	5	4	3	2	1	0
+0x21	CNT[15:8]							
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

Bit 7:0 – CNT[15:8]: Counter high byte
These bits hold the MSB of the 16-bit counter register.

14.12.14 PERL – Period register Low

The PERH and PERL register pair represents the 16-bit value, PER. PER contains the 16-bit TOP value in the timer/counter.

Bit	7	6	5	4	3	2	1	0
+0x26	PER[7:0]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	1	1	1	1	1	1	1	1

Bit 7:0 – PER[7:0]: Periodic low byte
These bits hold the LSB of the 16-bit period register.

14.12.15 PERH – Period register H

Bit	7	6	5	4	3	2	1	0
+0x27	PER[15:8]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	1	1	1	1	1	1	1	1

Bit 7:0 – PER[15:8]: Periodic high byte
These bits hold the MSB of the 16-bit period register.

14.12.16 CCxL – Compare or Capture x register Low

The CCxH and CCxL register pair represents the 16-bit value, CCx. These 16-bit register pairs have two functions, depending of the mode of operation.

For capture operation, these registers constitute the second buffer level and access point for the CPU and DMA.

For compare operation, these registers are continuously compared to the counter value. Normally, the outputs form the comparators are then used for generating waveforms.

CCx registers are updated with the buffer value from their corresponding CCxBUF register when an UPDATE condition occurs.

Bit	7	6	5	4	3	2	1	0
	CCx[7:0]							
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 7:0 – CCx[7:0]: Compare or Capture x low byte
These bits hold the LSB of the 16-bit compare or capture register.

14.12.17 CCxH – Compare or Capture x register High

Bit	7	6	5	4	3	2	1	0
	CCx[15:8]							
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

Bit 7:0 – CCx[15:8]: Compare or Capture x high byte
These bits hold the MSB of the 16-bit compare or capture register.

14.12.18 PERBUFL – Timer/Counter Period Buffer Low

The PERBUFH and PERBUFL register pair represents the 16-bit value, PERBUF. This 16-bit register serves as the buffer for the period register (PER). Accessing this register using the CPU or DMA will affect the PERBUFV flag.

Bit	7	6	5	4	3	2	1	0
+0x36	PERBUF[7:0]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	1	1	1	1	1	1	1	1

Bit 7:0 – PERBUF[7:0]: Period Buffer low byte
These bits hold the LSB of the 16-bit period buffer register.

14.12.19 PERBUFH – Timer/Counter Period Buffer High

Bit	7	6	5	4	3	2	1	0
+0x37	PERBUF[15:8]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	1	1	1	1	1	1	1	1

Bit 7:0 – PERBUF[15:8]: Period Buffer high byte
These bits hold the MSB of the 16-bit period buffer register.

14.12.20 CCxBUFL – Compare or Capture x Buffer register Low

The CCxBUFH and CCxBUFL register pair represents the 16-bit value, CCxBUF. These 16-bit registers serve as the buffer for the associated compare or capture registers (CCx). Accessing any of these registers using the CPU or DMA will affect the corresponding CCxBV status bit.

Bit	7	6	5	4	3	2	1	0
	CCxBUFx[7:0]							
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 7:0 – CCxBUF[7:0]: Compare or Capture low byte
These bits hold the LSB of the 16-bit compare or capture buffer register.

14.12.21 CCxBUFH – Compare or Capture x Buffer register High

Bit	7	6	5	4	3	2	1	0
	CCxBUF[15:8]							
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

Bit 7:0 – CCxBUF[15:8]: Compare or Capture high byte
These bits hold the MSB of the 16-bit compare or capture buffer register.

14.13 Register summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	CTRLA	–	–	–	–	CLKSEL[3:0]				175
+0x01	CTRLB	CCDEN	CCCEN	CCBEN	CCAEN	–	WGMODE[2:0]			175
+0x02	CTRLC	–	–	–	–	CMPD	CMPC	CMPB	CMPA	176
+0x03	CTRLD	EVACT[2:0]			EVDLY	EVSEL[3:0]				176
+0x04	CTRLE	–	–	–	–	–	–	BYTEM		178
+0x05	Reserved	–	–	–	–	–	–	–	–	
+0x06	INTCTRLA	–	–	–	–	ERRINTLVL[1:0]		OVINTLVL[1:0]		178
+0x07	INTCTRLB	CCCINTLVL[1:0]		CCCINTLVL[1:0]		CCBINTLVL[1:0]		CCAINTLVL[1:0]		178
+0x08	CTRLFCLR	–	–	–	QDECINDX	CMD[1:0]		LUPD	DIR	179
+0x09	CTRLFSET	–	–	–	QDECINDX	CMD[1:0]		LUPD	DIR	180
+0x0A	CTRLGCLR	–	–	–	CCDBV	CCCBV	CCBBV	CCABV	PERBV	180
+0x0B	CTRLGSET	–	–	–	CCDBV	CCCBV	CCBBV	CCABV	PERBV	180
+0x0C	INTFLAGS	CCDIF	CCCIF	CCBIF	CCAIF	–	–	ERRIF	OVFIF	180
+0x0D	Reserved	–	–	–	–	–	–	–	–	
+0x0E	Reserved	–	–	–	–	–	–	–	–	
+0x0F	TEMP	TEMP[7:0]								181
+0x10 to +0x1F	Reserved	–	–	–	–	–	–	–	–	
+0x20	CNTL	CNT[7:0]								181
+0x21	CNTH	CNT[15:8]								181
+0x22 to +0x25	Reserved	–	–	–	–	–	–	–	–	
+0x26	PERL	PER[7:0]								182
+0x27	PERH	PER[8:15]								182
+0x28	CCAL	CCA[7:0]								182
+0x29	CCAH	CCA[15:8]								182
+0x2A	CCBL	CCB[7:0]								182
+0x2B	CCBH	CCB[15:8]								182
+0x2C	CCCL	CCC[7:0]								182
+0x02D	CCCH	CCC[15:8]								182
+0x2E	CCDL	CCD[7:0]								182
+0x2F	CCDH	CCD[15:8]								182
+0x30 to +0x35	Reserved	–	–	–	–	–	–	–	–	
+0x36	PERBUFL	PERBUF[7:0]								183

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x37	PERBUFH	PERBUF[15:8]								183
+0x38	CCABUFL	CCABUF[7:0]								183
+0x39	CCABUFH	CCABUF[15:8]								183
+0x3A	CCBBUFL	CCBBUF[7:0]								183
+0x3B	CCBBUFH	CCBBUF[15:8]								183
+0x3C	CCCBUFL	CCCBUF[7:0]								183
+0x3D	CCCBUFH	CCCBUF[15:8]								183
+0x3E	CCDBUFL	CCDBUF[7:0]								183
+0x3F	CCDBUFH	CCDBUF[15:8]								183

14.14 Interrupt vector summary

Table 14-9. Timer/counter interrupt vectors and their word offset address.

Offset	Source	Interrupt description
0x00	OVF_vect	Timer/counter overflow/underflow interrupt vector offset
0x02	ERR_vect	Timer/counter error interrupt vector offset
0x04	CCA_vect	Timer/counter compare or capture channel A interrupt vector offset
0x06	CCB_vect	Timer/counter compare or capture channel B interrupt vector offset
0x08	CCC_vect(1)	Timer/counter compare or capture channel C interrupt vector offset
0x0A	CCD_vect(1)	Timer/counter compare or capture channel D interrupt vector offset

Note: 1. Available only on timer/counters with four compare or capture channels.

18. RTC – Real-Time Counter

18.1 Features

- 16-bit resolution
- Selectable clock source
  - 32.768kHz external crystal
  - External clock
  - 32.768kHz internal oscillator
  - 32kHz internal ULP oscillator
- Programmable 10-bit clock prescaling
- One compare register
- One period register
- Clear counter on period overflow
- Optional interrupt/event on overflow and compare match

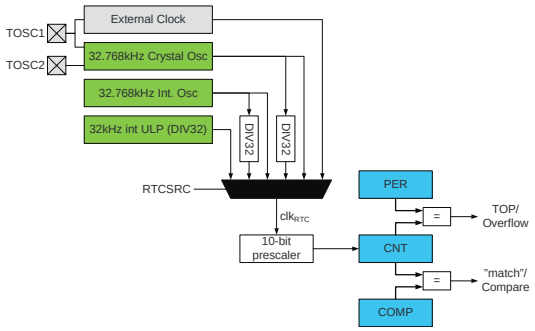
18.2 Overview

The 16-bit real-time counter (RTC) is a counter that typically runs continuously, including in low-power sleep modes, to keep track of time. It can wake up the device from sleep modes and/or interrupt the device at regular intervals.

The reference clock is typically the 1.024kHz output from a high-accuracy crystal of 32.768kHz, and this is the configuration most optimized for low power consumption. The faster 32.768kHz output can be selected if the RTC needs a resolution higher than 1ms. The RTC can also be clocked from an external clock signal, the 32.768kHz internal oscillator or the 32kHz internal ULP oscillator.

The RTC includes a 10-bit programmable prescaler that can scale down the reference clock before it reaches the counter. A wide range of resolutions and time-out periods can be configured. With a 32.768kHz clock source, the maximum resolution is 30.5µs, and time-out periods can range up to 2000 seconds. With a resolution of 1s, the maximum time-out period is more than18 hours (65536 seconds). The RTC can give a compare interrupt and/or event when the counter equals the compare register value, and an overflow interrupt and/or event when it equals the period register value.

Figure 18-1. Real-time counter overview.



18.2.1 Clock Domains

The RTC is asynchronous, operating from a different clock source independently of the main system clock and its derivative clocks, such as the peripheral clock. For control and count register updates, it will take a number of RTC clock and/or peripheral clock cycles before an updated register value is available in a register or until a configuration change has effect on the RTC. This synchronization time is described for each register. Refer to "RTCCTRL – RTC Control register" on page 92 for selecting the asynchronous clock source for the RTC.

18.2.2 Interrupts and Events

The RTC can generate both interrupts and events. The RTC will give a compare interrupt and/or event at the first count after the counter value equals the Compare register value. The RTC will give an overflow interrupt request and/or event at the first count after the counter value equals the Period register value. The overflow will also reset the counter value to zero.

Due to the asynchronous clock domain, events will be generated only for every third overflow or compare match if the period register is zero. If the period register is one, events will be generated only for every second overflow or compare match. When the period register is equal to or above two, events will trigger at every overflow or compare match, just as the interrupt request.

18.3 Register Descriptions

18.3.1 CTRL – Control register

Bit	7	6	5	4	3	2	1	0
+0x00	–	–	–	–	–	PRESCALER[2:0]		
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:3 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 2:0 – PRESCALER[2:0]: Clock Prescaling factor
These bits define the prescaling factor for the RTC clock according to Table 18-1 on page 214.

Table 18-1. Real-time counter clock prescaling factor.

PRESCALER[2:0]	Group configuration	RTC clock prescaling
000	OFF	No clock source, RTC stopped
001	DIV1	RTC clock / 1 (no prescaling)
010	DIV2	RTC clock / 2
011	DIV8	RTC clock / 8
100	DIV16	RTC clock / 16
101	DIV64	RTC clock / 64
110	DIV256	RTC clock / 256
111	DIV1024	RTC clock / 1024

18.3.2 STATUS – Status register

Bit	7	6	5	4	3	2	1	0
+0x01	–	–	–	–	–	–	–	SYNCBUSY
Read/Write	R	R	R	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

Bit 7:1 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 0 – SYNCBUSY: Synchronization Busy Flag
This flag is set when the CNT, CTRL, PER, or COMP register is busy synchronizing between the RTC clock and system clock domains after writing any of these registers or when waking up from a sleep mode where the peripheral clock is stopped. This flag is automatically cleared when the synchronisation is complete.

18.3.3 INTCTRL – Interrupt Control register

Bit	7	6	5	4	3	2	1	0
+0x02	–	–	–	–	COMPINTLVL[1:0]		OVFINTLVL[1:0]	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:4 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 3:2 – COMPINTLVL[1:0]: Compare Match Interrupt Enable
These bits enable the RTC compare match interrupt and select the interrupt level, as described in “Interrupts and Programmable Multilevel Interrupt Controller” on page 131. The enabled interrupt will trigger when COMPIF in the INTFLAGS register is set.

Bit 1:0 – OVFINTLVL[1:0]: Overflow Interrupt Enable
These bits enable the RTC overflow interrupt and select the interrupt level, as described in “Interrupts and Programmable Multilevel Interrupt Controller” on page 131. The enabled interrupt will trigger when OVFIF in the INTFLAGS register is set.

18.3.4 INTFLAGS – Interrupt Flag register

Bit	7	6	5	4	3	2	1	0
+0x03	–	–	–	–	–	–	COMPIF	OVFIF
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 unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 1 – COMPIF: Compare Match Interrupt Flag
This flag is set on the next count after a compare match condition occurs. It is cleared automatically when the RTC compare match interrupt vector is executed. The flag can also be cleared by writing a one to its bit location.

Bit 0 – OVFIF: Overflow Interrupt Flag
This flag is set on the next count after an overflow condition occurs. It is cleared automatically when the RTC overflow interrupt vector is executed. The flag can also be cleared by writing a one to its bit location.

18.3.5 TEMP – Temporary register

Bit	7	6	5	4	3	2	1	0
+0x04	TEMP[7:0]							
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 7:0 – TEMP[7:0]: Temporary bits
This register is used for 16-bit access to the counter value, compare value, and TOP value registers. The low byte of the 16-bit register is stored here when it is written by the CPU. The high byte of the 16-bit register is stored when the low byte is read by the CPU. For more details, refer to “The combined EIND + Z register.” on page 12.

18.3.6 CNTL – Counter register Low

The CNTH and CNTL register pair represents the 16-bit value, CNT. CNT counts positive clock edges on the prescaled RTC clock. Reading and writing 16-bit values requires special attention. Refer to "The combined EIND + Z register." on page 12 for details.

Due to synchronization between the RTC clock and system clock domains, there is a latency of two RTC clock cycles from updating the register until this has an effect. Application software needs to check that the SYNCBUSY flag in the "STATUS – Status register" on page 214 is cleared before writing to this register or reading the register after waking up from a sleep mode where the peripheral clock is stopped

Bit	7	6	5	4	3	2	1	0
+0x08	CNT[7:0]							
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 7:0 – CNT[7:0]: Counter Value low byte
These bits hold the LSB of the 16-bit real-time counter value.

18.3.7 CNTH – Counter register High

Bit	7	6	5	4	3	2	1	0
+0x09	CNT[15:8]							
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

Bit 7:0 – CNT[15:8]: Counter Value highbyte
These bits hold the MSB of the 16-bit real-time counter value.

18.3.8 PERL – Period register Low

The PERH and PERL register pair represents the 16-bit value, PER. PER is constantly compared with the counter value (CNT). A match will set OVFIF in the INTFLAGS register and clear CNT. Reading and writing 16-bit values requires special attention. Refer to "The combined EIND + Z register." on page 12 for details.

Due to synchronization between the RTC clock and system clock domains, there is a latency of two RTC clock cycles from updating the register until this has an effect. Application software needs to check that the SYNCBUSY flag in the "STATUS – Status register" on page 214 is cleared before writing to this register.

Bit	7	6	5	4	3	2	1	0
+0x0A	PER[7:0]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	1	1	1	1	1	1	1	1

Bit 7:0 – PER[7:0]: Period low byte
These bits hold the LSB of the 16-bit RTC TOP value.

18.3.9 PERH – Period register High

Bit	7	6	5	4	3	2	1	0
+0x0B	PER[15:8]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	1	1	1	1	1	1	1	1

Bits 7:0 – PER[15:8]: Period high byte
These bits hold the MSB of the 16-bit RTC TOP value.

18.3.10 COMPL – Compare register Low

The COMPH and COMPL register pair represent the 16-bit value, COMP. COMP is constantly compared with the counter value (CNT). A compare match will set COMPIF in the INTFLAGS register. Reading and writing 16-bit values requires special attention. Refer "The combined EIND + Z register." on page 12 for details.

Due to synchronization between the RTC clock and system clock domains, there is a latency of two RTC clock cycles from updating the register until this has an effect. Application software needs to check that the SYNCBUSY flag in the "STATUS – Status register" on page 214 is cleared before writing to this register.

If the COMP value is higher than the PER value, no RTC compare match interrupt requests or events will ever be generated.

Bit	7	6	5	4	3	2	1	0
+0x0C	COMP[7:0]							
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 7:0 – COMP[7:0]: Compare value low byte
These bits hold the LSB of the 16-bit RTC compare value.

18.3.11 COMPH – Compare register High

Bit	7	6	5	4	3	2	1	0
+0x0D	COMP[15:8]							
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

Bit 7:0 – COMP[15:8]: Compare value high byte
These bits hold the MSB of the 16-bit RTC compare value.

18.4 Register summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	CTRL	–	–	–	–	–	PRESCALER[2:0]			214
+0x01	STATUS	–	–	–	–	–	–	–	SYNCBUSY	214
+0x02	INTCTRL	–	–	–	–	COMPINTLVL[1:0]		OVFINTLVL[1:0]		215
+0x03	INTFLAGS	–	–	–	–	–	–	COMPIF	OVFIF	215
+0x04	TEMP	–	–	–	–	–	–	COMPIF	OVFIF	215
+0x05	Reserved	–	–	–	–	–	–	–	–	
+0x06	Reserved	–	–	–	–	–	–	–	–	
+0x07	Reserved	–	–	–	–	–	–	–	–	
+0x08	CNTL	TEMP[7:0]								216
+0x09	CNTH	CNT[7:0]								216
+0x0A	PERL	CNT[15:8]								216
+0x0B	PERH	PER[7:0]								216
+0x0C	COMPL	PER[15:8]								217
+0x0D	COMPH	COMP[7:0]								217

18.5 Interrupt Vector Summary

Table 18-2. RTC interrupt vectors and their word offset.

Offset	Source	Interrupt description
0x00	OVF_vect	Real-time counter overflow interrupt vector
0x02	COMP_vect	Real-time counter compare match interrupt vector

22. SPI – Serial Peripheral Interface

22.1 Features

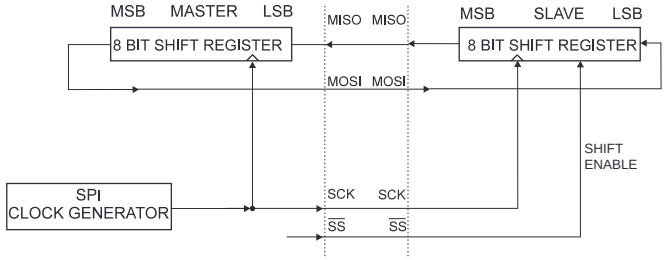
- Full-duplex, three-wire synchronous data transfer
- Master or slave operation
- Lsb first or msb first data transfer
- Eight programmable bit rates
- Interrupt flag at the end of transmission
- Write collision flag to indicate data collision
- Wake up from idle sleep mode
- Double speed master mode

22.2 Overview

The Serial Peripheral Interface (SPI) is a high-speed synchronous data transfer interface using three or four pins. It allows fast communication between an XMEGA device and peripheral devices or between several microcontrollers. The SPI supports full-duplex communication.

A device connected to the bus must act as a master or slave.The master initiates and controls all data transactions. The interconnection between master and slave devices with SPI is shown in Figure 22-1 on page 273. The system consists of two shift registers and a master clock generator. The SPI master initiates the communication cycle by pulling the slave select ($\overline{SS}$) signal low for 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 are always shifted from master to slave on the master output, slave input (MOSI) line, and from slave to master on the master input, slave output (MISO) line. After each data packet, the master can synchronize the slave by pulling the $\overline{SS}$ line high.

Figure 22-1. SPI master-slave interconnection.



The SPI module is unbuffered in the transmit direction and single 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, a received character must be read from the DATA register before the next character has been completely shifted in. Otherwise, the first byte will be lost.

In SPI slave mode, the control logic will sample the incoming signal on the SCK pin. To ensure correct sampling of this clock signal, the minimum low and high periods must each be longer than two CPU clock cycles.

When the SPI module is enabled, the data direction of the MOSI, MISO, SCK, and $\overline{SS}$ pins is overridden according to Table 22-1 on page 274. The pins with user-defined direction must be configured from software to have the correct direction according to the application.

Table 22-1. SPI pin override and directions.

Pin	Master mode	Slave mode
MOSI	User defined	Input
MISO	Input	User defined
SCK	User defined	Input
$\overline{SS}$	User defined	Input

22.3 Master Mode

In master mode, the SPI interface has no automatic control of the $\overline{SS}$ line. If the $\overline{SS}$ pin is used, it must be configured as output and controlled by user software. If the bus consists of several SPI slaves and/or masters, a SPI master can use general purpose I/O pins to control the $\overline{SS}$ line to each of the slaves on the bus.

Writing a byte to the DATA register starts the SPI clock generator and the hardware shifts the eight bits into the selected slave. After shifting one byte, the SPI clock generator stops and the SPI interrupt flag is set. The master may continue to shift the next byte by writing new data to the DATA register, or can signal the end of the transfer by pulling the $\overline{SS}$ line high. The last incoming byte will be kept in the buffer register.

If the $\overline{SS}$ pin is not used and is configured as input, it must be held high to ensure master operation. If the $\overline{SS}$ pin is set as input and is being driven low, the SPI module will interpret this as another master trying to take control of the bus. To avoid bus contention, the master will take the following action:

1. The master enters slave mode.
2. The SPI interrupt flag is set.

22.4 Slave Mode

In slave mode, the SPI module will remain sleeping with the MISO line tri-stated as long as the $\overline{SS}$ pin is driven high. In this state, software may update the contents of the DATA register, but the data will not be shifted out by incoming clock pulses on the SCK pin until the $\overline{SS}$ pin is driven low. If $\overline{SS}$ is driven low, the slave will start to shift out data on the first SCK clock pulse. When one byte has been completely shifted, the SPI interrupt flag is set. The slave may continue placing new data to be sent into the DATA register before reading the incoming data. The last incoming byte will be kept in the buffer register.

When $\overline{SS}$ is driven high, the SPI logic is reset, and the SPI slave will not receive any new data. Any partially received packet in the shift register will be dropped.

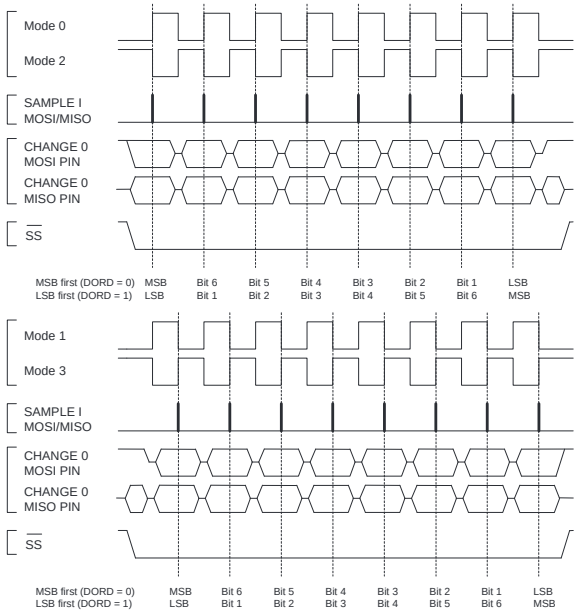
As the $\overline{SS}$ pin is used to signal the start and end of a transfer, it is also useful for doing packet/byte synchronization, keeping the slave bit counter synchronous with the master clock generator.

22.5 Data Modes

There are four combinations of SCK phase and polarity with respect to serial data. The SPI data transfer formats are shown in Figure 22-2. Data bits are shifted out and latched in on opposite edges of the SCK signal, ensuring sufficient time for data signals to stabilize.

The leading edge is the first clock edge of a clock cycle. The trailing edge is the last clock edge of a clock cycle.

Figure 22-2. SPI transfer modes.



22.6 DMA Support

DMA support on the SPI module is available only in slave mode. The SPI slave can trigger a DMA transfer as one byte has been shifted into the DATA register. It is possible, however, to use the XMEGA USART in SPI mode and then have DMA support in master mode. For details, refer to "USART in Master SPI Mode" on page 291.

22.7 Register Description

22.7.1 CTRL – Control register

Bit	7	6	5	4	3	2	1	0
+0x00	CLK2X	ENABLE	DORD	MASTER	MODE[1:0]		PRESCALER[1:0]	
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 – CLK2X: Clock Double
When this bit is set, the SPI speed (SCK frequency) will be doubled in master mode (see Table 22-3 on page 291).

Bit 6 – ENABLE: Enable
Setting this bit enables the SPI module. This bit must be set to enable any SPI operations.

Bit 5 – DORD: Data Order
DORD decides the data order when a byte is shifted out from the DATA register. When DORD is written to one, the least-significant bit (lsb) of the data byte is transmitted first, and when DORD is written to zero, the most-significant bit (msb) of the data byte is transmitted first.

Bit 4 – MASTER: Master Select
This bit selects master mode when written to one, and slave mode when written to zero. If $\overline{SS}$ is configured as an input and driven low while master mode is set, master mode will be cleared.

Bit 3:2 – MODE[1:0]: Transfer Mode
These bits select the transfer mode. The four combinations of SCK phase and polarity with respect to the serial data are shown in Table 22-2 on page 276. These bits decide whether the first edge of a clock cycle (leading edge) is rising or falling, and whether data setup and sample occur on the leading or trailing edge.

When the leading edge is rising, the SCK signal is low when idle, and when the leading edge is falling, the SCK signal is high when idle.

Table 22-2. SPI transfer modes.

MODE[1:0]	Group configuration	Leading edge	Trailing edge
00	0	Rising, sample	Falling, setup
01	1	Rising, setup	Falling, sample
10	2	Falling, sample	Rising, setup
11	3	Falling, setup	Rising, sample

Bits 1:0 – PRESCALER[1:0]: Clock Prescaler
These two bits control the SPI clock rate configured in master mode. These bits have no effect in slave mode. The relationship between SCK and the peripheral clock frequency (clk_{PER}) is shown in Table 22-3 on page 277.

Table 22-3. Relationship between SCK and the peripheral clock (Clk_{PER}) frequency.

CLK2X	PRESCALER[1:0]	SCK frequency
0	00	$Clk_{PER}/4$
0	01	$Clk_{PER}/16$
0	10	$Clk_{PER}/64$
0	11	$Clk_{PER}/128$
1	00	$Clk_{PER}/2$
1	01	$Clk_{PER}/8$
1	10	$Clk_{PER}/32$
1	11	$Clk_{PER}/64$

22.7.2 INTCTRL – Interrupt Control register

Bit	7	6	5	4	3	2	1	0
+0x01	–	–	–	–	–	–	INTLVL[1:0]	
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 unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 1:0 – INTLVL[1:0]: Interrupt Level
These bits enable the SPI interrupt and select the interrupt level, as described in "Interrupts and Programmable Multilevel Interrupt Controller" on page 131. The enabled interrupt will be triggered when IF in the STATUS register is set.

22.7.3 STATUS – Status register

Bit	7	6	5	4	3	2	1	0
+0x02	IF	WRCOL	–	–	–	–	–	–
Read/Write	R	R	R	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – IF: Interrupt Flag
This flag is set when a serial transfer is complete and one byte is completely shifted in/out of the DATA register. If $\overline{SS}$ is configured as input and is driven low when the SPI is in master mode, this will also set this flag. IF is cleared by hardware when executing the corresponding interrupt vector. Alternatively, the IF flag can be cleared by first reading the STATUS register when IF is set, and then accessing the DATA register.

Bit 6 – WRCOL: Write Collision Flag
The WRCOL flag is set if the DATA register is written during a data transfer. This flag is cleared by first reading the STATUS register when WRCOL is set, and then accessing the DATA register.

Bit 5:0 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

22.7.4 DATA – Data register

Bit	7	6	5	4	3	2	1	0
+0x03	DATA[7:0]							
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 DATA register is used for sending and receiving data. Writing to the register initiates the data transmission, and the byte written to the register will be shifted out on the SPI output line. Reading the register causes the shift register receive buffer to be read, returning the last byte successfully received.

22.8 Register summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	CTRL	CLK2X	ENABLE	DORD	MASTER	MODE[1:0]		PRESCALER[1:0]		276
+0x01	INTCTRL	–	–	–	–	–	–	INTLVL[1:0]		277
+0x02	STATUS	IF	WRCOL	–	–	–	–	–	–	277
+0x03	DATA	DATA[7:0]								278

22.9 Interrupt vector summary

Table 22-4. SPI interrupt vector and its offset word address.

Offset	Source	Interrupt Description
0x00	SPI_vect	SPI interrupt vector

23. USART

23.1 Features

- Full-duplex operation
- Asynchronous or synchronous operation
  - Synchronous clock rates up to 1/2 of the device clock frequency
  - Asynchronous clock rates up to 1/8 of the device clock frequency
- Supports serial frames with 5, 6, 7, 8, or 9 data bits and 1 or 2 stop bits
- Fractional baud rate generator
  - Can generate desired baud rate from any system clock frequency
  - No need for external oscillator with certain frequencies
- Built-in error detection and correction schemes
  - Odd or even parity generation and parity check
  - Data overrun and framing error detection
  - Noise filtering includes false start bit detection and digital low-pass filter
- Separate interrupts for
  - Transmit complete
  - Transmit data register empty
  - Receive complete
- Multiprocessor communication mode
  - Addressing scheme to address a specific devices on a multi device bus
  - Enable unaddressed devices to automatically ignore all frames
- Master SPI mode
  - Double buffered operation
  - Configurable data order
  - Operation up to 1/2 of the peripheral clock frequency
- IRCOM module for IrDA compliant pulse modulation/demodulation

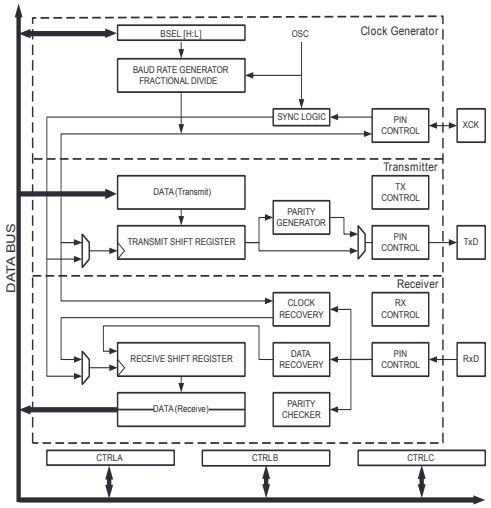
23.2 Overview

The universal synchronous and asynchronous serial receiver and transmitter (USART) is a fast and flexible serial communication module. The USART supports full-duplex communication and asynchronous and synchronous operation. The USART can be configured to operate in SPI master mode and used for SPI communication.

Communication is frame based, and the frame format can be customized to support a wide range of standards. The USART is buffered in both directions, enabling continued data transmission without any delay between frames. Separate interrupts for receive and transmit complete enable fully interrupt driven communication. Frame error and buffer overflow are detected in hardware and indicated with separate status flags. Even or odd parity generation and parity check can also be enabled.

A block diagram of the USART is shown in Figure 23-1 on page 281. The main functional blocks are the clock generator, the transmitter, and the receiver, which are indicated in dashed boxes.

Figure 23-1. USART block diagram.



The clock generator includes a fractional baud rate generator that is able to generate a wide range of USART baud rates from any system clock frequencies. This removes the need to use an external crystal oscillator with a specific frequency to achieve a required baud rate. It also supports external clock input in synchronous slave operation.

The transmitter consists of a single write buffer (DATA), a shift register, and a parity generator. The write buffer allows continuous data transmission without any delay between frames.

The receiver consists of a two-level receive buffer (DATA) and a shift register. Data and clock recovery units ensure robust synchronization and noise filtering during asynchronous data reception. It includes frame error, buffer overflow, and parity error detection.

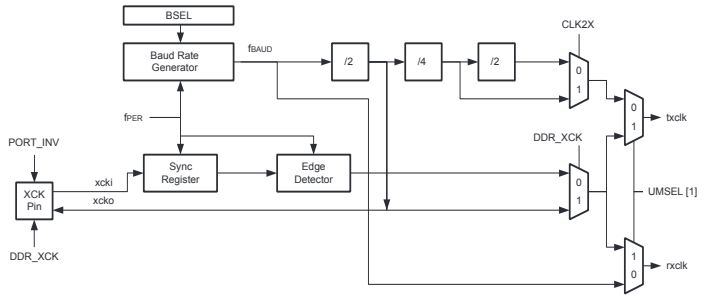
When the USART is set in master SPI mode, all USART-specific logic is disabled, leaving the transmit and receive buffers, shift registers, and baud rate generator enabled. Pin control and interrupt generation are identical in both modes. The registers are used in both modes, but their functionality differs for some control settings.

An IRCOM module can be enabled for one USART to support IrDA 1.4 physical compliant pulse modulation and demodulation for baud rates up to 115.2kbps. For details, refer to “IRCOM – IR Communication Module” on page 301.

23.3 Clock Generation

The clock used for baud rate generation and for shifting and sampling data bits is generated internally by the fractional baud rate generator or externally from the transfer clock (XCK) pin. Five modes of clock generation are supported: normal and double-speed asynchronous mode, master and slave synchronous mode, and master SPI mode.

Figure 23-2. Clock generation logic, block diagram.



23.3.1 Internal Clock Generation - The Fractional Baud Rate Generator

The fractional baud rate generator is used for internal clock generation for asynchronous modes, synchronous master mode, and master SPI mode operation. The output frequency generated (f_{BAUD}) is determined by the period setting (BSEL), an optional scale setting (BSCALE), and the peripheral clock frequency (f_{PER}). Table 23-1 on page 282 contains equations for calculating the baud rate (in bits per second) and for calculating the BSEL value for each mode of operation. It also shows the maximum baud rate versus peripheral clock frequency. BSEL can be set to any value between 0 and 4095. BSCALE can be set to any value between -7 and +7, and increases or decreases the baud rate slightly to provide the fractional baud rate scaling of the baud rate generator.

When BSEL is 0, BSCALE must also be 0. Also, the value $2^{ABS(BSCALE)}$ must at most be one half of the minimum number of clock cycles a frame requires. For more details, see "Fractional Baud Rate Generation" on page 289.

Table 23-1. Equations for calculating baud rate register settings.

Operating mode	Conditions	Baud rate[1] calculation	BSEL value calculation
Asynchronous normal speed mode (CLK2X = 0)	BSCALE ≥ 0 $f_{BAUD} \le \frac{f_{PER}}{16}$	$f_{BAUD} = \frac{f_{PER}}{2^{BSCALE}16(BSEL+1)}$	$BSEL = \frac{f_{PER}}{2^{BSCALE}16f_{BAUD}} - 1$
	BSCALE < 0 $f_{BAUD} \le \frac{f_{PER}}{16}$	$f_{BAUD} = \frac{f_{PER}}{16((2^{BSCALE}BSEL)+1)}$	$BSEL = \frac{1}{2^{BSCALE}}\left(\frac{f_{PER}}{16f_{BAUD}} - 1\right)$
Asynchronous double speed mode (CLK2X = 1)	BSCALE ≥ 0 $f_{BAUD} \le \frac{f_{PER}}{8}$	$f_{BAUD} = \frac{f_{PER}}{2^{BSCALE}8(BSEL+1)}$	$BSEL = \frac{f_{PER}}{2^{BSCALE}8f_{BAUD}} - 1$
	BSCALE < 0 $f_{BAUD} \le \frac{f_{PER}}{8}$	$f_{BAUD} = \frac{f_{PER}}{8((2^{BSCALE}BSEL)+1)}$	$BSEL = \frac{1}{2^{BSCALE}}\left(\frac{f_{PER}}{8f_{BAUD}} - 1\right)$
Synchronous and master SPI mode	$f_{BAUD} \le \frac{f_{PER}}{2}$	$f_{BAUD} = \frac{f_{PER}}{2(BSEL+1)}$	$BSEL = \frac{f_{PER}}{2f_{BAUD}} - 1$

Note: 1. The baud rate is defined to be the transfer rate in bits per second (bps)

For BSEL=0, all baud rates must be achieved by changing BSEL instead of setting BSCALE:

BSEL = ($2^{\text{desired BSCALE}^{-1}}$)

BSCALE	BSEL	BSCALE	BSEL
1	0	0	1
2	0	0	3
3	0	0	7
4	0	0	15
5	0	0	31
6	0	0	63
7	0	0	127

23.3.2 External Clock

External clock (XCK) is used in synchronous slave mode operation. The XCK clock input is sampled on the peripheral clock frequency (f_{PER}), and the maximum XCK clock frequency (f_{XCK})is limited by the following:

$f_{XCK} \le \frac{f_{PER}}{4}$

For each high and low period, XCK clock cycles must be sampled twice by the peripheral clock. If the XCK clock has jitter, or if the high/low period duty cycle is not 50/50, the maximum XCK clock speed must be reduced or the peripheral clock must be increased accordingly.

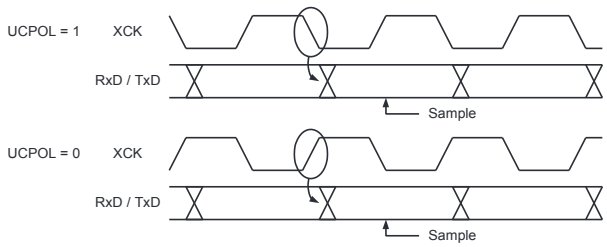
23.3.3 Double Speed Operation

Double speed operation allows for higher baud rates under asynchronous operation with lower peripheral clock frequencies. When this is enabled, the baud rate for a given asynchronous baud rate setting shown in Table 23-1 on page 282 will be doubled. In this mode, the receiver will use half the number of samples (reduced from 16 to 8) for data sampling and clock recovery. Due to the reduced sampling, a more accurate baud rate setting and peripheral clock are required. See "Asynchronous Data Reception" on page 287 for more details.

23.3.4 Synchronous Clock Operation

When synchronous mode is used, the XCK pin controls whether the transmission clock is input (slave mode) or output (master mode). The corresponding port pin must be set to output for master mode or to input for slave mode. The normal port operation of the XCK pin will be overridden. The dependency between the clock edges and data sampling or data change is the same. Data input (on RxD) is sampled at the XCK clock edge which is opposite the edge where data output (TxD) is changed.

Figure 23-3. Synchronous mode XCK timing.



Using the inverted I/O (INVEN) setting for the corresponding XCK port pin, the XCK clock edges used for data sampling and data change can be selected. If inverted I/O is disabled (INVEN=0), data will be changed at the rising XCK clock edge and sampled at the falling XCK clock edge. If inverted I/O is enabled (INVEN=1), data will be changed at the falling XCK clock edge and sampled at the rising XCK clock edge. For more details, see "I/O Ports" on page 139.

23.3.5 Master SPI Mode Clock Generation

For master SPI mode operation, only internal clock generation is supported. This is identical to the USART synchronous master mode, and the baud rate or BSEL setting is calculated using the same equations (see Table 23-1 on page 282).

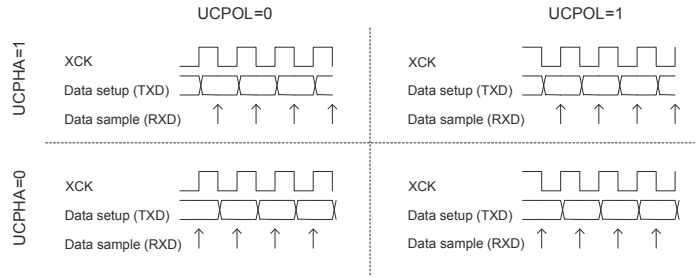
There are four combinations of the SPI clock (SCK) phase and polarity with respect to the serial data, and these are determined by the clock phase (UCPHA) control bit and the inverted I/O pin (INVEN) settings. The data transfer timing diagrams are shown in Figure 23-4 on page 284. Data bits are shifted out and latched in on opposite edges of the XCK signal, ensuring sufficient time for data signals to stabilize. The UCPHA and INVEN settings are summarized in Table 23-2 on page 284. Changing the setting of any of these bits during transmission will corrupt both the receiver and transmitter.

Table 23-2. INVEN and UCPHA functionality.

SPI Mode	INVEN	UCPHA	Leading edge	Trailing edge
0	0	0	Rising, sample	Falling, setup
1	0	1	Rising, setup	Falling, sample
2	1	0	Falling, sample	Rising, setup
3	1	1	Falling, setup	Rising, sample

The leading edge is the first clock edge of a clock cycle. The trailing edge is the last clock edge of a clock cycle.

Figure 23-4. UCPHA and INVEN data transfer timing diagrams.



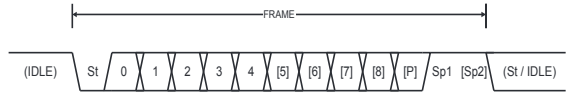
23.4 Frame Formats

Data transfer is frame based, where a serial frame consists of one character of data bits with synchronization bits (start and stop bits) and an optional parity bit for error checking. Note that this does not apply to master SPI operation (See "SPI Frame Formats" on page 285). The USART accepts all combinations of the following as valid frame formats:

- 1 start bit
- 5, 6, 7, 8, or 9 data bits
- no, even, or odd parity bit
- 1 or 2 stop bits

A frame starts with the start bit, followed by all the data bits (least-significant bit first and most-significant bit last). If enabled, the parity bit is inserted after the data bits, before the first stop bit. One frame can be directly followed by a start bit and a new frame, or the communication line can return to the idle (high) state. Figure 23-5 on page 285 illustrates the possible combinations of frame formats. Bits inside brackets are optional.

Figure 23-5. Frame formats.



St	:	Start bit, always low.
(n)	:	Data bits (0 to 8).
P	:	Parity bit, may be odd or even.
Sp	:	Stop bit, always high.
IDLE	:	No transfers on the communication line (RxD or TxD). The IDLE state is always high.

23.4.1 Parity Bit Calculation

Even or odd parity can be selected for error checking. If even parity is selected, the parity bit is set to one if the number of logical one data bits is odd (making the total number of ones even). If odd parity is selected, the parity bit is set to one if the number of logical one data bits is even (making the total number of ones odd).

23.4.2 SPI Frame Formats

The serial frame in SPI mode is defined to be one character of eight data bits. The USART in master SPI mode has two selectable frame formats:

- 8-bit data, msb first
- 8-bit data, lsb first

After a complete, 8-bit frame is transmitted, a new frame can directly follow it, or the communication line can return to the idle (high) state.

23.5 USART Initialization

USART initialization should use the following sequence:

1. Set the TxD pin value high, and optionally set the XCK pin low.
2. Set the TxD and optionally the XCK pin as output.
3. Set the baud rate and frame format.
4. Set the mode of operation (enables XCK pin output in synchronous mode).
5. Enable the transmitter or the receiver, depending on the usage.

For interrupt-driven USART operation, global interrupts should be disabled during the initialization.

Before doing a re-initialization with a changed baud rate or frame format, be sure that there are no ongoing transmissions while the registers are changed.

23.6 Data Transmission - The USART Transmitter

When the transmitter has been enabled, the normal port operation of the TxD pin is overridden by the USART and given the function as the transmitter's serial output. The direction of the pin must be set as output using the direction register for the corresponding port. For details on port pin control and output configuration, refer to "I/O Ports" on page 139.

23.6.1 Sending Frames

A data transmission is initiated by loading the transmit buffer (DATA) with the data to be sent. The data in the transmit buffer are moved to the shift register when the shift register is empty and ready to send a new frame. The shift register is loaded if it is in idle state (no ongoing transmission) or immediately after the last stop bit of the previous frame is transmitted. When the shift register is loaded with data, it will transfer one complete frame.

The transmit complete interrupt flag (TXCIF) is set and the optional interrupt is generated when the entire frame in the shift register has been shifted out and there are no new data present in the transmit buffer.

The transmit data register (DATA) can only be written when the data register empty flag (DREIF) is set, indicating that the register is empty and ready for new data.

When using frames with fewer than eight bits, the most-significant bits written to DATA are ignored. If 9-bit characters are used, the ninth bit must be written to the TXB8 bit before the low byte of the character is written to DATA.

23.6.2 Disabling the Transmitter

A disabling of the transmitter will not become effective until ongoing and pending transmissions are completed; i.e., when the transmit shift register and transmit buffer register do not contain data to be transmitted. When the transmitter is disabled, it will no longer override the TxDn pin, and the pin direction is set as input automatically by hardware, even if it was configured as output by the user.

23.7 Data Reception - The USART Receiver

When the receiver is enabled, the RxD pin functions as the receiver's serial input. The direction of the pin must be set as input, which is the default pin setting.

23.7.1 Receiving Frames

The receiver starts data reception when it detects a valid start bit. Each bit that follows the start bit will be sampled at the baud rate or XCK clock and shifted into the receive shift register until the first stop bit of a frame is received. A second stop bit will be ignored by the receiver. When the first stop bit is received and a complete serial frame is present in the receive shift register, the contents of the shift register will be moved into the receive buffer. The receive complete interrupt flag (RXCIF) is set, and the optional interrupt is generated.

The receiver buffer can be read by reading the data register (DATA) location. DATA should not be read unless the receive complete interrupt flag is set. When using frames with fewer than eight bits, the unused most-significant bits are read as zero. If 9-bit characters are used, the ninth bit must be read from the RXB8 bit before the low byte of the character is read from DATA.

23.7.2 Receiver Error Flags

The USART receiver has three error flags. The frame error (FERR), buffer overflow (BUFOVF) and parity error (PERR) flags are accessible from the status register. The error flags are located in the receive FIFO buffer together with their corresponding frame. Due to the buffering of the error flags, the status register must be read before the receive buffer (DATA), since reading the DATA location changes the FIFO buffer.

23.7.3 Parity Checker

When enabled, the parity checker calculates the parity of the data bits in incoming frames and compares the result with the parity bit of the corresponding frame. If a parity error is detected, the parity error flag is set.

23.7.4 Disabling the Receiver

A disabling of the receiver will be immediate. The receiver buffer will be flushed, and data from ongoing receptions will be lost.

23.7.5 Flushing the Receive Buffer

If the receive buffer has to be flushed during normal operation, read the DATA location until the receive complete interrupt flag is cleared.

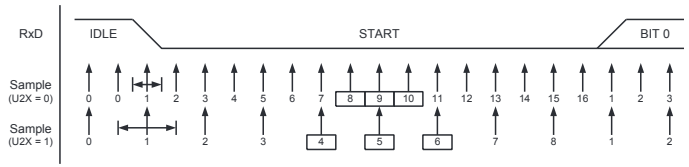
23.8 Asynchronous Data Reception

The USART includes a clock recovery and a data recovery unit for handling asynchronous data reception. The clock recovery unit is used for synchronizing the incoming asynchronous serial frames at the RxD pin to the internally generated baud rate clock. It samples and low-pass filters each incoming bit, thereby improving the noise immunity of the receiver. The asynchronous reception operational range depends on the accuracy of the internal baud rate clock, the rate of the incoming frames, and the frame size in number of bits.

23.8.1 Asynchronous Clock Recovery

The clock recovery unit synchronizes the internal clock to the incoming serial frames. Figure 23-6 on page 287 illustrates the sampling process for the start bit of an incoming frame. The sample rate is 16 times the baud rate for normal mode, and eight times the baud rate for double speed mode. The horizontal arrows illustrate the synchronization variation due to the sampling process. Note the larger time variation when using the double speed mode of operation. Samples denoted as zero are samples done when the RxD line is idle; i.e., when there is no communication activity.

Figure 23-6. Start bit sampling.

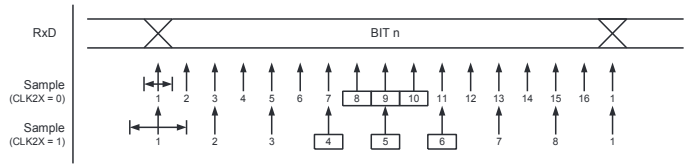


When the clock recovery logic detects a high (idle) to low (start) transition on the RxD line, the start bit detection sequence is initiated. Sample 1 denotes the first zero-sample, as shown in the figure. The clock recovery logic then uses samples 8, 9, and 10 for normal mode and samples 4, 5, and 6 for double speed mode to decide if a valid start bit is received. If two or three samples have a low level, the start bit is accepted. The clock recovery unit is synchronized, and the data recovery can begin. If two or three samples have a high level, the start bit is rejected as a noise spike, and the receiver looks for the next high-to-low transition. The process is repeated for each start bit.

23.8.2 Asynchronous Data Recovery

The data recovery unit uses sixteen samples in normal mode and eight samples in double speed mode for each bit. Figure 23-7 on page 287 shows the sampling process of data and parity bits.

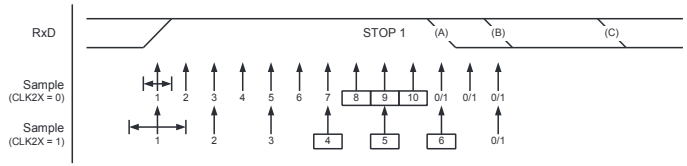
Figure 23-7. Sampling of data and parity bits.



As for start bit detection, an identical majority voting technique is used on the three center samples for deciding of the logic level of the received bit. The process is repeated for each bit until a complete frame is received. It includes the first stop bit, but excludes additional ones. If the sampled stop bit is a 0 value, the frame error (FERR) flag will be set.

Figure 23-8 on page 288 shows the sampling of the stop bit in relation to the earliest possible beginning of the next frame's start bit.

Figure 23-8. Stop bit and next start bit sampling.



A new high-to-low transition indicating the start bit of a new frame can come right after the last of the bits used for majority voting. For normal speed mode, the first low level sample can be at the point marked (A) in Stop Bit Sampling and Next Start Bit Sampling. For double speed mode, the first low level must be delayed to point (B). Point (C) marks a stop bit of full length at nominal baud rate. The early start bit detection influences the operational range of the receiver.

23.8.3 Asynchronous Operational Range

The operational range of the receiver is dependent on the mismatch between the received bit rate and the internally generated baud rate. If an external transmitter is sending using bit rates that are too fast or too slow, or if the internally generated baud rate of the receiver does not match the external source's base frequency, the receiver will not be able to synchronize the frames to the start bit.

The following equations can be used to calculate the ratio of the incoming data rate and internal receiver baud rate.

$$R_{slow} = \frac{(D+1)S}{S-1+D\,S+S_F} \qquad R_{fast} = \frac{(D+2)S}{(D+1)S+S_M}$$

D	:	Sum of character size and parity size (D = 5 to 10 bits).
S	:	Samples per bit. S = 16 for normal speed mode and S = 8 for double speed mode.
S_F	:	First sample number used for majority voting. S_F = 8 for normal speed mode and S_F = 4 for double speed mode.
S_M	:	Middle sample number used for majority voting. S_M = 9 for normal speed mode and S_M = 5 for double speed mode.
R_{slow}	:	The ratio of the slowest incoming data rate that can be accepted in relation to the receiver baud rate.
R_{fast}	:	The ratio of the fastest incoming data rate that can be accepted in relation to the receiver baud rate.

Table 23-3 and Table 23-4 on page 289 list the maximum receiver baud rate error that can be tolerated. Normal speed mode has higher tolerance of baud rate variations

Table 23-3. Recommended maximum receiver baud rate error for normal speed mode.

D #(Data + Parity Bit)	R_{slow} [%]	R_{fast} [%]	Max total error [%]	Recommended max receiver error [%]
5	93.20	106.67	+6.67/-6.80	± 3.0
6	94.12	105.79	+5.79/-5.88	± 2.5
7	94.81	105.11	+5.11/-5.19	± 2.0
8	95.36	104.58	+4.58/-4.54	± 2.0
9	95.81	104.14	+4.14/-4.19	± 1.5
10	96.17	103.78	+3.78/-3.83	± 1.5

Table 23-4. Recommended maximum receiver baud rate error for double speed mode.

D #(Data + Parity Bit)	R_{slow} [%]	R_{fast} [%]	Max Total Error [%]	Recommended Max Receiver Error [%]
5	94.12	105.66	+5.66/-5.88	± 2.5
6	94.92	104.92	+4.92/-5.08	± 2.0
7	95.52	104.35	+4.35/-4.48	± 1.5
8	96.00	103.90	+3.90/-4.00	± 1.5
9	96.39	103.53	+3.53/-3.61	± 1.5
10	96.70	103.23	+3.23/-3.30	± 1.0

The recommendations for the maximum receiver baud rate error assume that the receiver and transmitter equally divide the maximum total error.

23.9 Fractional Baud Rate Generation

Fractional baud rate generation is possible for asynchronous operation due to the relatively high number of clock cycles for each frame. Each bit is sampled sixteen times, but only the three middle samples are of importance. The total number of samples for one frame is also relatively high. Given a 1-start, 8-data, no-parity, and 1-stop-bit frame format, and assuming that normal speed mode is used, the total number of samples for a frame is (1+8+1)×16 or 160. As stated earlier, the UART can tolerate some variation in clock cycles for each sample. The critical factor is the time from the falling edge of the start bit (i.e., the clock synchronization) until the last bit's (i.e., the first stop bit's) value is recovered.

Standard baud rate generators have the unwanted property of having large frequency steps between high baud rate settings. The worst case is found between the BSEL values 0x000 and 0x001. Going from a BSEL value of 0x000, which has a 10-bit frame of 160 clock cycles, to a BSEL value of 0x001, with 320 clock cycles, gives a 50% change in frequency. Ideally, the step size should be small even between the fastest baud rates. This is where the advantage of the fractional baud rate generator emerges.

In principle, the fractional baud rate generator works by doing uneven counting and then distributing the error evenly over the entire frame. A typical count sequence for an ordinary baud rate generator is:

2, 1, 0, 2, 1, 0, 2, 1, 0, 2, …

which has an even period time. A baud rate clock ticks each time the counter reaches zero, and a sample of the signal received on RxD is taken for every 16th baud rate clock tick.

For the fractional baud rate generator, the count sequence can have an uneven period:

2, 1, 0, 2, 1-1, 0, 2, 1, 0, 2, 1-1, 0, ...

In this example, an extra cycle is added to every second baud clock. This gives a baud rate clock tick jitter, but the average period has been increased by a fraction of 0.5 clock cycles.

Figure 23-9 on page 290 shows an example of how BSEL and BSCALE can be used to achieve baud rates in between what is possible by just changing BSEL.

The impact of fractional baud rate generation is that the step size between baud rate settings has been reduced. Given a scale factor of -1, the worst-case step then becomes from 160 to 240 clock cycles per 10-bit frame, compared to the previous step of from 160 to 320. A higher negative scale factor gives even finer granularity. There is a limit however, to how high the scale factor can be. The value $2^{|BSCALE|}$ must be at most half the minimum number of clock cycles of a frame. For instance, for 10-bit frames, the minimum number of clock cycles is 160. This means that the highest applicable scale factor is -6 ($2^{|-6|} = 64 < (160/2) = 80$).

For higher BSEL settings, the scale factor can be increased.

Table 23-5 on page 290 shows BSEL and BSCALE settings when using the internal oscillators to generate the most commonly used baud rates for asynchronous operation and how reducing the BSCALE can be used to reduce the baud rate error even further.

Figure 23-9. Fractional baud rate example.

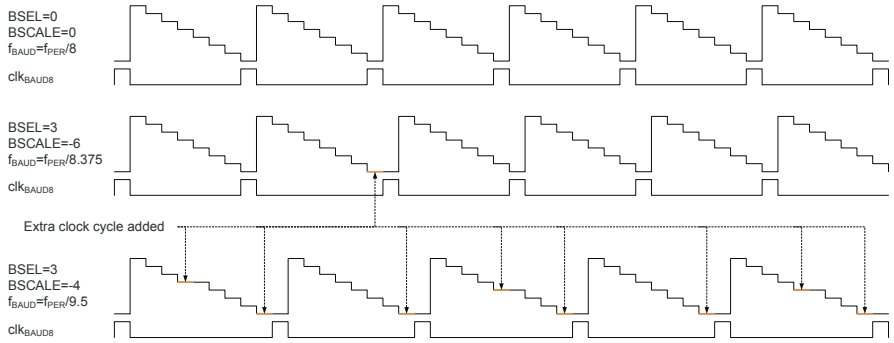


Table 23-5. USART baud rate.

Baud rate (bps)	f_{OSC} = 32.0000MHz					
	CLK2X = 0			CLK2X = 1		
	BSEL	BSCALE	Error [%]	BSEL	BSCALE	Error [%]
2400	12	6	0.2	12	7	0.2
4800	12	5	0.2	12	6	0.2
9600	12	4	0.2	12	5	0.2
14.4k	34	2	0.8	34	3	0.8
	138	0	-0.1	138	1	-0.1
19.2k	12	3	0.2	12	4	0.2
28.8k	34	1	-0.8	34	2	-0.8
	137	-1	-0.1	138	0	-0.1
38.4k	12	2	0.2	12	3	0.2
57.6k	34	0	-0.8	34	1	-0.8
	135	-2	-0.1	137	-1	-0.1
76.8k	12	1	0.2	12	2	0.2
115.2k	33	-1	-0.8	34	0	-0.8
	131	-3	-0.1	135	-2	-0.1
230.4k	31	-2	-0.8	33	-1	-0.8
	123	-4	-0.1	131	-3	-0.1
460.8k	27	-3	-0.8	31	-2	-0.8
	107	-5	-0.1	123	-4	-0.1
921.6k	19	-4	-0.8	27	-3	-0.8
	75	-6	-0.1	107	-5	-0.1
1.382M	7	-4	0.6	15	-3	0.6
	57	-7	0.1	121	-6	0.1
1.843M	3	-5	-0.8	19	-4	-0.8
	11	-7	-0.1	75	-6	-0.1
2.00M	0	0	0.0	1	0	0.0
2.304M	–	–	–	3	-2	-0.8
				47	-6	-0.1
2.5M	–	–	–	19	-4	0.4
				77	-7	-0.1
3.0M	–	–	–	11	-5	-0.8
				43	-7	-0.2
4.0M	–	–	–	0	0	0.0
Max	2.0Mbps			4.0Mbps		

23.10 USART in Master SPI Mode

Using the USART in master SPI mode requires the transmitter to be enabled. The receiver can optionally be enabled to serve as the serial input. The XCK pin will be used as the transfer clock.

As for the USART, a data transfer is initiated by writing to the DATA register. This is the case for both sending and receiving data, since the transmitter controls the transfer clock. The data written to DATA are moved from the transmit buffer to the shift register when the shift register is ready to send a new frame.

The transmitter and receiver interrupt flags and corresponding USART interrupts used in master SPI mode are identical in function to their use in normal USART operation. The receiver error status flags are not in use and are always read as zero.

Disabling of the USART transmitter or receiver in master SPI mode is identical to their disabling in normal USART operation.

23.11 USART SPI vs. SPI

The USART in master SPI mode is fully compatible with the standalone SPI module in that:

- Timing diagrams are the same
- UCPHA bit functionality is identical to that of the SPI CPHA bit
- UDORD bit functionality is identical to that of the SPI DORD bit

When the USART is set in master SPI mode, configuration and use are in some cases different from those of the standalone SPI module. In addition, the following differences exist:

- The USART transmitter in master SPI mode includes buffering, but the SPI module has no transmit buffer
- The USART receiver in master SPI mode includes an additional buffer level
- The USART in master SPI mode does not include the SPI write collision feature
- The USART in master SPI mode does not include the SPI double speed mode feature, but this can be achieved by configuring the baud rate generator accordingly
- Interrupt timing is not compatible
- Pin control differs due to the master-only operation of the USART in SPI master mode

A comparison of the USART in master SPI mode and the SPI pins is shown Table 23-6.

Table 23-6. Comparison of USART in master SPI mode and SPI pins.

USART	SPI	Comment
TxD	MOSI	Master out only
RxD	MISO	Master in only
XCK	SCK	Functionally identical
N/A	$\overline{SS}$	Not supported by USART in master SPI mode

23.12 Multiprocessor Communication Mode

The multiprocessor communication mode effectively reduces the number of incoming frames that have to be handled by the receiver in a system with multiple microcontrollers communicating via the same serial bus. In this mode, a dedicated bit in the frames is used to indicate whether the frame is an address or data frame type.

If the receiver is set up to receive frames that contain five to eight data bits, the first stop bit is used to indicate the frame type. If the receiver is set up for frames with nine data bits, the ninth bit is used. When the frame type bit is one, the frame contains an address. When the frame type bit is zero, the frame is a data frame. If 5-bit to 8-bit character frames are used, the transmitter must be set to use two stop bits, since the first stop bit is used for indicating the frame type.

If a particular slave MCU has been addressed, it will receive the following data frames as usual, while the other slave MCUs will ignore the frames until another address frame is received.

23.12.1 Using Multiprocessor Communication Mode

The following procedure should be used to exchange data in multiprocessor communication mode (MPCM):

1. All slave MCUs are in multiprocessor communication mode.
2. The master MCU sends an address frame, and all slaves receive and read this frame.
3. Each slave MCU determines if it has been selected.

4. The addressed MCU will disable MPCM and receive all data frames. The other slave MCUs will ignore the data frames.
5. When the addressed MCU has received the last data frame, it must enable MPCM again and wait for a new address frame from the master.

The process then repeats from step 2.

Using any of the 5-bit to 8-bit character frame formats is impractical, as the receiver must change between using n and n+1 character frame formats. This makes full-duplex operation difficult, since the transmitter and receiver must use the same character size setting.

23.13 IRCOM Mode of Operation

IRCOM mode can be enabled to use the IRCOM module with the USART. This enables IrDA 1.4 compliant modulation and demodulation for baud rates up to 115.2kbps. When IRCOM mode is enabled, double speed mode cannot be used for the USART.

For devices with more than one USART, IRCOM mode can be enabled for only one USART at a time. For details, refer to "IRCOM – IR Communication Module" on page 301.

23.14 DMA Support

DMA support is available on UART, USRT, and master SPI mode peripherals. For details on different USART DMA transfer triggers, refer to "Transfer Triggers" on page 55.

23.15 Register Description

23.15.1 DATA – Data register

Bit	7	6	5	4	3	2	1	0
+0x00	RXB[[7:0] / TXB[[7:0]							
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 USART transmit data buffer register (TXB) and USART receive data buffer register (RXB) share the same I/O address and is referred to as USART data register (DATA). The TXB register is the destination for data written to the DATA register location. Reading the DATA register location returns the contents of the RXB register.

For 5-bit, 6-bit, or 7-bit characters, the upper unused bits will be ignored by the transmitter and set to zero by the receiver.

The transmit buffer can be written only when DREIF in the STATUS register is set. Data written to the DATA register when DREIF is not set will be ignored by the USART transmitter. When data are written to the transmit buffer and the transmitter is enabled, the transmitter will load the data into the transmit shift register when the shift register is empty. The data are then transmitted on the TxD pin.

The receive buffer consists of a two-level FIFO. Always read STATUS before DATA in order to get the correct status of the receive buffer.

23.15.2 STATUS – Status register

Bit	7	6	5	4	3	2	1	0
+0x01	RXCIF	TXCIF	DREIF	FERR	BUFOVF	PERR	–	RXB8
Read/Write	R	R/W	R	R	R	R	R	R/W
Initial Value	0	0	1	0	0	0	0	0

Bit 7 – RXCIF: Receive Complete Interrupt Flag
This flag is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). When the receiver is disabled, the receive buffer will be flushed, and consequently RXCIF will become zero.

When interrupt-driven data reception is used, the receive complete interrupt routine must read the received data from DATA in order to clear RXCIF. If not, a new interrupt will occur directly after the return from the current interrupt. This flag can also be cleared by writing a one to its bit location.

Bit 6 – TXCIF: Transmit Complete Interrupt Flag
This flag is set when the entire frame in the transmit shift register has been shifted out and there are no new data in the transmit buffer (DATA). TXCIF is automatically cleared when the transmit complete interrupt vector is executed. The flag can also be cleared by writing a one to its bit location.

Bit 5 – DREIF: Data Register Empty Flag
This flag indicates whether the transmit buffer (DATA) is ready to receive new data. The flag is one when the transmit buffer is empty and zero when the transmit buffer contains data to be transmitted that has not yet been moved into the shift register. DREIF is set after a reset to indicate that the transmitter is ready. Always write this bit to zero when writing the STATUS register.

DREIF is cleared by writing DATA. When interrupt-driven data transmission is used, the data register empty interrupt routine must either write new data to DATA in order to clear DREIF or disable the data register empty interrupt. If not, a new interrupt will occur directly after the return from the current interrupt.

Bit 4 – FERR: Frame Error
The FERR flag indicates the state of the first stop bit of the next readable frame stored in the receive buffer. The bit is set if the received character had a frame error, i.e., the first stop bit was zero, and cleared when the stop bit of the received data is one. This bit is valid until the receive buffer (DATA) is read. FERR is not affected by setting the number of stop bits used, as it always uses only the first stop bit. Always write this bit location to zero when writing the STATUS register.

This flag is not used in master SPI mode operation.

Bit 3 – BUFOVF: Buffer Overflow
This flag indicates data loss due to a receiver buffer full condition. This flag is set if a buffer overflow condition is detected. A buffer overflow occurs when the receive buffer is full (two characters) with a new character waiting in the receive shift register and a new start bit is detected. This flag is valid until the receive buffer (DATA) is read. Always write this bit location to zero when writing the STATUS register.

This flag is not used in master SPI mode operation.

Bit 2 – PERR: Parity Error
If parity checking is enabled and the next character in the receive buffer has a parity error, this flag is set. If parity check is not enabled, this flag will always be read as zero. This bit is valid until the receive buffer (DATA) is read. Always write this bit location to zero when writing the STATUS register. For details on parity calculation, refer to "Parity Bit Calculation" on page 285.

This flag is not used in master SPI mode operation.

Bit 1 – Reserved
This bit is unused and reserved for future use. For compatibility with future devices, always write this bit to zero when this register is written.

Bit 0 – RXB8: Receive Bit 8
RXB8 is the ninth data bit of the received character when operating with serial frames with nine data bits. When used, this bit must be read before reading the low bits from DATA.

This bit is unused in master SPI mode operation.

23.15.3 CTRLA – Control register A

Bit	7	6	5	4	3	2	1	0
+0x03	–	–	RXCINTLVL[1:0]		TXCINTLVL[1:0]		DREINTLVL[1:0]	
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

Bit 7:6 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 5:4 – RXCINTLVL[1:0]: Receive Complete Interrupt Level
These bits enable the receive complete interrupt and select the interrupt level, as described in "Interrupts and Programmable Multilevel Interrupt Controller" on page 131. The enabled interrupt will be triggered when the RXCIF flag in the STATUS register is set.

Bit 3:2 – TXCINTLVL[1:0]: Transmit Complete Interrupt Level
These bits enable the transmit complete interrupt and select the interrupt level, as described in "Interrupts and Programmable Multilevel Interrupt Controller" on page 131. The enabled interrupt will be triggered when the TXCIF flag in the STATUS register is set.

Bit 1:0 – DREINTLVL[1:0]: Data Register Empty Interrupt Level
These bits enable the data register empty interrupt and select the interrupt level, as described in "Interrupts and Programmable Multilevel Interrupt Controller" on page 131. The enabled interrupt will be triggered when the DREIF flag in the STATUS register is set.

23.15.4 CTRLB – Control register B

Bit	7	6	5	4	3	2	1	0
+0x04	–	–	–	RXEN	TXEN	CLK2X	MPCM	TXB8
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:5 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 4 – RXEN: Receiver Enable
Setting this bit enables the USART receiver. The receiver will override normal port operation for the RxD pin, when enabled. Disabling the receiver will flush the receive buffer, invalidating the FERR, BUFOVF, and PERR flags.

Bit 3 – TXEN: Transmitter Enable
Setting this bit enables the USART transmitter. The transmitter will override normal port operation for the TxD pin, when enabled. Disabling the transmitter (writing TXEN to zero) will not become effective until ongoing and pending transmissions are completed; i.e., when the transmit shift register and transmit buffer register do not contain data to be transmitted. When disabled, the transmitter will no longer override the TxD port.

Bit 2 – CLK2X: Double Transmission Speed
Setting this bit will reduce the divisor of the baud rate divider from16 to 8, effectively doubling the transfer rate for asynchronous communication modes. For synchronous operation, this bit has no effect and should always be written to zero. This bit must be zero when the USART communication mode is configured to IRCOM.
This bit is unused in master SPI mode operation.

Bit 1 – MPCM: Multiprocessor Communication Mode
This bit enables the multiprocessor communication mode. When the MPCM bit is written to one, the USART receiver ignores all the incoming frames that do not contain address information. The transmitter is unaffected by the MPCM setting. For more detailed information, see "Multiprocessor Communication Mode" on page 292.
This bit is unused in master SPI mode operation.

Bit 0 – TXB8: Transmit Bit 8
TXB8 is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. When used, this bit must be written before writing the low bits to DATA.
This bit is unused in master SPI mode operation.

23.15.5 CTRLC – Control register C

Bit	7	6	5	4	3	2	1	0
+0x05	CMODE[1:0]		PMODE[1:0]		SBMODE	CHSIZE[2:0]		
+0x05[1]	CMODE[1:0]		–	–	–	UDORD	UCPHA	–
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	1	1
	0	0	0	0	0	1	1	0

Note: 1. Master SPI mode

Bits 7:6 – CMODE[1:0]: Communication Mode
These bits select the mode of operation of the USART as shown in Table 23-7.

Table 23-7. CMODE bit settings.

CMODE[1:0]	Group configuration	Mode
00	ASYNCHRONOUS	Asynchronous USART
01	SYNCHRONOUS	Synchronous USART
10	IRCOM	IRCOM[1]
11	MSPI	Master SPI[2]

Notes: 1. See "IRCOM – IR Communication Module" on page 301 for full description on using IRCOM mode.
2. See "USART in Master SPI Mode" on page 291 for full description of the master SPI operation.

Bits 5:4 – PMODE[1:0]: Parity Mode
These bits enable and set the type of parity generation according to Table 23-8 on page 297. When enabled, the transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The receiver will generate a parity value for the incoming data and compare it to the PMODE setting, and if a mismatch is detected, the PERR flag in STATUS will be set.
These bits are unused in master SPI mode operation.

Table 23-8. PMODE bit settings.

PMODE[1:0]	Group configuration	Parity Mode
00	DISABLED	Disabled
01	–	Reserved
10	EVEN	Enabled, even parity
11	ODD	Enabled, odd parity

Bit 3 – SBMODE: Stop Bit Mode
This bit selects the number of stop bits to be inserted by the transmitter according to Table 23-9 on page 297. The receiver ignores this setting.
This bit is unused in master SPI mode operation.

Table 23-9. SBMODE bit settings.

SBMODE	Stop Bit(s)
0	1
1	2

Bit 2:0 – CHSIZE[2:0]: Character Size
The CHSIZE[2:0] bits set the number of data bits in a frame according to Table 23-10 on page 298. The receiver and transmitter use the same setting.

Table 23-10. CHSIZE bit settings.

CHSIZE[2:0]	Group configuration	Character size
000	5BIT	5-bit
001	6BIT	6-bit
010	7BIT	7-bit
011	8BIT	8-bit
100	–	Reserved
101	–	Reserved
110	–	Reserved
111	9BIT	9-bit

Bit 2 – UDORD: Data Order
This bit is only for master SPI mode, and this bit sets the frame format. When written to one, the lsb of the data word is transmitted first. When written to zero, the msb of the data word is transmitted first. The receiver and transmitter use the same setting. Changing the setting of UDORD will corrupt all ongoing communication for both receiver and transmitter.

Bit 1 – UCPHA: Clock Phase
This bit is only for master SPI mode, and the bit determine whether data are sampled on the leading (first) edge or tailing (last) edge of XCKn. Refer to the "Master SPI Mode Clock Generation" on page 284 for details.

23.15.6 BAUDCTRLA – Baud Rate register A

Bit	7	6	5	4	3	2	1	0
+0x06	BSEL[7:0]							
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 7:0 – BSEL[7:0]: Baud Rate bits
These are the lower 8 bits of the 12-bit BSEL value used for USART baud rate setting. BAUDCTRLB contains the four most-significant bits. Ongoing transmissions by the transmitter and receiver will be corrupted if the baud rate is changed. Writing BSEL will trigger an immediate update of the baud rate prescaler. See the equations in Table 23-1 on page 282.

23.15.7 BAUDCTRLB – Baud Rate register B

Bit	7	6	5	4	3	2	1	0
+0x07	BSCALE[3:0]				BSEL[11:8]			
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

Bit 7:4 – BSCALE[3:0]: Baud Rate Scale factor
These bits select the baud rate generator scale factor. The scale factor is given in two's complement form from -7 (0b1001) to +7 (0b0111). The -8 (0b1000) setting is reserved. See the equations in Table 23-1 on page 282.

Bit 3:0 – BSEL[11:8]: Baud Rate bits
These are the upper 4 bits of the 12-bit value used for USART baud rate setting. BAUDCTRLA contains the eight least-significant bits. Ongoing transmissions by the transmitter and receiver will be corrupted if the baud rate is changed. Writing BAUDCTRLA will trigger an immediate update of the baud rate prescaler.

23.16 Register summary

23.16.1 Register description – USART

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	DATA	DATA[7:0]								294
+0x01	STATUS	RXCIF	TXCIF	DREIF	FERR	BUFOVF	PERR	–	RXB8	294
+0x02	Reserved	–	–	–	–	–	–	–	–	
+0x03	CTRLA	–	–	RXCINTLVL[1:0]		TXCINTLVL[1:0]		DREINTLVL[1:0]		295
+0x04	CTRLB	–	–	–	RXEN	TXEN	CLK2X	MPCM	TXB8	296
+0x05	CTRLC	CMODE[1:0]		PMODE[1:0]		SBMODE	CHSIZE[2:0]			296
+0x06	BAUDCTRLA	BSEL[7:0]								298
+0x07	BAUDCTRLB	BSCALE[3:0]				BSEL[11:8]				298

23.16.2 Register description – USART in SPI Master Mode

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	DATA	DATA[7:0]								294
+0x01	STATUS	RXCIF	TXCIF	DREIF	–	–	–	–	–	294
+0x02	Reserved	–	–	–	–	–	–	–	–	
+0x03	CTRLA	–	–	RXCINTLVL[1:0]		TXCINTLVL[1:0]		DREINTLVL[1:0]		295
+0x04	CTRLB	–	–	–	RXEN	TXEN	–	–	–	296
+0x05	CTRLC	CMODE[1:0]		–	–	–	UDORD	UCPHA	–	296
+0x06	BAUDCTRLA	BSEL[7:0]								298
+0x07	BAUDCTRLB	BSCALE[3:0]				BSEL[11:8]				298

23.17 Interrupt vector summary

Table 23-11. USART interrupt vectors and their word offset address.

Offset	Source	Interrupt description
0x00	RXC_vect	USART receive complete interrupt vector
0x02	DRE_vect	USART data register empty interrupt vector
0x04	TXC_vect	USART transmit complete interrupt vector

27. EBI – External Bus Interface

27.1 Features

- Supports SRAM up to:
  - 512KB using 2-port EBI
  - 16MB using 3-port EBI
- Supports SDRAM up to:
  - 128Mb using 3-port EBI
- Four software configurable chip selects
- Software configurable wait state insertion
- Can run from the 2x peripheral clock frequency for fast access

27.2 Overview

The External Bus Interface (EBI) is used to connect external peripherals and memory for access through the data memory space. When the EBI is enabled, data address space outside the internal SRAM becomes available using dedicated EBI pins.

The EBI can interface external SRAM, SDRAM, and peripherals, such as LCD displays and other memory mapped devices.

The address space for the external memory is selectable from 256 bytes (8-bit) up to 16MB (24-bit). Various multiplexing modes for address and data lines can be selected for optimal use of pins when more or fewer pins are available for the EBI. The complete memory will be mapped into one linear data address space continuing from the end of the internal SRAM. Refer to "Data Memory" on page 22 for details.

The EBI has four chip selects, each with separate configuration. Each can be configured for SRAM, SRAM low pin count (LPC), or SDRAM.

The EBI is clocked from the fast, 2x peripheral clock, running up to two times faster than the CPU.

Four-bit and eight-bit SDRAM are supported, and SDRAM configurations, such as CAS latency and refresh rate, are configurable in software.

For more details on SRAM and SDRAM, and on how these memory types are organized and work, refer to SRAM and SDRAM-specific documentation and datasheets. This section only contains EBI-specific details.

27.3 Chip Select

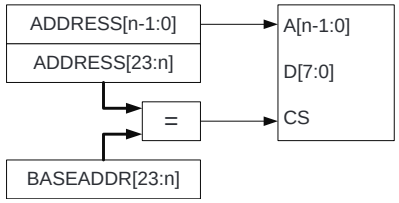
The EBI module has four chip select lines (CS0 to CS3), which can be associated with separate address ranges. The chip selects control which memory or memory mapped external hardware is accessed when a given memory address is issued on the EBI. Each chip select has separate configuration, and can be configured for SRAM or SRAM low pin count (LPC). Chip select 3 can also be configured for SDRAM.

Each chip select has a configurable base address and address size, which are used to determine the data memory address space associated with each chip select.

27.3.1 Base Address

The base address assigned to a chip select is the lowest address in the address space, and determines the first location in data memory space where the connected memory hardware can be accessed. The base address associated with each chip select must be on a 4KB boundary.

Figure 27-1. Base Address



27.3.2 Address Size

The address size selects how many bits of the address should be compared when generating a chip select. The address size can be anywhere from 256 bytes to 16MB. If the address space is set to anything larger than 4KB, the base address must be on a boundary equal to the address space. For example, with 1MB address space for a chip select, the base address must be on a 1MB, 2MB, etc. boundary.

If the EBI is configured so that the address spaces overlap, the internal memory space will have priority, followed by chip select 0 (CS0), CS1, CS2, and CS3.

27.3.3 Chip Select as Address Lines

If any chip select lines are unused, these can, in some combinations, be used as address lines. This enables larger external memory or external CS generation. Each column in Figure 27-2 on page 320 shows enabled chip select lines (CSn) and the address lines available on unused chip select lines (An). The right-hand column shows that all four CS lines are used as address lines when only CS3 is enabled.

Figure 27-2. Chip Select and address line combinations

CS3	CS3	CS3	A19
CS2	CS2	CS2	A18
CS1	CS1	A17	A17
CS0	A16	A16	A16

27.4 EBI Clock

The EBI is clocked from the Peripheral 2x (Clk_{PER2}) Clock. This clock can run at the CPU Clock frequency, or at two times the CPU Clock frequency. This can be used to lower the EBI access time. Refer to "System Clock and Clock Options" on page 82 for details the Peripheral 2x Clock and how to configure this.

27.5 SRAM Configuration

When used with SRAM, the EBI can be configured with no multiplexing, or it can employ various address multiplexing modes by using external address latches. When a limited number of pins are available on the device for the EBI, address latch enable (ALE) signals are used to control the external latches that multiplex address lines from the EBI. The

available configurations are shown in "No Multiplexing" on page 321 through "Multiplexing address byte 0, 1and 2" on page 322. Table 27-1 on page 321 describes the SRAM interface signals.

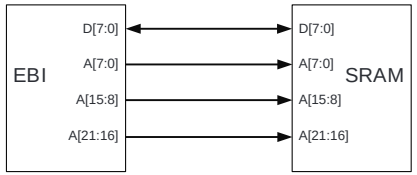
Table 27-1. SRAM Interface signals.

Signal	Description
CS	Chip Select
WE	Write Enable
RE	Read Enable
ALE[2:1]	Address Latch Enable
A[23:0]	Address
D[7:0]	Data bus
AD[7:0]	Combined Address and Data

27.5.1 No Multiplexing

When no multiplexing is used, there is a one-to-one connection between the EBI and the SRAM. No external address latches are used.

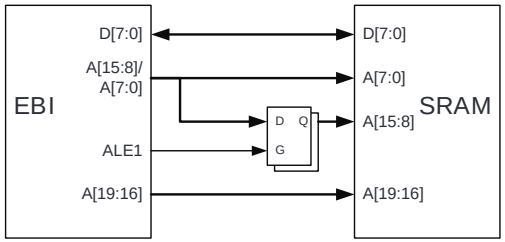
Figure 27-3. Non-multiplexed SRAM connection.



27.5.2 Multiplexing address byte 0 and 1

When address byte 0 (A[7:0]) and address byte 1 (A[15:8]) are multiplexed, they are output from the same port, and the ALE1 signal from the device controls the address latch.

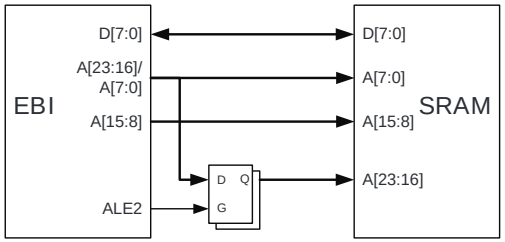
Figure 27-4. Multiplexed SRAM connection using ALE1.



27.5.3 Multiplexing address byte 0 and 2

When address byte 0 (A[7:0]) and address byte 2 (A[23:16) are multiplexed, they are output from the same port, and the ALE2 signal from the device controls the address latch.

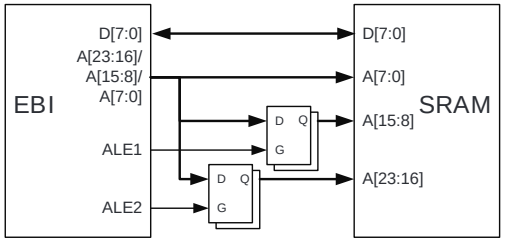
Figure 27-5. Multiplexed SRAM connection using ALE2.



27.5.4 Multiplexing address byte 0, 1and 2

When address byte 0 (A[7:0]), address byte 1 (A[15:8]) and address byte 2 (A[23:16] are multiplexed, they are output from the same port, and the ALE1 and ALE2 signal from the device control the external address latches.

Figure 27-6. Multiplexed SRAM connection using ALE1 and ALE2.



27.5.5 Address Latches

The Address Latch timing and parameter requirements are described in EBI Timing. See the device datasheet characteristics for details. To reduce access time when using multiplexing of address, the ALE signals are only issued when it is required to update the latched address. For instance if address lines A[15:8] are multiplexed with A[7:0] the ALE1 and A[15:8] are only given if any bit in A[15:8] are changed since the last time ALE was set.

27.5.6 Timing

SRAM or external memory devices may have different timing requirements. To meet these varying requirements, each Chip Select can be configured with different wait-states. Timing details are described in the device datasheet.

27.6 SRAM LPC Configuration

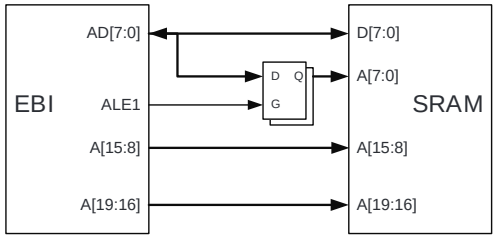
The SRAM Low Pin Count (LPC) configuration enables EBI to be configured for multiplexing modes where the data and address lines are multiplexed. Compared to SRAM configuration, this can further reduce the number of pins required for the EBI. The available configurations is shown in "Multiplexing Data with Address Byte 0" on page 323 through "Multiplexing Data with Address Byte 0 and 1" on page 323.

Timing and Address Latch requirements is as for SRAM configuration.

27.6.1 Multiplexing Data with Address Byte 0

When the data byte and address byte 0 (AD[7:0]) are multiplexed, they are output from the same port, and the ALE1 signal from the device controls the address latch.

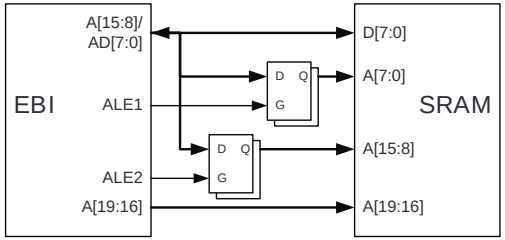
Figure 27-7. Multiplexed SRAM LPC connection using ALE1.



27.6.2 Multiplexing Data with Address Byte 0 and 1

When the data byte and address byte 0 (AD[7:0]), and address byte 1 (A[15:8]) are multiplexed, they are output from the same port, and the ALE1 and ALE2 signal from the device control the external address latches.

Figure 27-8. Multiplexed SRAM LPC connection using ALE1 and ALE2.



27.7 SDRAM Configuration

Chip Select 3 on the EBI can be configured from SDRAM operation, and the EBI must be configured as a three-port or four-port interface. The SDRAM can be configured for 4-bit or 8-bit data bus, and four-Port interface must be used for 8-bit data bus. The SDRAM interface signals from the EBI to the SDRAM is listed in Table 27-2 on page 324.

Table 27-2. SDRAM Interface signals

Signal	Description
CS	Chip select
WE	Write enable
RAS	Row address strobe
CAS	Column address strobe
DQM	Data mask signal/ output enable
CKE	Clock enable
CLK	Clock
BA[1:0]	Bank address
A[12:0]	Address bus
A[10]	Precharge
D[7:0]	Data bus

27.7.1 Supported Commands

The SDRAM commands that are supported by the EBI is listed in Table 27-3 on page 324.

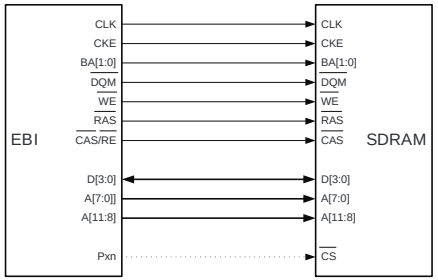
Table 27-3. Supported SDRAM commands.

Command	Description
NOP	No Operation
ACTIVE	Activate the selected bank and select the row
READ	Input the starting column address and begin the burst read operation
WRITE	Input the starting column address and begin the burst write operation
PRECHARGE	Deactivate the open row of selected bank or all banks
AUTO REFRESH	Refresh one row of each bank
LOAD MODE	Load mode register
SELF REFRESH	Activate self refresh mode

27.7.2 Three-Port EBI Configuration

When three EBI ports are available, SDRAM can be connected with a three-Port EBI configuration. When this is done only four-bit data bus is available, and any chip select must be controlled from software using a general purpose I/O pin (Pxn).

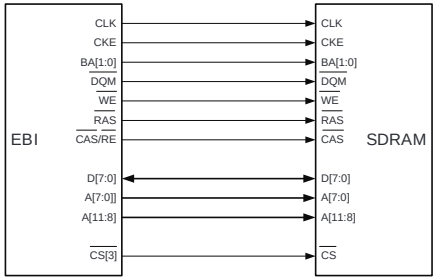
Figure 27-9. Three-Port SDRAM configuration.



27.7.3 Four-Port EBI Configuration

When four EBI ports are available, SDRAM can be connected with a three-port or four-port EBI configuration. When a four-port configuration is used, an eight-bit data bus is available, and all four chip selects will be available.

Figure 27-10.Four-Port SDRAM configuration.



27.7.4 Timing

The Clock Enable (CKE) signal is required for SDRAM when the EBI is clocked at 2x the CPU clock speed.

27.7.5 Initialization

Configuring Chip Select 3 to SDRAM will enable the initialization of the SDRAM. The Load Mode Register command is automatically issued at the end of the initialization. For correct information to be loaded to the SDRAM, one of the following must be done:

1. Configure the SDRAM control registers before enabling chip select 3 to SDRAM
2. Issue a Load Mode Register command, and perform a dummy access after the SDRAM is initialized

The SDRAM initialization is not interruptible by other EBI accesses.

27.7.6 Refresh

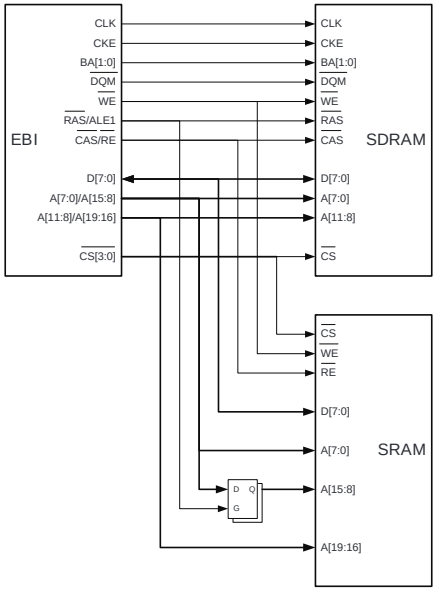
The EBI will automatically handle the SDRAM refresh as long as the refresh period is configured. On average will one refresh command be issues at the interval given by the SDRAM Refresh Period Register. The EBI can collect up to four

refresh commands in case the interface is busy on another chip select or in the middle of a read/write at the time a refresh should have been performed.

27.8 Combined SRAM & SDRAM Configuration

Combined SRAM and SDRAM configuration enables the EBI to have both SDRAM and SRAM connected at the same time. This is available only when using a four-port EBI interface. Figure 27-11 on page 326 shows the configuration, with all interface signals.

Figure 27-11.Combined SRAM and SDRAM connection



27.9 I/O Pin and Pin-out Configuration

When the EBI is enabled, it will override the direction and/or value of the I/O pins where the EBI data lines are placed. The EBI will also override the value, but not the direction, of the I/O pins where the EBI address and control lines are placed. These I/O pins must be configured to output when the EBI is used. I/O pins for unused EBI address and control lines can be used as normal I/O pins or for other alternate functions on the pins.

For control signals that are active-low, the pin output value should be set to one (high). For control signals that are active-high, the pin output value should be set to zero (low). Address lines do not require specific pin output value configuration. The chip select lines should have pull-up resistors to ensure that they are kept high during power on and reset. If a chip select line is active-high, a pull-down resistor should be used instead of a pull-up.

The pin-out for the fourth EBI port can be configured with the EBIOUT register.

For more details on I/O pin configuration, refer to "I/O Ports" on page 139.

The tables below summaries the actual port pin-out for the various SRAM and SDRAM configurations, and shows required pins and pin usage. Refer to the device datasheet to see which actual I/O ports are used as EBI PORT0-3 for a specific AVR XMEGA device.

Table 27-4. Pin-out SRAM.

PORT	PIN	SRAM 3PORT ALE1	SRAM 3PORT ALE12	SRAM 4PORT ALE2	SRAM 4PORT NOALE
PORT3	7:0	–	–	A[15:8]	A[15:8]
PORT2	7:0	A[7:0]/ A[15:8]	A[7:0]/ A[15:8]/ A[23:16]	A[7:0]/ A[23:16]	A[7:0]
PORT1	7:0	D[7:0]	D[7:0]	D[7:0]	D[7:0]
PORT0	7:4	$\overline{CS}$[3:0] (A[19:16])	$\overline{CS}$[3:0]	$\overline{CS}$[3:0]	$\overline{CS}$[3:0] (A[21:18])
	3	–	ALE2	ALE2	A17
	2	ALE1	ALE1	–	A16
	1	RE	RE	RE	RE
	0	WE	WE	WE	WE

Table 27-5. Pin-out SRAM LPC.

PORT	PIN	SRAM LPC 2PORT ALE1	SRAM LPC 3PORT/4PORT ALE1	SRAM 2/3/4PORT ALE12
PORT3	7:0	–	–	–
PORT2	7:0	–	A[15:8]	–
PORT1	7:0	D[7:0]/ A[7:0]	D[7:0]/ A[7:0]	D[7:0]/ A[7:0]/ A[15:8]
PORT0	7:4	$\overline{CS}$[3:0]	$\overline{CS}$[3:0] (A[19:16])	$\overline{CS}$[3:0] (A[19:16])
	3	–	–	ALE2
	2	ALE1	ALE1	–
	1	RE	RE	RE
	0	WE	WE	WE

Table 27-6. Pin-out for SRAM and SRAM LPC when combined with SDRAM (four-port only).

PORT	PIN	SRAM LPC ALE1 (with SDRAM)	SRAM LPC ALE12 (with SDRAM)	SRAM ALE1 (with SDRAM)	SRAM ALE12 (with SDRAM)
PORT3	7:4	$\overline{CS}$[3:0] (A[23:20])	$\overline{CS}$[3:0] (A[23:20])	$\overline{CS}$[3:0] (A[23:20])	$\overline{CS}$[3:0]
	3:0	A[15:8]	A[19:16]	A[19:16]	–
PORT2	7:0	A[15:8]	–	A[7:0]/ A[15:8]	A[7:0]/ A[15:8]/ A[23:16]
PORT1	7:0	D[7:0]/ A[7:0]	D[7:0]/ A[7:0]/ A[15:8]	D[7:0]	D[7:0]
PORT0	7:4	–	–	–	–
	3	–	ALE2	–	ALE2
	2	ALE1	ALE1	ALE1	ALE1
	1	RE	RE	RE	RE
	0	WE	WE	WE	WE

Table 27-7. Pin-out SDRAM.

PORT	PIN	SDRAM 3PORT 4BIT	SDRAM 4PORT 4BIT	SDRAM 4PORT 8BIT
PORT3	7:4	–	$\overline{CS}$[3:0]	$\overline{CS}$[3:0]
	3:0			A[11:8]
PORT2	7:0	A[7:0]	A[7:0]	A[7:0]
PORT1	7:4	A[11:8]	A[11:8]	D[7:4]
	3:0	D[3:0]	D[3:0]	D[3:0]
PORT0	7	CLK	CLK	CLK
	6	CKE	CKE	CKE
	5	BA1	BA1	BA1
	4	BA0	BA0	BA0
	3	DQM	DQM	DQM
	2	RAS	RAS	RAS
	1	CAS	CAS	CAS
	0	WE	WE	WE

27.10 Register Description – EBI

27.10.1 CTRL – Control register

Bit	7	6	5	4	3	2	1	0
+0x00	SDDATAW[1:0]		LPCMODE[1:0]		SRMODE[1:0]		IFMODE[1:0]	
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 7:6 – SDDATAW[1:0]: SDRAM Data Width Setting
These bits select the EBI SDRAM data width configuration, according to Table 27-8 on page 329.

Table 27-8. SDRAM mode.

SDDATAW[1:0]	Group configuration	Description
00	4BIT	Four-bit data bus
01	8BIT(1)	Eight-bit data bus
10	–	Reserved
11	–	Reserved

Note: 1. Eight-bit data bus only available for four-port EBI interface

Bit 5:4 – LPCMODE[1:0]: SRAM Low Pin Count Mode
These bits select the EBI SRAM LPC configuration according to Table 27-9 on page 329.

Table 27-9. SRAM LPC mode.

LPCMODE[1:0]	Group configuration	ALE	Description
00	ALE1	ALE1	Data multiplexed with Address byte 0
01	–	–	Reserved
10	ALE12	ALE1 & 2	Data multiplexed with Address byte 0 and 1
11	–	–	Reserved

Bit 3:2 – SRMODE[1:0]: SRAM Mode
These bits selects the EBI SRAM configuration according to Table 27-10 on page 329.

Table 27-10. SRAM mode.

SRMODE[1:0]	Group configuration	ALE	Description
00	ALE1	ALE1	Address byte 0 and 1 multiplexed
01	ALE2(1)	ALE2	Address byte 0 and 2 multiplexed
10	ALE12(1)	ALE1 & 2	Address byte 0, 1 and 2 multiplexed
11	NOALE	No ALE	No address multiplexing

Note: 1. ALE2 and NOALE only available with 4-port EBI interface

Bit 1:0 – IFMODE[1:0]: Interface Mode
These bits select EBI interface mode and the number of ports that should be enabled and overridden for EBI, according to Table 27-11 on page 330.

Table 27-11. EBI mode.

IFMODE[1:0]	Group configuration	Description
00	DISABLED	EBI disabled
01	3PORT	EBI enabled with three-port interface
10	4PORT	EBI enabled with four-port interface
11	2PORT	EBI enabled with two-port interface

27.10.2 SDRAMCTRLA – SDRAM Control register A

Bit	7	6	5	4	3	2	1	0
+0x01	–	–	–	–	SDCAS	SDROW	SDCOL[1:0]	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:4 – Reserved
These bits are unused and reserved for future use.

Bit 3 – SDCAS: SDRAM CAS Latency
This bit sets the CAS latency as a number of Clk_{PER2} cycles. By default this bit is zero and the CAS latency is two Clk_{PER2} cycles. When this bit is set to one, the CAS latency is three Clk_{PER2} cycles.

Table 27-12. SDRAM CAS latency.

SDROW	Group configuration	Description
0	2CLK	2 Clk_{PER2} cycles delay
1	3CLK	3 Clk_{PER2} cycles delay

Bit 2 – SDROW: SDRAM Row Bits
This bit sets the number of row bits used for the connected SDRAM. By default this bit is zero, and the row bit setting is set to 11 row bits. When this bit is set to one, the row bit setting is set to 12 row bits.

Table 27-13. SDRAM row bits.

SDROW	Group configuration	Description
0	11BIT	11 row bits
1	12BIT	12 row bits

Bit 1:0 – SDCOL[1:0]: SDRAM Column Bits
These bits select the number of column bits that are used for the connected SDRAM according to table. Table 27-14 on page 331.

Table 27-14. SDRAM column bits.

SDCOL[1:0]	Group configuration	Description
00	8BIT	8 column bits
01	9BIT	9 column bits
10	10BIT	10 column bits
11	11BIT	11 column bits

27.10.3 REFRESH – SDRAM Refresh Period Register

Bit	7	6	5	4	3	2	1	0
+0x04	REFRESH[7:0]							
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	6	5	4	3	2	1	0
+0x05	–	–	–	–	–	–	REFRESH[9-8]	
Read/Write	R	R	R	R	R	R	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 15:10 – Reserved
These bits are unused and reserved for future use.

Bit 9:0 – REFRESH[9:0]: SDRAM Refresh Period
This register sets the refresh period as a number of Clk_{PER2} cycles. If the EBI is busy with another external memory access at time of refresh, up to 4 refresh will be remembered and given at the first available time.

27.10.4 INITDLY – SDRAM Initialization Delay register

Bit	7	6	5	4	3	2	1	0
+0x06	INITDLY[7:0]							
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	6	5	4	3	2	1	0
+0x07	–	–	INITDLY[13-8]					
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

Bit 15:14 – Reserved
These bits are unused and reserved for future use.

Bit 13:0 – INITDLY[13:0]: SDRAM Initialization Delay
This register is used to delay the initialisation sequence after the controller is enabled until all voltages are stabilized and the SDRAM clock has been running long enough to take the SDRAM chip through its initialisation sequence. The initialisation sequence includes pre-charge all banks to their idle state issuing an auto-refresh cycle and then loading the mode register. The setting in this register is as a number of Clk_{PER2} cycles.

27.10.5 SDRAMCTRLB – SDRAM Control register B

Bit	7	6	5	4	3	2	1	0
+0x08	MRDLY[1:0]		ROWCYCDLY[2:0]			RPDLY[2:0]		
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 7:6 – MRDLY[1:0]: SDRAM Mode Delay
These bits select the delay between a LOAD MODE command and an ACTIVE command, in number of Clk_{PER2} cycles, according to Table 27-15 on page 332.

Table 27-15. SDRAM Load Mode to Active command delays settings.

MRDLY[1:0]	Group configuration	Description
00	0CLK	Zero Clk_{PER2} cycles delay
01	1CLK	One Clk_{PER2} cycles delay
10	2CLK	Two Clk_{PER2} cycles delay
11	3CLK	Three Clk_{PER2} cycles delay

Bit 5:3 – ROWCYCDLY[2:0]: SDRAM Row Cycle Delay
These bits select the delay between a REFRESH and an ACTIVE command in number of Clk_{PER2} cycles, according to Table 27-16 on page 332.

Table 27-16. SDRAM Row cycle delay settings.

ROWCYDLY[2:0]	Group configuration	Description
000	0CLK	Zero Clk_{PER2} cycles delay
001	1CLK	One Clk_{PER2} cycles delay
010	2CLK	Two Clk_{PER2} cycles delay
011	3CLK	Three Clk_{PER2} cycles delay
100	4CLK	Four Clk_{PER2} cycles delay
101	5CLK	Five Clk_{PER2} cycles delay
110	6CLK	Six Clk_{PER2} cycles delay
111	7CLK	seven Clk_{PER2} cycles delay

Bit 2:0 – RPDLY[2:0]: SDRAM Row to Precharge Delay
RPDLY defines the delay between an Active command and a Precharge command in number of Clk_{PER2} cycles, according to Table 27-17 on page 333.

Table 27-17. SDRAM row to precharge delay settings.

RPDLY[2:0]	Group configuration	Description
000	0CLK	Zero Clk_{PER2} cycles delay
001	1CLK	One Clk_{PER2} cycles delay
010	2CLK	Two Clk_{PER2} cycles delay
011	3CLK	Three Clk_{PER2} cycles delay
100	4CLK	Four Clk_{PER2} cycles delay
101	5CLK	Five Clk_{PER2} cycles delay
110	6CLK	Six Clk_{PER2} cycles delay
111	7CLK	Seven Clk_{PER2} cycles delay

27.10.6 SDRAMCTRLC – SDRAM Control register C

Bit	7	6	5	4	3	2	1	0
+0x09	WRDLY[1:0]		ESRDLY[1:0]			ROWCOLDLY[1:0]		
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 7:6 – WRDLY[1:0]: SDRAM Write Recovery Delay
These bits select the write recovery time in number of Clk_{PER2} cycles, according to Table 27-18 on page 333.

Table 27-18. SDRAM write recovery delay settings.

WRDLY[1:0]	Group configuration	Description
00	0CLK	Zero Clk_{PER2} cycles delay
01	1CLK	One Clk_{PER2} cycles delay
10	2CLK	Two Clk_{PER2} cycles delay
11	3CLK	Three Clk_{PER2} cycles delay

Bit 5:3 – ESRDLY[2:0]: SDRAM Exit Self-refresh to Active Delay
This field defines the delay between CKE set high and an ACTIVE command in a number of Clk_{PER2} cycles, according to Table 27-19 on page 333.

Table 27-19. SDRAM exit self-refresh delay settings.

ESRDLY[2:0]	Group configuration	Description
000	0CLK	Zero Clk_{PER2} cycles delay
001	1CLK	One Clk_{PER2} cycles delay
010	2CLK	Two Clk_{PER2} cycles delay
011	3CLK	Three Clk_{PER2} cycles delay
100	4CLK	Four Clk_{PER2} cycles delay

ESRDLY[2:0]	Group configuration	Description
101	5CLK	Five Clk_{PER2} cycles delay
110	6CLK	Six Clk_{PER2} cycles delay
111	7CLK	Seven Clk_{PER2} cycles delay

Bit 2:0 – ROWCOLDLY[2:0]: SDRAM Row to Column Delay
This field defines the delay between an Active command and a Read/Write command as a number of Clk_{PER2} cycles, according to Table 27-20 on page 334.

Table 27-20. SDRAM row column delay settings.

ROWCOLDLY[2:0]	Group configuration	Description
000	0CLK	Zero Clk_{PER2} cycles delay
001	1CLK	One Clk_{PER2} cycles delay
010	2CLK	Two Clk_{PER2} cycles delay
011	3CLK	Three Clk_{PER2} cycles delay
100	4CLK	Four Clk_{PER2} cycles delay
101	5CLK	Five Clk_{PER2} cycles delay
110	6CLK	Six Clk_{PER2} cycles delay
111	7CLK	seven Clk_{PER2} cycles delay

27.11 Register Description – EBI Chip Select

27.11.1 CTRLA – Control register A

Bit	7	6	5	4	3	2	1	0
+0x00	–	ASIZE[4:0]					MODE[1:0]	
Read/Write	R	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 – Reserved
This bit is unused and reserved for future use.

Bit 6:2 – ASIZE[4:0]: Address Size
These bits select the address size for the Chip Select. This is the size of the block above the base address.

Table 27-21. Address size encoding.

ASIZE[4:0]	Group configuration	Address size	Address lines compared
00000	256B	256 bytes	ADDR[23:8]
00001	512B	512 bytes	ADDR[23:9]
00010	1K	1KB	ADDR[23:10]
00011	2K	2KB	ADDR[23:11]
00100	4K	4KB	ADDR[23:12]
00101	8K	8KB	ADDR[23:13]
00110	16K	16KB	ADDR[23:14
00111	32K	32KB	ADDR[23:15]
01000	64K	64KB	ADDR[23:16]
01001	128K	128KB	ADDR[23:17]
01010	256K	256KB	ADDR[23:18]
01011	512K	512KB	ADDR[23:19]
01100	1M	1MB	ADDR[23:20]
01101	2M	2MB	ADDR[23:21]
01110	4M	4MB	ADDR[23:22]
01111	8M	8MB	ADDR[23]
10000	16M	16MB(1)	–
Other	–	–	Reserved

Note: 1. Entire available data space used.

Bit 1:0 – MODE[1:0]: Chip Select Mode
These bits select the Chip Select Mode and decide what type of interface is used for the external memory or peripheral according to Table 27-22 on page 336.

Table 27-22. Chip Select Mode selection.

MODE[1:0]	Group configuration	Description
00	DISABLE	Chip select disabled
01	SRAM	Enable chip select for SRAM
10	LPC	Enable chip select for SRAM LPC
11	SDRAM	Enable chip select for SDRAM(1)

Note: 1. SDRAM can only be selected for CS3

27.11.2 CTRLB (SRAM) – Control register B

The configuration options for this register depend on the chip select mode configuration. The register description below is valid when the chip select mode is configured for SRAM or SRAM LPC.

Bit	7	6	5	4	3	2	1	0
+0x01	–	–	–	–	–	SRWS[2:0]		
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:3 – Reserved
These bits are unused and reserved for future use.

Bit 2:0 – SRWS[2:0]: SRAM Wait State
These bits select the number of wait states for SRAM and SRAM LPC access as a number of Clk_{PER2} cycles, according to Table 27-23 on page 336.

Table 27-23. Wait state selection.

SRWS[2:0]	Group configuration	Description
000	0CLK	Zero Clk_{PER2} cycles wait state
001	1CLK	One Clk_{PER2} cycles wait state
010	2CLK	Two Clk_{PER2} cycles wait state
011	3CLK	Three Clk_{PER2} cycles wait state
100	4CLK	Four Clk_{PER2} cycles wait state
101	5CLK	Five Clk_{PER2} cycles wait state
110	6CLK	Six Clk_{PER2} cycles wait state
111	7CLK	Seven Clk_{PER2} cycles wait state

27.11.3 CTRLB (SDRAM) – Control register B

The configuration options for this register depend on the chip select mode configuration. The register description below is valid for CS3 when the chip select mode is configured for SDRAM.

Bit	7	6	5	4	3	2	1	0
+0x01	SDINITDONE	–	–	–	–	SDREN	SDMODE[1:0]	
Read/Write	R/W	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7 – SDINITDONE: SDRAM Initialization Complete**
  This flag is set at the end of the SDRAM initialization sequence. The flag will remain set as long as the EBI is enabled and the Chip Select is configured for SDRAM.
- **Bit 6:3 – Reserved**
  These bits are unused and reserved for future use.
- **Bit 2 – SDSREN: SDRAM Self-refresh Enable**
  When this bit is written to one the EBI controller will send a Self-refresh command to the SDRAM. For leaving the self refresh mode, the bit must be written to zero.
- **Bit 1:0 SDMODE[1:0]: SDRAM Mode**
  These bits select the mode when accessing SDRAM according to Table 27-24 on page 337.

Table 27-24. SDRAM mode.

SDMODE[1:0]	Group configuration	Description
00	NORMAL	Normal mode - access to the SDRAM is decoded normally
01	LOAD	Load Mode - the EBI issues a Load Mode Register command when the SDRAM is accessed
10	–	Reserved
11	–	Reserved

27.11.4 BASEADDR – Base Address register

Bit	7	6	5	4	3	2	1	0
+0x02	BASEADDR[15:12]				–	–	–	–
Read/Write	R/W	R/W	R/W	R/W	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
+0x03	BASEADDR[23:16]							
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 15:4 – BASEADDR[23:12]: Chip Select Base Address**
  The base address is the lowest address in the address space enabled by a chip select. Together with the Chip Select Address Size (ASIZE) setting in “CTRLA - Chip Select Control Register A”, this gives the address space for the Chip Select.
- **Bit 3:0 – Reserved**
  These bits are unused and reserved for future use.

27.12 Register summary – EBI

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	CTRL	SDDATAW[1:0]		LPCMODE[1:0]		SRMODE[1:0]		IFMODE[1:0]		329
+0x01	SDRAMCTRLA	–	–	–	–	SDCAS	SDROW	SDCOL[1:0]		330
+0x02	Reserved	–	–	–	–	–	–	–	–	
+0x03	Reserved	–	–	–	–	–	–	–	–	
+0x04	REFRESHL	SDRAM Refresh Period Low Byte								331
+0x05	REFRESHH	–	–	–	–	–	–	SDRAM Refresh Period High		331
+0x06	INITDLYL	SDRAM Initialization Time Low Byte								331
+0x07	INITDLYH	–	–	SDRAM Initialization Time High Byte						331
+0x08	SDRAMCTRLB	MRDLY[1:0]		ROWCYCDLY[[2:0]		RPDLY[2:0]				332
+0x09	SDRAMCTRLC	WRDLY[1:0]		ESRDLY[2:0]		ROWCOLDLY[2:0]				333
+0x0A	Reserved	–	–	–	–	–	–	–	–	
+0x0B	Reserved	–	–	–	–	–	–	–	–	
+0x0C	Reserved	–	–	–	–	–	–	–	–	
+0x0D	Reserved	–	–	–	–	–	–	–	–	
+0x0E	Reserved	–	–	–	–	–	–	–	–	
+0x0F	Reserved	–	–	–	–	–	–	–	–	
+0x10	CS0	Chip Select 0 Offset Address								
+0x14	CS1	Chip Select 1 Offset Address								
+0x18	CS2	Chip Select 2 Offset Address								
+0x1C	CS3	Chip Select 3 Offset Address								

27.13 Register summary – EBI chip select

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	CTRLA		–	ASIZE[4:0]					MODE[1:0]		329
+0x01	CTRLB	(SRAM)	–	–	–	–	–	SRWS[2:0]			330
		(SDRAM)	SDINITDONE	–	–	–	–	SDSREN	SDMODE[1:0]		
+0x02	BASEADDRL		Chip Select Base Address Low Byte				–	–	–	–	337
+0x03	BASEADDRH		Chip Select Base Address High Byte								337

28. ADC – Analog-to-Digital Converter

28.1 Features

- 12-bit resolution
- Up to two million samples per second
  - Two inputs can be sampled simultaneously using ADC and 1x gain stage
  - Four inputs can be sampled within 1.5µs
  - Down to 2.5µs conversion time with 8-bit resolution
  - Down to 3.5µs conversion time with 12-bit resolution
- Differential and single-ended input
  - Up to 16 single-ended inputs
  - 16x4 differential inputs without gain
  - 8x4 differential input with gain
- Built-in differential gain stage
  - 1/2x, 1x, 2x, 4x, 8x, 16x, 32x, and 64x gain options
- Single, continuous and scan conversion options
- Four internal inputs
  - Internal temperature sensor
  - DAC output
  - V_{CC} voltage divided by 10
  - 1.1V bandgap voltage
- Four conversion channels with individual input control and result registers
  - Enable four parallel configurations and results
- Internal and external reference options
- Compare function for accurate monitoring of user defined thresholds
- Optional event triggered conversion for accurate timing
- Optional DMA transfer of conversion results
- Optional interrupt/event on compare result

28.2 Overview

The ADC converts analog signals to digital values. The ADC has 12-bit resolution and is capable of converting up to two million samples per second (MSPS). The input selection is flexible, and both single-ended and differential measurements can be done. For differential measurements, an optional gain stage is available to increase the dynamic range. In addition, several internal signal inputs are available. The ADC can provide both signed and unsigned results.

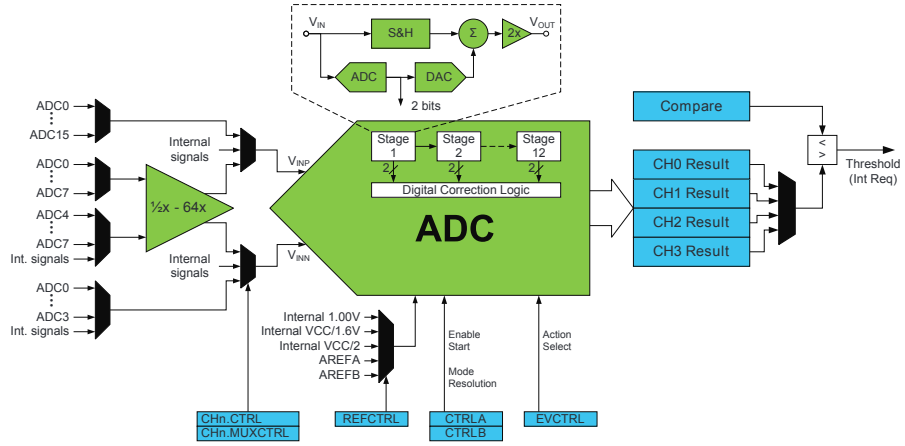
This is a pipelined ADC that consists of several consecutive stages. The pipelined design allows a high sample rate at a low system clock frequency. It also means that a new input can be sampled and a new ADC conversion started while other ADC conversions are still ongoing. This removes dependencies between sample rate and propagation delay.

The ADC has four conversion channels (0-3) with individual input selection, result registers, and conversion start control. The ADC can then keep and use four parallel configurations and results, and this will ease use for applications with high data throughput or for multiple modules using the ADC independently. It is possible to use DMA to move ADC results directly to memory or peripherals when conversions are done.

Both internal and external reference voltages can be used. An integrated temperature sensor is available for use with the ADC. The output from the DAC, $V_{CC}/10$ and the bandgap voltage can also be measured by the ADC.

The ADC has a compare function for accurate monitoring of user defined thresholds with minimum software intervention required.

Figure 28-1. ADC overview.



28.3 Input Sources

Input sources are the voltage inputs that the ADC can measure and convert. Four types of measurements can be selected:

- Differential input
- Differential input with gain
- Single-ended input
- Internal input

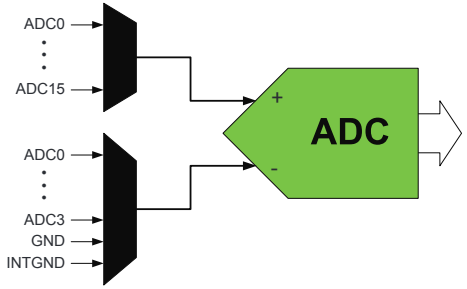
The input pins are used for single-ended and differential input, while the internal inputs are directly available inside the device. In devices with two ADCs, PORTA pins can be input to ADCA and PORTB pins can be input to ADCB. For AVR XMEGA devices with only one ADC, input pins may be available for ADCA on both PORTA and PORTB.

The ADC is differential, and so for single-ended measurements the negative input is connected to a fixed internal value. The four types of measurements and their corresponding input options are shown in Figure 28-2 on page 341 to Figure 28-6 on page 343.

28.3.1 Differential Input

When differential input is enabled, all input pins can be selected as positive input, and input pins 0 to 3 can be selected as negative input. The ADC must be in signed mode when differential input is used.

Figure 28-2. Differential measurement without gain.

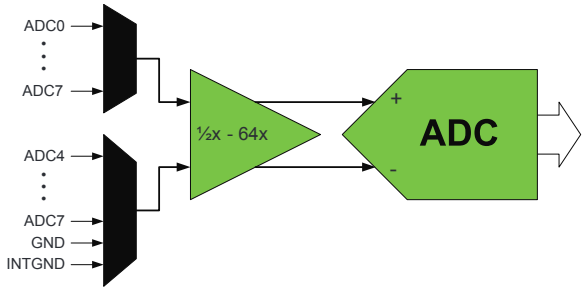


28.3.2 Differential Input with Gain

When differential input with gain is enabled, all input pins can be selected as positive input, and input pins 4 to 7 can be selected as negative input. When the gain stage is used, the differential input is first sampled and amplified by the gain stage before the result is fed into the ADC. The ADC must be in signed mode when differential input with gain is used.

The gain is selectable to 1/2x, 1x, 2x, 4x, 8x, 16x, 32x, and 64x gain.

Figure 28-3. Differential measurement with gain.

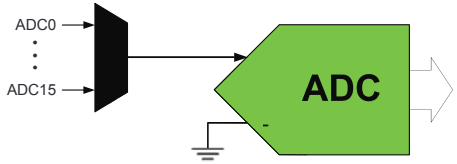


28.3.3 Single-ended Input

For single-ended measurements, all input pins can be used as inputs. Single-ended measurements can be done in both signed and unsigned mode.

The negative input is connected to internal ground in signed mode.

Figure 28-4. Single-ended measurement in signed mode.

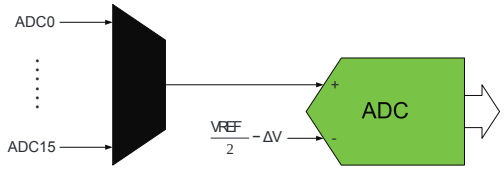


In unsigned mode, the negative input is connected to half of the voltage reference (VREF) voltage minus a fixed offset. The nominal value for the offset is:

$$\Delta V = VREF \cdot 0.05$$

Since the ADC is differential, the input range is VREF to zero for the positive single-ended input. The offset enables the ADC to measure zero crossing in unsigned mode, and allows for calibration of any positive offset when the internal ground in the device is higher than the external ground. See Figure 28-11 on page 345 for details.

Figure 28-5. Single-ended measurement in unsigned mode.



28.3.4 Internal Inputs

These internal signals can be measured or used by the ADC.

- Temperature sensor
- Bandgap voltage
- V_{CC} scaled
- DAC output
- Pad and Internal Ground

The temperature sensor gives an output voltage that increases linearly with the internal temperature of the device. One or more calibration points are needed to compute the temperature from a measurement of the temperature sensor. The temperature sensor is calibrated at one point in production test, and the result is stored to TEMPESENSE0 and TEMPSENSE1 in the production signature row. For more calibration condition details, refer to the device datasheet.

The bandgap voltage is an accurate internal voltage reference.

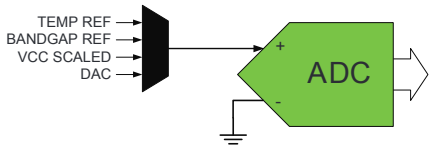
V_{CC} can be measured directly by scaling it down by a factor of 10 before the ADC input. Thus, a V_{CC} of 1.8V will be measured as 0.18V, and V_{CC} of 3.6V will be measured as 0.36V. This enables easy measurement of the V_{CC} voltage.

The internal signals need to be enabled before they can be measured. Refer to their manual sections for Bandgap and DAC for details of how to enable these. The sample rate for the internal signals is lower than that of the ADC. Refer to the ADC characteristics in the device datasheets for details.

For differential measurement Pad Ground (Gnd) and Internal Gnd can be selected as negative input. Pad Gnd is the gnd level on the pin and identical or very close to the external gnd. Internal Gnd is the internal device gnd level.

Internal Gnd is used as the negative input when other internal signals are measured in single-ended signed mode.

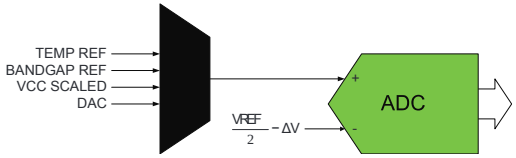
Figure 28-6. Internal measurements in single-ended signed mode.



To measure the internal signals in unsigned mode, the negative input is connected to a fixed value given by the formula below, which is half of the voltage reference (VREF) minus a fixed offset, as it is for single-ended unsigned input. Refer to Figure 28-11 on page 345 for details.

VINN = VREF/2 - ΔV

Figure 28-7. Internal measurements in unsigned mode.



28.4 ADC Channels

To facilitate the maximum utilization of the ADC, it has four separate pairs of MUX control registers with corresponding result registers. Each pair forms an ADC channel. See Figure 28-1 on page 340. The ADC can then keep and use four parallel configurations of input sources and triggers. Each channel has dedicated result register, events and interrupts, and DMA triggers.

As an example of the ADC channel usage, one channel can be setup for single-ended measurements triggered by an event channel, the second channel can measure a differential input using a different event, and the two last channels can measure two other input sources started by the application software.

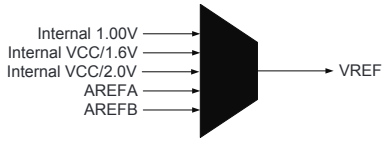
All the ADC channels use the same ADC pipeline for the conversions, and the pipeline enables a new conversion to be started for each ADC clock cycle. This means that multiple ADC measurements from different channels can be converted simultaneously and independently. The channels' result registers are individually updated and are unaffected by conversions on other channels. This can help reduce software complexity by allowing different software modules to start conversions and read conversion results fully independently of each other.

28.5 Voltage Reference Selection

The following voltages can be used as the reference voltage (VREF) for the ADC:

- Accurate internal 1.00V voltage generated from the bandgap
- Internal V_{CC}/1.6V voltage
- Internal V_{CC}/2V voltage
- External voltage applied to AREF pin on PORTA
- External voltage applied to AREF pin on PORTB

Figure 28-8. ADC voltage reference selection



28.6 Conversion Result

The result of the analog-to-digital conversion is written to the corresponding channel result registers. The ADC is either in signed or unsigned mode. This setting is global for the ADC and all ADC channels.

In signed mode, negative and positive results are generated. Signed mode must be used when any of the ADC channels are set up for differential measurements. In unsigned mode, only single-ended or internal signals can be measured. With 12-bit resolution, the TOP value of a signed result is 2047, and the results will be in the range -2048 to +2047 (0xF800 - 0x07FF).

The ADC transfer function can be written as:

$$RES = \frac{VINP - VINN}{VREF} \cdot GAIN \cdot TOP + 1$$

VINP and VINN are the positive and negative inputs to the ADC.

For differential measurements, GAIN is 1/2 to 64. For single-ended and internal measurements, GAIN is always 1 and VINP is the internal ground.

In unsigned mode, only positive results are generated. The TOP value of an unsigned result is 4095, and the results will be in the range 0 to +4095 (0x0 - 0x0FFF).

The ADC transfer functions can be written as:

$$RES = \frac{VINP - (-V)}{VREF} \cdot TOP + 1$$

VINP is the single-ended or internal input.

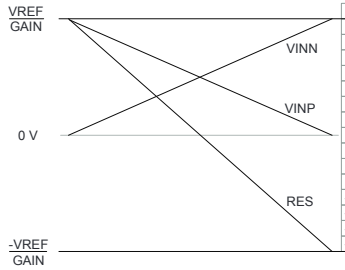
The ADC can be configured to generate either an 8-bit or a 12-bit result. A result with lower resolution will be available faster. See the "ADC Clock and Conversion Timing" on page 346 for a description on the propagation delay.

The result registers are 16 bits wide, and data are stored as right adjusted 16-bit values. Right adjusted means that the eight least-significant bits (lsb) are found in the low byte. A 12-bit result can be represented either left or right adjusted. Left adjusted means that the eight most-significant bits (msb) are found in the high byte.

When the ADC is in signed mode, the msb represents the sign bit. In 12-bit right adjusted mode, the sign bit (bit 11) is padded to bits 12-15 to create a signed 16-bit number directly. In 8-bit mode, the sign bit (bit 7) is padded to the entire high byte.

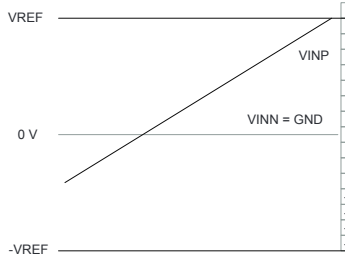
Figure 28-9 on page 345 to Figure 28-11 on page 345 show the different input options, the signal input range, and the result representation with 12-bit right adjusted mode.

Figure 28-9. Signed differential input (with gain), input range, and result representation.



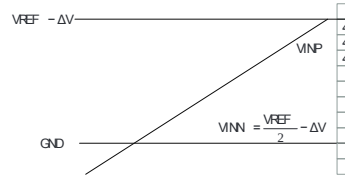
Dec	Hex	Binary	16-bit result register
2047	7FF	0111 1111 1111	0000 0111 1111 1111
2046	7FE	0111 1111 1110	0000 0111 1111 1110
2045	7FD	0111 1111 1101	0000 0111 1111 1101
...	...	...	...
3	3	0000 0000 0011	0000 0000 0000 0011
2	2	0000 0000 0010	0000 0000 0000 0010
1	1	0000 0000 0001	0000 0000 0000 0001
0	0	0000 0000 0000	0000 0000 0000 0000
-1	FFF	1111 1111 1111	1111 1111 1111 1111
-2	FFE	1111 1111 1110	1111 1111 1111 1110
...	...	...	...
-2045	803	1000 0000 0011	1111 1000 0000 0011
-2046	802	1000 0000 0010	1111 1000 0000 0010
-2047	801	1000 0000 0001	1111 1000 0000 0001
-2048	800	1000 0000 0000	1111 1000 0000 0000

Figure 28-10.Signed single-ended and internal input, input range, and result representation.



Dec	Hex	Binary	16-bit result register
2047	7FF	0111 1111 1111	0000 0111 1111 1111
2046	7FE	0111 1111 1110	0000 0111 1111 1110
2045	7FD	0111 1111 1101	0000 0111 1111 1101
...	...	...	...
3	3	0000 0000 0011	0000 0000 0000 0011
2	2	0000 0000 0010	0000 0000 0000 0010
1	1	0000 0000 0001	0000 0000 0000 0001
0	0	0000 0000 0000	0000 0000 0000 0000
-1	FFF	1111 1111 1111	1111 1111 1111 1111
-2	FFE	1111 1111 1110	1111 1111 1111 1110
...	...	...	...
-2045	803	1000 0000 0011	1111 1000 0000 0011
-2046	802	1000 0000 0010	1111 1000 0000 0010
-2047	801	1000 0000 0001	1111 1000 0000 0001
-2048	800	1000 0000 0000	1111 1000 0000 0000

Figure 28-11.Unsigned single-ended and internal input, input range, and result representation.



Dec	Hex	Binary	16-bit result register
4095	FFF	1111 1111 1111	0000 1111 1111 1111
4094	FFE	1111 1111 1110	0000 1111 1111 1110
4093	FFD	1111 1111 1101	0000 1111 1111 1101
...	...	...	...
203	0CB	0000 1100 1011	0000 0000 1100 1011
202	0CA	0000 1100 1010	0000 0000 1100 1010
201	0C9	0000 1100 1001	0000 0000 1100 1001
200	0C8	0000 1100 1000	0000 0000 1100 1000
...			
0	0	0000 0000 0000	0000 0000 0000 0000

28.7 Compare Function

The ADC has a built-in 12-bit compare function. The ADC compare register can hold a 12-bit value that represents a threshold voltage. Each ADC channel can be configured to automatically compare its result with this compare value to give an interrupt or event only when the result is above or below the threshold.

All four ADC channels share the same compare register.

28.8 Starting a Conversion

Before a conversion is started, the input source must be selected for one or more ADC channels. An ADC conversion for a channel can be started either by the application software writing to the start conversion bit for the channel or from any events in the event system. It is possible to write the start conversion bit for several channels at the same time, or use

one event to trigger conversions on several channels at the same time. This makes it possible to scan several or all channels from one event. The scan will start from the lowest channel number.

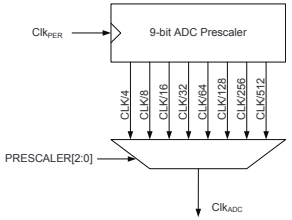
28.8.1 Input Source Scan

For ADC Channel 0 it is possible to select a range of consecutive input sources that is automatically scanned and measured when a conversion is started. This is done by setting the first (lowest) positive ADC channel input using the MUX control register, and a number of consecutive positive input sources. When a conversion is started, the first selected input source is measured and converted, then the positive input source selection is incremented after each conversion until it reaches the specified number of sources to scan.

28.9 ADC Clock and Conversion Timing

The ADC is clocked from the peripheral clock. The ADC can prescale the peripheral clock to provide an ADC Clock (clk_{ADC}) that matches the application requirements and is within the operating range of the ADC.

Figure 28-12.ADC prescaler.



The maximum ADC sample rate is given by the he ADC clock frequency (f_{ADC}). The ADC can sample a new measurement on every ADC clock cycle.

$$\text{Sample Rate} = f_{ADC}$$

The propagation delay of an ADC measurement is given by:

$$\text{Propagation Delay} = \frac{1 + \frac{RESOLUTION}{2} + GAIN}{f_{ADC}}$$

RESOLUTION is the resolution, 8 or 12 bits. The propagation delay will increase by one extra ADC clock cycle if the gain stage (GAIN) is used.

The propagation delay is longer than one ADC clock cycle, but the pipelined design means that the sample rate is limited not by the propagation delay, but by the ADC clock rate.

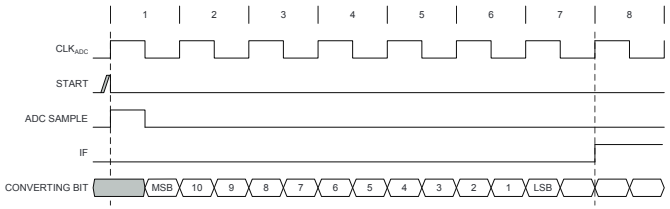
The most-significant bit (msb) of the result is converted first, and the rest of the bits are converted during the next three (for 8-bit results) or five (for 12-bit results) ADC clock cycles. Converting one bit takes a half ADC clock period. During the last cycle, the result is prepared before the interrupt flag is set and the result is available in the result register for readout.

28.9.1 Single Conversion without Gain

Figure 28-13 on page 347 shows the ADC timing for a single conversion without gain. The writing of the start conversion bit, or the event triggering the conversion (START), must occur at least one peripheral clock cycle before the ADC clock cycle on which the conversion starts (indicated with the grey slope of the START trigger).

The input source is sampled in the first half of the first cycle.

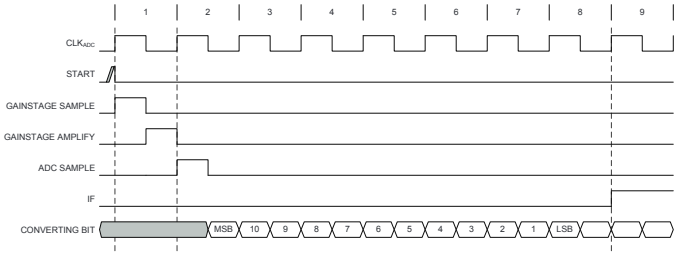
Figure 28-13.ADC timing for one single conversion without gain.



28.9.2 Single Conversion with Gain

Figure 28-14 on page 347 shows the ADC timing for one single conversion with gain. As seen in the "Overview" on page 339, the gain stage is placed prior to the actual ADC. The gain stage will sample and amplify the input source before the ADC samples it, and converts the amplified value. Compared to a single conversion without gain, this adds one ADC clock cycle (between START and ADC sample) for the gain stage sample and amplify. The sample time for the gain stage is one half ADC clock cycle.

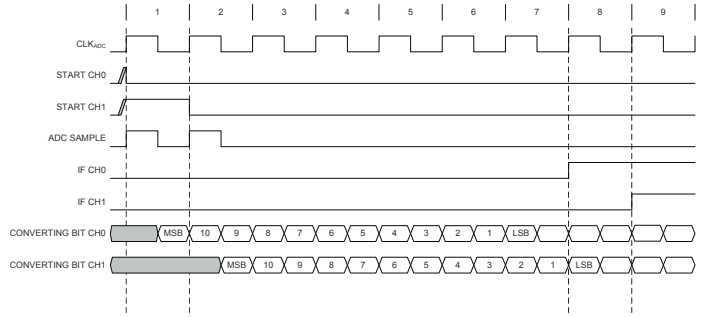
Figure 28-14.ADC timing for one single conversion with gain.



28.9.3 Single Conversions on Two ADC Channels

Figure 28-15 on page 348 shows the ADC timing for single conversions on two channels. The pipelined design enables the second conversion to start on the next ADC clock cycle after the first conversion has started. In this example, both conversions take place at the same time, but the conversion on ADC channel 1(CH1) does not start until the ADC samples and performs conversion on the msb on channel 0 (CH0).

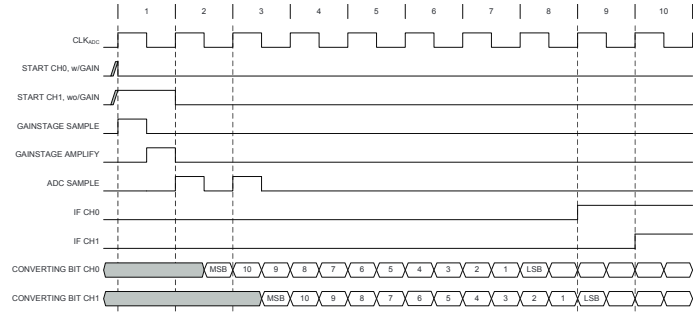
Figure 28-15.ADC timing for single conversions on two ADC channels.



28.9.4 Single Conversions on Two ADC Channels, CH0 with Gain

Figure 28-16 on page 348 shows the conversion timing for single conversions on two ADC channels where ADC channel 0 uses the gain stage. As the gain stage introduces one addition cycle for the gain sample and amplify, the sample for ADC channel 1 is also delayed one ADC clock cycle, until the ADC sample and msb conversion is done for ADC channel 0.

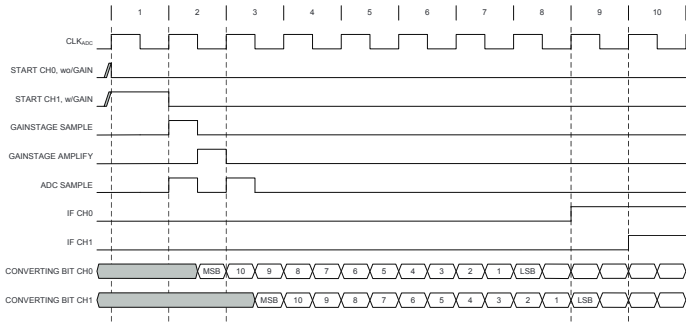
Figure 28-16.ADC timing for single conversion on two ADC channels, CH0 with gain.



28.9.5 Single Conversions on Two ADC Channels, CH1 with Gain

Figure 28-17 on page 349 shows the conversion timing for single conversions on two ADC channels where ADC channel 1 uses the gain stage.

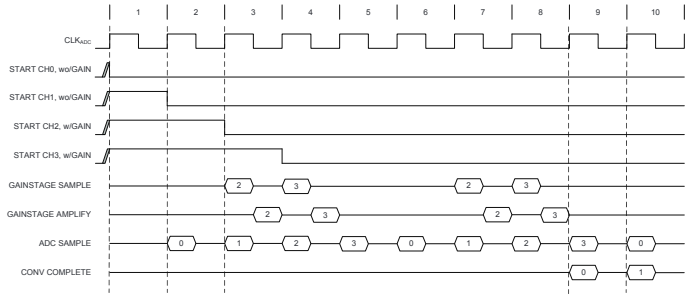
Figure 28-17.ADC timing for single conversion on two ADC channels, CH1 with gain.



28.9.6 Free Running Mode on Two ADC Channels with Gain

Figure 28-18 on page 349 shows the conversion timing for all four ADC channels in free running mode, CH0 and CH1 without gain and CH2 and CH3 with gain. When set up in free running mode, an ADC channel will continuously sample and do new conversions. In this example, all ADC channels are triggered at the same time, and each ADC channel samples and start converting as soon as the previous ADC channel is done with its sample and msb conversion. After four ADC clock cycles, all ADC channels have done the first sample and started the first conversion, and each ADC channels can then do the sample conversion start for their second conversion. After eight (for 12-bit mode) ADC clock cycles, the first conversion is done for ADC channel 0, and the results for the rest of the ADC channels are available in subsequent ADC clock cycles. After the next clock cycle (in cycle 10), the result from the second ADC channel is done and available, and so on. In this mode, up to eight conversions are ongoing at the same time.

Figure 28-18.ADC timing for free running mode.



28.10 ADC Input Model

The voltage input must charge the sample and hold (S/H) capacitor in the ADC in order to achieve maximum accuracy. Seen externally, the ADC input consists of an input resistance (R_{in} = $R_{channel}$ + R_{switch}) and the S/H capacitor (C_{sample}). Figure 28-19 on page 350 and Figure 28-20 on page 350 show the ADC input channels.

Figure 28-19.ADC input for single-ended measurements.

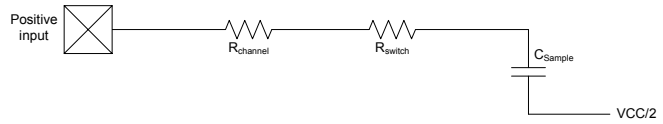
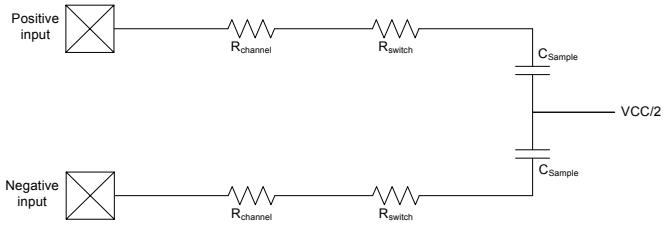


Figure 28-20.ADC input for differential measurements and differential measurements with gain.



In order to achieve *n* bits of accuracy, the source output resistance, R_{source}, must be less than the ADC input resistance on a pin:

$$R_{source} \leq \frac{T_s}{C_{sample} \cdot \ln(2^{n+1})} - R_{channel} - R_{switch}$$

where the ADC sample time, T_S is one-half the ADC clock cycle given by:

$$T_s \geq \frac{1}{2 \cdot f_{ADC}}$$

For details on $R_{channel}$, R_{switch}, and C_{sample}, refer to the ADC and ADC gain stage electrical characteristic in the device datasheet.

28.10.1 Gain Stage Impedance mode

To support applications with very high source output resistance, the gain stage has a high impedance mode. In this mode the charge on the S/H capacitor is kept after each sample, and the S/H capacitor can be fully charged by doing multiple samples on the same input channel. When low impedance mode is used, the S/H capacitor charge is flushed after each sample.

28.11 DMA Transfer

The DMA controller can be used to transfer ADC conversion results to memory or other peripherals. A new conversion result for any of the ADC channels can trigger a DMA transaction for one or several ADC channels. Refer to "DMAC - Direct Memory Access Controller" on page 53 for more details on DMA transfers.

28.12 Interrupts and Events

The ADC can generate interrupt requests and events. Each ADC channel has individual interrupt settings and interrupt vectors. Interrupt requests and events can be generated when an ADC conversion is complete or when an ADC measurement is above or below the ADC compare register value.

28.13 Calibration

The ADC has built-in linearity calibration. The value from the production test calibration must be loaded from the signature row and into the ADC calibration register from software to achieve specified accuracy. User calibration of the linearity is not needed, hence not possible. Offset and gain calibration must be done in software.

28.14 Channel Priority

Since the peripheral clock is faster than the ADC clock, it is possible to set the start conversion bit for several ADC channels within the same ADC clock period. Events may also trigger conversions on several ADC channels and give the same scenario. In this case, the ADC channel with the lowest number will be prioritized. This is shown the timing diagrams in “ADC Clock and Conversion Timing” on page 346.

28.15 Synchronous Sampling

The ADC can be configured to do synchronous sampling in three different ways.

1. Sample two input channels at the same time
2. Sample two ADCs at the same time
3. Sample on external trigger

28.15.1 Synchronous sampling of two ADC inputs

The ADC supports sampling of two input channels at the same time. This is achieved by setting up channel n to not use gain and channel n+1 to use 1x gain. The converted result from the channel using gain will be ready one ADC clock cycle after the other channel. See “Single Conversions on Two ADC Channels, CH1 with Gain” on page 348 for detailed timing diagram.

28.15.2 Synchronous sampling on event

Starting an ADC conversion can cause an unknown delay between the start trigger or event and the actual conversion start, since conversions of higher priority ADC channels may be pending, or since the peripheral clock is faster than the ADC clock. To start an ADC conversion immediately on an incoming event, it is possible to flush the ADC of all measurements, reset the ADC clock, and start the conversion at the next peripheral clock cycle (which then will also be the next ADC clock cycle). If this is done, all ongoing conversions in the ADC pipeline will be lost.

The ADC can be flushed from software, or an incoming event can do this automatically. When this function is used, the time between each conversion start trigger must be longer than the ADC propagation delay to ensure that one conversion is finished before the ADC pipeline is flushed and the next conversion is started.

It is also important to clear pending events or start ADC conversion commands before doing a flush. If not, pending conversions will start immediately after the flush.

28.15.3 Synchronous sampling of two ADCs

In devices with two ADC peripherals, it is possible to start two ADC samples synchronously in the two ADCs by using the same event channel to trigger both ADC.

28.16 Register Description – ADC

28.16.1 CTRLA – Control register A

Bit	7	6	5	4	3	2	1	0
+0x00	DMASEL[1:0]		CHSTART[3:0]				FLUSH	ENABLE
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 7:6 – DMASEL[1:0]: DMA Request Selection
To allow one DMA channel to serve more than one ADC channel, the DMA request from the channels can be combined into a common DMA request. See Table 28-1 for details.

Table 28-1. DMA request selection.

DMASEL[1:0]	Group configuration	Description
00	OFF	No combined DMA request
01	CH01	Common request for ADC channels 0 and 1
10	CH012	Common request for ADC channels 0, 1, and 2
11	CH0123	Common request for ADC channels 0, 1, 2, and 3

Bit 5:2 – CHSTART[3:0]: Channel Start Single Conversion
Setting any of these bits will start a conversion on the corresponding ADC channel. Setting several bits at the same time will start conversions on all selected ADC channels, starting with the channel with the lowest number. These bits are cleared by hardware when the conversion has started.

Bit 1 – FLUSH: Pipeline Flush:
Setting this bit will flush the ADC pipeline. When this is done, the ADC clock is restarted on the next peripheral clock edge, and all conversions in progress are aborted and lost.

After the flush and the ADC clock restart, the ADC will resume where it left off; i.e., if a channel sweep was in progress or any conversions were pending, these will enter the ADC pipeline and complete.

Bit 0 – ENABLE: Enable
Setting this bit enables the ADC.

28.16.2 CTRLB – ADC Control register B

Bit	7	6	5	4	3	2	1	0
+0x01	IMPMODE	CURRLIMIT[1:0]		CONVMODE	FREERUN	RESOLUTION[1:0]		–
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – IMPMODE: Gain Stage Impedance Mode
This bit controls the impedance mode of the gain stage.

See GAIN setting in ADC channel register description for more information (“CTRL – Channel Control register” on page 359).

Table 28-2. Gain stage impedance mode.

IMPMODE	Group configuration[1]	Description
0	HIGHIMP	For high-impedance sources; charge will remain on input
1	LOWIMP	For low impedance sources

Note: 1. This is either high or low impedance. While high impedance mode is only available for 1x, 2x, 4x, and 8x, for all other it will be forced to low impedance mode. See Table 28-10 on page 359.

Bit 6:5 – CURRLIMIT[1:0]: Current Limitation
These bits can be used to limit the current consumption of the ADC by reducing the maximum ADC sample rate. The available settings are shown in Table 28-3 on page 353. The indicated current limitations are nominal values. Refer to the device datasheet for actual current limitation for each setting.

Table 28-3. ADC current limitations.

CURRLIMIT[1:0]	Group configuration	Description
00	NO	No limit
01	LOW	Low current limit, max. sampling rate 1.5MSPS
10	MED	Medium current limit, max. sampling rate 1MSPS
11	HIGH	High current limit, max. sampling rate 0.5MSPS

Bit 4 – CONVMODE: Conversion Mode
This bit controls whether the ADC will work in signed or unsigned mode. By default, this bit is cleared and the ADC is configured for unsigned mode. When this bit is set, the ADC is configured for signed mode.

Bit 3 – FREERUN: Free Running Mode
When the bit is set to one, the ADC is in free running mode and the ADC channels defined in the EVCTRL register are swept repeatedly.

Bit 2:1 – RESOLUTION[1:0]: Conversion Result Resolution
These bits define whether the ADC completes the conversion at 12- or 8-bit result resolution. They also define whether the 12-bit result is left or right adjusted within the 16-bit result registers. See Table 28-4 on page 353 for possible settings.

Table 28-4. ADC conversion result resolution.

RESOLUTION[1:0]	Group configuration	Description
00	12BIT	12-bit result, right adjusted
01		Reserved
10	8BIT	8-bit result, right adjusted
11	LEFT12BIT	12-bit result, left adjusted

Bit 0 – Reserved
This bit is unused and reserved for future use. For compatibility with future devices, always write this bit to zero when this register is written.

28.16.3 REFCTRL – Reference Control register

Bit	7	6	5	4	3	2	1	0
+0x02	–	REFSEL[2:0]			–	–	BANDGAP	TEMPREF
Read/Write	R	R/W	R/W	R/W	R	R	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – Reserved
This bit is unused and reserved for future use. For compatibility with future devices, always write this bit to zero when this register is written.

Bits 6:4 – REFSEL[2:0]: Reference Selection
These bits selects the reference for the ADC according to Table 28-5 on page 354.

Table 28-5. ADC reference selection.

REFSEL[2:0]	Group configuration	Description
000	INT1V	10/11 of bandgap (1.0V)
001	INTVCC	V_{CC}/1.6
010	AREFA	External reference from AREF pin on PORT A
011	AREFB	External reference from AREF pin on PORT B
100	INTVCC2	V_{CC}/2
101 - 111		Reserved

Bit 3:2 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 1 – BANDGAP: Bandgap Enable
Setting this bit enables the bandgap for ADC measurement. Note that if any other functions are already using the bandgap, this bit does not need to be set when the internal 1.00V reference is used for another ADC, the DAC or if the brownout detector is enabled.

Bit 0 – TEMPREF: Temperature Reference Enable
Setting this bit enables the temperature sensor for ADC measurement.

28.16.4 EVCTRL – Event Control register

Bit	7	6	5	4	3	2	1	0
+0x03	SWEEP[1:0]		EVSEL[2:0]			EVACT[2:0]		
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 7:6 – SWEEP[1:0]: Channel Sweep
These bits control which ADC channels are included in a channel sweep triggered by the event system or when in free running mode. See Table 28-6 on page 355.

Table 28-6. ADC channel select.

SWEEP[1:0]	Group configuration	Active ADC channels for channel sweep
00	0	Only ADC channel 0
01	01	ADC channels 0 and 1
10	012	ADC channels 0, 1, and 2
11	0123	ADC channels 0, 1, 2, and 3

Bit 5:3 – EVSEL[2:0]: Event Channel Input Select
These bits select which event channel will trigger which ADC channel. Each setting defines a group of event channels, where the event channel with the lowest number will trigger ADC channel 0, the next event channel will trigger ADC channel 1, and so on. See Table 28-7 on page 355.

Table 28-7. ADC event channel select.

EVSEL[2:0]	Group configuration	Selected event lines
000	0123	Event channel 0, 1, 2, and 3 as selected inputs
001	1234	Event channel 1, 2, 3, and 4 as selected inputs
010	2345	Event channel 2, 3, 4, and 5 as selected inputs
011	3456	Event channel 3, 4, 5, and 6 as selected inputs
100	4567	Event channel 4, 5, 6, and 7 as selected inputs
101	567	Event channel 5, 6, and 7 as selected inputs
110	67	Event channel 6and7 as selected inputs
111	7	Event channel 7 as selected input

Bit 2:0 – EVACT[2:0]: Event Mode
These bits select and limit how many of the selected event input channel are used, and also further limit the ADC channels triggers. They also define more special event triggers as defined in Table 28-8 on page 355.

Table 28-8. ADC event mode select.

EVACT[2:0]	Group configuration	Selected input operation mode
000	NONE	No event inputs
001	CH0	Event channel with the lowest number defined by EVSEL triggers conversion on ADC channel 0
010	CH01	Event channels with the two lowest numbers defined by EVSEL trigger conversions on ADC channels 0 and 1, respectively
011	CH012	Event channels with the three lowest numbers defined by EVSEL trigger conversions on ADC channels 0, 1, and 2, respectively
100	CH0123	Event channels defined by EVSEL trigger conversion on ADC channels 0, 1, 2, and 3, respectively

EVACT[2:0]	Group configuration	Selected input operation mode
101	SWEEP	One sweep of all ADC channels defined by SWEEP on incoming event channel with the lowest number defined by EVSEL
110	SYNCSWEEP	One sweep of all active ADC channels defined by SWEEP on incoming event channel with the lowest number defined by EVSE. In addition the ADC is flushed and restarted for accurate timing
111		Reserved

28.16.5 PRESCALER – Clock Prescaler register

Bit	7	6	5	4	3	2	1	0
+0x04	–	–	–	–	–	PRESCALER[2:0]		
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:3 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 2:0 – PRESCALER[2:0]: Prescaler Configuration
These bits define the ADC clock relative to the peripheral clock according to Table 28-9 on page 356.

Table 28-9. ADC prescaler settings.

PRESCALER[2:0]	Group configuration	Peripheral clock division factor
000	DIV4	4
001	DIV8	8
010	DIV16	16
011	DIV32	32
100	DIV64	64
101	DIV128	128
110	DIV256	256
111	DIV512	512

28.16.6 INTFLAGS – Interrupt Flag register

Bit	7	6	5	4	3	2	1	0
+0x06	–	–	–	–	CH[3:0]IF			
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:4 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 3:0 – CH[3:0]IF: Interrupt Flags
These flags are set when the ADC conversion is complete for the corresponding ADC channel. If an ADC channel is configured for compare mode, the corresponding flag will be set if the compare condition is met. CHnIF is automatically cleared when the ADC channel n interrupt vector is executed. The flag can also be cleared by writing a one to its bit location.

28.16.7 TEMP – Temporary register

Bit	7	6	5	4	3	2	1	0
+0x07	TEMP[7:0]							
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 7:0 – TEMP[7:0]: Temporary bits
This register is used when reading 16-bit registers in the ADC controller. The high byte of the 16-bit register is stored here when the low byte is read by the CPU. This register can also be read and written from the user software.
For more details on 16-bit register access, refer to "The combined EIND + Z register." on page 12.

28.16.8 CALL – Calibration Value register

The CALL and CALH register pair hold the 12-bit calibration value. The ADC pipeline is calibrated during production programming, and the calibration value must be read from the signature row and written to the CAL register from software.

Bit	7	6	5	4	3	2	1	0
+0x0C	CAL[7:0]							
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 7:0 – CAL[7:0]: ADC Calibration value
These are the eight lsbs of the 12-bit CAL value.

28.16.9 CALH – Calibration Value register

Bit	7	6	5	4	3	2	1	0
+0x0D	–	–	–	–	CAL[11:8]			
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

Bit 3:0 – CAL[11:8]: Calibration value
These are the four msbs of the 12-bit CAL value.

28.16.10 CHnRESH – Channel n Result register High

The CHnRESL and CHnRESH register pair represents the 16-bit value, CHnRES. For details on reading 16-bit registers, refer to "The combined EIND + Z register." on page 12.

Bit	7	6	5	4	3	2	1	0
12-bit, left	CHRES[11:4]							
12-bit, right	–	–	–	–	CHRES[11:8]			
8-bit	–	–	–	–	–	–	–	–
Read/Write	R	R	R	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

28.16.10.1 12-bit Mode, Left Adjusted

Bit 7:0 – CHRES[11:4]: Channel Result high byte
These are the eight msbs of the 12-bit ADC result.

28.16.10.2 12-bit Mode, Right Adjusted

Bit 7:4 – Reserved
These bits will in practice be the extension of the sign bit, CHRES11, when the ADC works in differential mode, and set to zero when the ADC works in signed mode.

Bit 3:0 – CHRES[11:8]: Channel Result high byte
These are the four msbs of the 12-bit ADC result.

28.16.10.3 8-bit Mode

Bit 7:0 – Reserved
These bits will in practice be the extension of the sign bit, CHRES7, when the ADC works in signed mode, and set to zero when the ADC works in single-ended mode.

28.16.11 CHnRESL – Channel n Result register Low

Bit	7	6	5	4	3	2	1	0
12-/8-bit, right	CHRES[7:0]							
12-bit, left	CHRES[3:0]				–	–	–	–
Read/Write	R	R	R	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

28.16.11.1 12-/8-bit Mode

Bit 7:0 – CHRES[7:0]: Channel Result low byte
These are the eight lsbs of the ADC result.

28.16.11.2 12-bit Mode, Left Adjusted

Bit 7:4 – CHRES[3:0]: Channel Result low byte
These are the four lsbs of the 12-bit ADC result.

Bit 3:0 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

28.16.12 CMPH – Compare register High

The CMPH and CMPL register pair represents the 16-bit value, CMP. For details on reading and writing 16-bit registers, refer to "The combined EIND + Z register." on page 12.

Bit	7	6	5	4	3	2	1	0
+0x19	CMP[15:0]							
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 7:0 – CMP[15:0]: Compare Value high
These are the eight msbs of the 16-bit ADC compare value. In signed mode, the number representation is 2's complement, and the msb is the sign bit.

28.16.13 CMPL – Compare register Low

Bit	7	6	5	4	3	2	1	0
+0x18	CMP[7:0]							
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 7:0 – CMP[7:0]: Compare Value Low
These are the eight lsbs of the 16-bit ADC compare value. In signed mode, the number representation is 2's complement.

28.17 Register Description – ADC Channel

28.17.1 CTRL – Channel Control register

Bit	7	6	5	4	3	2	1	0
+0x00	START	–	–	GAIN[2:0]			INPUTMODE[1:0]	
Read/Write	R/W	R	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – START: START Conversion on Channel
Setting this bit will start a conversion on the channel. The bit is cleared by hardware when the conversion has started. Setting this bit when it already is set will have no effect. Writing or reading this bit is equivalent to writing the CH[3:0]START bits in "CTRLA – Control register A" on page 352.

Bit 6:5 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 4:2 – GAIN[2:0]: Gain Factor
These bits define the gain factor for the ADC gain stage.

See Table 28-10 on page 359. Gain is valid only with certain MUX settings. See "MUXCTRL – ADC Channel MUX Control registers" on page 360.

Table 28-10. ADC gain factor.

GAIN[2:0]	Group configuration	Gain factor
000	1X	1x
001	2X	2x
010	4X	4x
011	8X	8x
100	16X	16x
101	32X	32x
110	64X	64x
111	DIV2	½x

Bit 1:0 – INPUTMODE[1:0]: Channel Input Mode
These bits define the channel mode. Changing input mode will corrupt any data in the pipeline.

Table 28-11. Channel input modes, CONVMODE=0 (unsigned mode).

INPUTMODE[1:0]	Group configuration	Description
00	INTERNAL	Internal positive input signal
01	SINGLEENDED	Single-ended positive input signal
10		Reserved
11		Reserved

Table 28-12. Channel input modes, CONVMODE=1 (signed mode).

INPUTMODE[1:0]	Group configuration	Description
00	INTERNAL	Internal positive input signal
01	SINGLEENDED	Single-ended positive input signal
10	DIFF	Differential input signal
11	DIFFWGAIN	Differential input signal with gain

28.17.2 MUXCTRL – ADC Channel MUX Control registers

The MUXCTRL register defines the input source for the channel.

Bit	7	6	5	4	3	2	1	0
+0x01	–	MUXPOS[3:0]				MUXNEG[2:0]		
Read/Write	R	R/W	R/W	R/W	R/W	R	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – Reserved
This bit is unused and reserved for future use. For compatibility with future devices, always write this bit to zero when this register is written.

Bit 6:3 – MUXPOS[3:0]: MUX Selection on Positive ADC Input
These bits define the MUX selection for the positive ADC input. Table 28-13 on page 360 and Table 28-14 on page 361 show the possible input selection for the different input modes.

Table 28-13. Channel input modes, CONVMODE=1 (unsigned mode).

MUXPOS[3:0]	Group configuration	Description
0000	TEMP	Temperature reference
0001	BANDGAP	Bandgap voltage
0010	SCALEDVCC	1/10 scaled V_{CC}
0011	DAC	DAC output
0100-1111		Reserved

Table 28-14. ADC MUXPOS configuration when INPUTMODE[1:0] = 01 (single-ended) or INPUTMODE[1:0] = 10 (differential) is used.

MUXPOS[3:0]	Group configuration	Description
0000	PIN0	ADC0 pin
0001	PIN1	ADC1 pin
0010	PIN2	ADC2 pin
0011	PIN3	ADC3 pin
0100	PIN4	ADC4 pin
0101	PIN5	ADC5 pin
0110	PIN6	ADC6 pin
0111	PIN7	ADC7 pin
1000	PIN8	ADC8 pin
1001	PIN9	ADC9 pin
1010	PIN10	ADC10 pin
1011	PIN11	ADC11 pin
1100	PIN12	ADC12 pin
1101	PIN13	ADC13 pin
1110	PIN14	ADC14 pin
1111	PIN15	ADC15 pin

Table 28-15. ADC MUXPOS configuration when INPUTMODE[1:0] = 11 (differential with gain) is used.

MUXPOS[3:0]	Group configuration	Description
0000	PIN0	ADC0 pin
0001	PIN1	ADC1 pin
0010	PIN2	ADC2 pin
0011	PIN3	ADC3 pin
0100	PIN4	ADC4 pin
0101	PIN5	ADC5 pin
0110	PIN6	ADC6 pin
0111	PIN7	ADC7 pin
1XXX		Reserved

Depending on the device pin count and feature configuration, the actual number of analog input pins may be less than 16. Refer to the device datasheet and pin-out description for details.

Bit 2:0 – MUXNEG[2:0]: MUX Selection on Negative ADC Input
These bits define the MUX selection for the negative ADC input when differential measurements are done. For internal or single-ended measurements, these bits are not used.

Table 28-16 on page 362 and Table 28-17 on page 362 show the possible input sections.

Table 28-16. ADC MUXNEG configuration, INPUTMODE[1:0] = 10, differential without gain.

MUXNEG[2:0]	Group configuration	Analog Input
000	PIN0	ADC0 pin
001	PIN1	ADC1 pin
010	PIN2	ADC2 pin
011	PIN3	ADC3 pin
100	-	Reserved
101	GND	PAD ground
110	-	Reserved
111	INTGND	Internal ground

Table 28-17. ADC MUXNEG configuration, INPUTMODE[1:0] = 11, differential with gain.

MUXNEG[2:0]	Group configuration	Analog Input
000	PIN4	ADC4 pin
001	PIN5	ADC5 pin
010	PIN6	ADC6 pin
011	PIN7	ADC7 pin
100	INTGND	Internal ground
101	-	Reserved
110	-	Reserved
111	GND	PAD ground

28.17.3 INTCTRL – Channel Interrupt Control registers

Bit	7	6	5	4	3	2	1	0
+0x02	–	–	–	–	INTMODE[1:0]		INTLVL[1:0]	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bits 7:4 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 3:2 – INTMODE: Interrupt Mode
These bits select the interrupt mode for the channel according to Table 28-18.

Table 28-18. ADC channel select.

INTMODE[1:0]	Group configuration	Interrupt mode
00	COMPLETE	Conversion complete
01	BELOW	Compare result below threshold
10		Reserved
11	ABOVE	Compare result above threshold

Bits 1:0 – INTLVL[1:0]: Interrupt Priority Level and Enable
These bits enable the ADC channel interrupt and select the interrupt level, as described in "Interrupts and Programmable Multilevel Interrupt Controller" on page 131. The enabled interrupt will be triggered for conditions when the IF bit in the INTFLAGS register is set.

28.17.4 INTFLAGS – ADC Channel Interrupt Flag registers

Bit	7	6	5	4	3	2	1	0
+0x03	–	–	–	–	–	–	–	IF
Read/Write	R	R	R	R	R	R	R	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:1 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 0 – IF: Channel Interrupt Flag
The interrupt flag is set when the ADC conversion is complete. If the channel is configured for compare mode, the flag will be set if the compare condition is met. IF is automatically cleared when the ADC channel interrupt vector is executed. The bit can also be cleared by writing a one to the bit location.

28.17.5 RESH – Channel n Result register High

For all result registers and with any ADC result resolution, a signed number is represented in 2's complement form, and the msb represents the sign bit.

The RESL and RESH register pair represents the 16-bit value, ADCRESULT. Reading and writing 16-bit values require special attention. Refer to "The combined EIND + Z register." on page 12 for details.

	Bit	7	6	5	4	3	2	1	0
12-bit, left.	+0x05	RES[11:4]							
12-bit, right		–	–	–	–	RES[11:8]			
8-bit		–	–	–	–	–	–	–	–
	Read/Write	R	R	R	R	R	R	R	R
	Initial Value	0	0	0	0	0	0	0	0

28.17.5.1 12-bit Mode, Left Adjusted

Bit 7:0 – RES[11:4]: Channel Result High
These are the eight msbs of the 12-bit ADC result.

28.17.5.2 12-bit Mode, Right Adjusted

Bit 7:4 – Reserved
These bits will in practice be the extension of the sign bit, CHRES11, when the ADC works in differential mode, and set to zero when the ADC works in signed mode.

Bits 3:0 – RES[11:8]: Channel Result High byte
These are the four msbs of the 12-bit ADC result.

28.17.5.3 8-bit Mode

Bit 7:0 – Reserved
These bits will in practice be the extension of the sign bit, CHRES7, when the ADC works in signed mode, and set to zero when the ADC works in single-ended mode.

28.17.6 RESL – Channel n Result register Low

	Bit	7	6	5	4	3	2	1	0
12-/8-bit, right	+0x04	RES[7:0]							
12-bit, left.		RES[3:0]				–	–	–	–
	Read/Write	R	R	R	R	R	R	R	R
	Initial Value	0	0	0	0	0	0	0	0

28.17.6.1 12-/8-bit Mode

Bit 7:0 – RES[7:0]: Channel Result Low
These are the eight lsbs of the ADC result.

28.17.6.2 12-bit Mode, Left Adjusted

Bit 7:4 – RES[3:0]: Channel Result Low
These are the four lsbs of the 12-bit ADC result.

Bit 3:0 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

28.17.7 SCAN – Channel Scan register

Scan is enabled when COUNT is set differently than 0. This register is available only for ADC channel 0.

Bit	7	6	5	4	3	2	1	0
+0x06	OFFSET[3:0]				COUNT[3:0]			
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 7:4 – OFFSET[3:0]: Positive MUX Setting Offset
The channel scan is enabled when COUNT != 0 and this register contains the offset for the next input source to be converted on ADC channel 0 (CH0). The actual MUX setting for positive input equals MUXPOS + OFFSET. The value is incremented after each conversion until it reaches the maximum value given by COUNT. When OFFSET is equal to COUNT, OFFSET will be cleared on the next conversion.

Bit 3:0 – COUNT[3:0]: Number of Input Channels Included in Scan
This register gives the number of input sources included in the channel scan. The number of input sources included is COUNT + 1. The input channels included are the range from MUXPOS to MUXPOS + COUNT.

28.18 Register summary – ADC

This is the register summary when the ADC is configured to give standard 12-bit results. The register summaries for 8-bit and 12-bit left adjusted will be similar, but with some changes in the result registers, CHnRESH and CHnRESL.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	CTRLA	DMASEL[1:0]		CH[3:0]START				FLUSH	ENABLE	352
+0x01	CTRLB	IMPMODE	CURRLIMIT[1:0]		CONVMODE	FREERUN	RESOLUTION[1:0]		–	352
+0x02	REFCTRL	–	REFSEL[2:0]			–	–	BANDGAP	TEMPREF	354
+0x03	EVCTRL	SWEEP[1:0]		EVSEL[2:0]			EVACT[2:0]			354
+0x04	PRESCALER	–	–	–	–	–	PRESCALER[2:0]			356
+0x05	Reserved	–	–	–	–	–	–	–	–	
+0x06	INTFLAGS	–	–	–	–	CH[3:0]IF				356
+0x07	TEMP	TEMP[7:0]								357
+0x08	Reserved	–	–	–	–	–	–	–	–	
+0x09	Reserved	–	–	–	–	–	–	–	–	
+0x0A	Reserved	–	–	–	–	–	–	–	–	
+0x0B	Reserved	–	–	–	–	–	–	–	–	
+0x0C	CALL	CAL[7:0]								357
+0x0D	CALH	–	–	–	–	CAL[11:8]				357
+0x0E	Reserved	–	–	–	–	–	–	–	–	
+0x0F	Reserved	–	–	–	–	–	–	–	–	
+0x10	CH0RESL	CH0RES[7:0]								358
+0x11	CH0RESH	CH0RES[15:8]								357
+0x12	CH1RESL	CH1RES[7:0]								358
+0x13	CH1RESH	CH1RES[15:8]								357
+0x14	CH2RESL	CH2RES[7:0]								358
+0x15	CH2RESH	CH2RES[15:8]								357
+0x16	CH3RESL	CH3RES[7:0]								358
+0x17	CH3RESH	CH3RES[15:8]								357
+0x18	CMPL	CMP[7:0]								359
+0x19	CMPH	CMP[15:8]								358
+0x1A	Reserved	–	–	–	–	–	–	–	–	
+0x1B	Reserved	–	–	–	–	–	–	–	–	
+0x1C	Reserved	–	–	–	–	–	–	–	–	
+0x1D	Reserved	–	–	–	–	–	–	–	–	
+0x1E	Reserved	–	–	–	–	–	–	–	–	
+0x1F	Reserved	–	–	–	–	–	–	–	–	

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x20	CH0 Offset	–	–	–	–	–	–	–	–	
+0x28	CH1 Offset	–	–	–	–	–	–	–	–	
+0x30	CH2 Offset	–	–	–	–	–	–	–	–	
+0x38	CH3 Offset	–	–	–	–	–	–	–	–	

28.19 Register summary – ADC channel

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	CTRL	START	–	–	GAIN[2:0]			INPUTMODE[1:0]		359
+0x01	MUXCTRL	–	MUXPOS[3:0]				MUXNEG[2:0]			360
+0x02	INTCTRL	–	–	–	–	INTMODE[1:0]		INTLVL[1:0]		362
+0x03	INTFLAGS	–	–	–	–	–	–	–	IF	363
+0x04	RESL	RES[7:0]								364
+0x05	RESH	RES[15:8]								363
+0x06	SCAN	OFFSET				COUNT				363
+0x07	Reserved	–	–	–	–	–	–	–	–	

28.20 Interrupt vector summary

Table 28-19. Analog-to-digital converter interrupt vectors and their word offset address.

Offset	Source	Interrupt Description
0x00	CH0	Analog-to-digital converter channel 0 interrupt vector
0x02	CH1	Analog-to-digital converter channel 1 interrupt vector
0x04	CH2	Analog-to-digital converter channel 2 interrupt vector
0x06	CH3	Analog-to-digital converter channel 3 interrupt vector

29. DAC – Digital to Analog Converter

29.1 Features

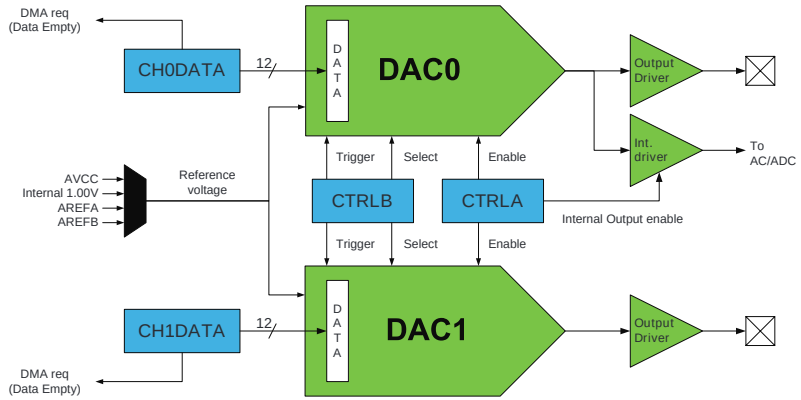
- 12-bit resolution
- Two independent, continuous-drive output channels
- Up to one million samples per second conversion rate per DAC channel
- Built-in calibration that removes:
  - Offset error
  - Gain error
- Multiple conversion trigger sources
  - On new available data
  - Events from the event system
- High drive capabilities and support for
  - Resistive loads
  - Capacitive loads
  - Combined resistive and capacitive loads
- Internal and external reference options
- DAC output available as input to analog comparator and ADC
- Low-power mode, with reduced drive strength
- Optional DMA transfer of data

29.2 Overview

The digital-to-analog converter (DAC) converts digital values to voltages. The DAC has two channels, each with12-bit resolution, and is capable of converting up to one million samples per second (MSPS) on each channel. The built-in calibration system can remove offset and gain error when loaded with calibration values from software.

Figure 29-1 illustrates the basic functionality of the DAC. Not all functions are shown.

Figure 29-1. DAC overview.



A DAC conversion is automatically started when new data to be converted are available. Events from the event system can also be used to trigger a conversion, and this enables synchronized and timed conversions between the DAC and other peripherals, such as a timer/counter. The DMA controller can be used to transfer data to the DAC.

The DAC has high drive strength, and is capable of driving both resistive and capacitive loads, as well as loads which combine both. A low-power mode is available, which will reduce the drive strength of the output.

Internal and external voltage references can be used. The DAC output is also internally available for use as input to the analog comparator or ADC.

29.3 Voltage reference selection

The following can be used as the reference voltage (VREF) for the DAC”

- AV_{CC} voltage
- Accurate internal 1.00V voltage
- External voltage applied to AREF pin on PORTA
- External voltage applied to AREF pin on PORTB

29.4 Starting a Conversion

By default, conversions are started automatically when new data are written to the channel data register. It is also possible to enable events from the event system to trigger conversion starts. When enabled, a new conversion is started when the DAC channel receives an event and the channel data register has been updated. This enables conversion starts to be synchronized with external events and/or timed to ensure regular and fixed conversion intervals.

29.5 Output and output channels

The two DAC channels have fully independent outputs and individual data and conversion control registers. This enables the DAC to create two different analog signals. The channel 0 output can also be made internally available as input for the Analog Comparator and the ADC.

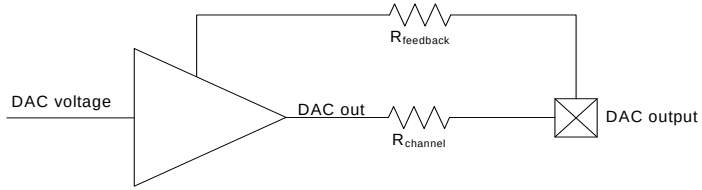
The output voltage from a DAC channel (V_{DAC}) is given as:

$$V_{DACn} = \frac{CHnDATA}{0xFFF} VREF$$

29.6 DAC Output model

Each DAC output channel has a driver buffer with feedback to ensure that the voltage on the DAC output pin is equal to the DACs internal voltage. Figure 29-2 on page 368 shows the DAC output model. For details on $R_{channel}$, refer to the DAC characteristics in the device data sheet.

Figure 29-2. DAC output model



29.7 DAC clock

The DAC is clocked directly from the peripheral clock (clk_{PER}), and this puts a limitation on how fast new data can be clocked into the DAC data registers.

29.8 Low Power mode

To reduce the power consumption in DAC conversions, the DAC may be set in a Low Power mode. Conversion time will be longer if new conversions are started in this mode. This increases the DAC conversion time per DAC channel by a factor of two.

29.9 Calibration

For improved accuracy, it is possible to calibrate for gain and offset errors in the DAC.

To get the best calibration result, it is recommended to use the same DAC configuration during calibration as will be used in the final application. The theoretical transfer function for the DAC was shown by the equation in "Output and output channels" on page 368. Including gain and offset errors, the DAC output value can be expressed as:

Equation 29-1.Calculation of DAC output value

$$V_{DAC} = VREF \cdot \frac{DATA}{0xFFF} \cdot ERROR_{GAIN} + V_{OFFSET}$$

To calibrate for offset error, output the DAC channel's middle code (0x800) and adjust the offset calibration value until the measured output value is as close as possible to the middle value (VREF / 2). The formula for the offset calibration is given by the Equation 29-2 on page 369, where OCAL is OFFSETCAL and GCAL is GAINCAL.

Equation 29-2.Offset calibration.

$$V_{OCAL} = VREF \cdot (2 + OCAL_7) \cdot \left(1 + \frac{OCAL_6}{64} + \frac{OCAL_5}{128} + \frac{OCAL_4}{256} + \frac{OCAL_3}{512} + \frac{OCAL_2}{1024} + \frac{OCAL_1}{2048} + \frac{OCAL_0}{4096}\right)$$

To calibrate for gain error, output the DAC channel's maximum code (0xFFF) and adjust the gain calibration value until the measured output value is as close as possible to the top value (VREF x 4095 / 4096). The gain calibration controls the slope of the DAC characteristic by rotating the transfer function around the middle code. The formula for gain calibration is given by the Equation 29-3 on page 369.

Equation 29-3.Gain calibration.

$$V_{GCAL} = V_{DAC} - \frac{VREF}{2} \cdot (1 - 2 \cdot GCAL_7) \cdot \left(\frac{GCAL_6}{16} + \frac{GCAL_5}{32} + \frac{GCAL_4}{64} + \frac{GCAL_3}{128} + \frac{GCAL_2}{256} + \frac{GCAL_1}{512} + \frac{GCAL_0}{1024}\right)$$

Including calibration in the equation, the DAC output can be expressed by Equation 29-4 on page 369.

Equation 29-4.DAC output calculation

$$V_{DAC\_out} = V_{DAC} + V_{OCAL} + V_{GCAL}$$

29.10 Register description

29.10.1 CTRLA – Control register A

Bit	7	6	5	4	3	2	1	0
+0x00	–	–	–	IDOEN	CH1EN	CH0EN	LPMODE	ENABLE
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:5 – Reserved
These bite are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 4 – IDOEN: Internal Output Enable
Setting this bit will enable the internal DAC channel 0 output to be used by the Analog Comparator and ADC. This will then also disable the output pin for DAC Channel 0.

Bit 3 – CH1EN: Channel 1 Output Enable
Setting this bit will make channel 1 available on the output pin.

Bit 2 – CH0EN: Channel 0 Output Enable
Setting this bit will make channel 0 available on the output pin unless IDOEN is set to 1.

Bit 1 – LPMODE: Low Power Mode
Setting this bit enables the DAC low-power mode. The DAC is turned off between each conversion to save current. Conversion time will be doubled when new conversions are started in this mode.

Bit 0 – ENABLE: Enable
This bit enables the entire DAC.

29.10.2 CTRLB – Control register B

Bit	7	6	5	4	3	2	1	0
+0x01	–	CHSEL[1:0]		–	–	–	CH1TRIG	CH0TRIG
Read/Write	R	R/W	R/W	R	R	R	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – Reserved
This bit is unused and reserved for future use. For compatibility with future devices, always write this bit to zero when this register is written.

Bit 6:5 – CHSEL[1:0]: Channel Selection
These bits control which DAC channels are enabled and operating. Table 29-1 on page 370 shows the available selections.

Table 29-1. DAC channel selection.

CHSEL[1:0]	Group configuration	Description
00	SINGLE	Single-channel operation on channel 0
01	SINGLE1	Single-channel operation on channel 1
10	DUAL	Dual-channel operation
11	–	Reserved

Bit 4:2 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 1 – CH1TRIG: Auto trigged mode Channel 1
If this bit is set, an event on the configured event channel, set in EVCTRL, will trigger a conversion on DAC channel 1 if its data register, CH1DATA, has been updated.

Bit 0 – CH0TRIG: Auto trigged mode Channel 0
If this bit is set, an event on the configured event channel, set in EVCTRL, will trigger a conversion on DAC channel 0 if its data register, CH0DATA, has been updated.

29.10.3 CTRLC – Control register C

Bit	7	6	5	4	3	2	1	0
+0x02	–	–	–	REFSEL[1:0]		–	–	LEFTADJ
Read/Write	R	R	R	R/W	R/W	R	R	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:5 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 4:3 – REFSEL[1:0]: Reference Selection
These bits select the reference voltage for the DAC according to Table 29-2 on page 371.

Table 29-2. DAC reference selection.

CHSEL[1:0]	Group configuration	Description
00	INT1V	Internal 1.00V
01	AVCC	AV_{CC}
10	AREFA	AREF on PORTA
11	AREFB	AREF on PORTB

Bit 2:1 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 0 - LEFTADJ: Left-Adjust Value
If this bit is set, CH0DATA and CH1DATA are left-adjusted.

29.10.4 EVCTRL – Event Control register

Bit	7	6	5	4	3	2	1	0
+0x03	–	–	–	–	EVSEL[3:0]			
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:4 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 3 – EVSEL[3]: Event Selection bit 3
Setting this bit to 1 enables event channel EVSEL[2:0]+1 as the trigger source for DAC Channel 1. When this bit is 0, the same event channel is used as the trigger source for both DAC channels.

Bit 2:0 – EVSEL[2:0]: Event Channel Input Selection
These bits select which Event System channel is used for triggering a DAC conversion. Table 29-3 on page 372 shows the available selections.

Table 29-3. DAC reference selection.

EVSEL[2:0]	Group configuration	Description
000	0	Event channel 0 as input to DAC
001	1	Event channel 1 as input to DAC
010	2	Event channel 2 as input to DAC
011	3	Event channel 3 as input to DAC
100	4	Event channel 4 as input to DAC
101	5	Event channel 5 as input to DAC
110	6	Event channel 6 as input to DAC
111	7	Event channel 7 as input to DAC

29.10.5 STATUS – Status register

Bit	7	6	5	4	3	2	1	0
+0x05	–	–	–	–	–	–	CH1DRE	CH0DRE
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 unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 1 – CH1DRE: Channel 1 Data Register Empty
This bit when set indicates that the data register for channel 1 is empty, meaning that a new conversion value may be written. Writing to the data register when this bit is cleared will cause the pending conversion data to be overwritten. This bit is directly used for DMA requests.

Bit 0 – CH0DRE: Channel 0 Data register Empty
This bit when set indicates that the data register for channel 0 is empty, meaning that a new conversion value may be written. Writing to the data register when this bit is cleared will cause the pending conversion data to be overwritten. This bit is directly used for DMA requests.

29.10.6 CH0DATAH – Channel 0 Data register High

These two channel data registers, CHnDATAH and CHnDATAL, are the high byte and low byte, respectively, of the 12-bit CHnDATA value that is converted to a voltage on DAC channel n. By default, the 12 bits are distributed with 8 bits in CHnDATAL and 4 bits in the four lsb positions of CHnDATAH (right-adjusted).To select left-adjusted data, set the LEFTADJ bit in the CTRLC register.

When left adjusted data is selected, it is possible to do 8-bit conversions by writing only to the high byte of CHnDATA, i.e., CHnDATAH. The TEMP register should be initialized to zero if only 8-bit conversion mode is used.

	Bit	7	6	5	4	3	2	1	0
Right-adjust	+0x19	–	–	–	–	CHDATA[11:8]			
Left-adjust		CHDATA[11:4]							
Right-adjust	Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Left-adjust	Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Right-adjust	Initial Value	0	0	0	0	0	0	0	0
Left-adjust	Initial Value	0	0	0	0	0	0	0	0

29.10.6.1 Right-adjusted

Bit 7:4 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 3:0 – CHDATA[11:8]: Conversion Data Channel 0, Four msbs
These bits are the four msbs of the 12-bit value to convert to channel 0 in right-adjusted mode.

29.10.6.2 Left-adjusted

Bits 7:0 – CHDATA[11:4]: Conversion Data Channel 0, Eight msbs
These bits are the eight msbs of the 12-bit value to convert to channel 0 in left-adjusted mode

29.10.7 CH0DATAL – Channel 0 Data register Low

	Bit	7	6	5	4	3	2	1	0
Right-adjust	+0x18	CHDATA[7:0]							
Left-adjust		CHDATA[3:0]				–	–	–	–
Right-adjust	Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Left-adjust	Read/Write	R/W	R/W	R/W	R/W	R	R	R	R
Right-adjust	Initial Value	0	0	0	0	0	0	0	0
Left-adjust	Initial Value	0	0	0	0	0	0	0	0

29.10.7.1 Right-adjusted

Bit 7:0 – CHDATA[7:0]: Conversion Data Channel 0, Eight lsbs
These bits are the eight lsbs of the 12-bit value to convert to channel 0 in right-adjusted mode.

29.10.7.2 Left-adjusted

Bit 7:4 – CHDATA[3:0]: Conversion Data Channel 0, Four lsbs
These bits are the four lsbs of the 12-bit value to convert to channel 0 in left-adjusted mode.

Bit 3:0 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

29.10.8 CH1DATAH – Channel 1 Data register High

	Bit	7	6	5	4	3	2	1	0
Right-adjust	+0x1B	–	–	–	–	CHDATA[11:8]			
Left-adjust		CHDATA[11:4]							
Right-adjust	Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Left-adjust	Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Right-adjust	Initial Value	0	0	0	0	0	0	0	0
Left-adjust	Initial Value	0	0	0	0	0	0	0	0

29.10.8.1 Right-adjusted

Bit 7:4 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

Bit 3:0 – CHDATA[11:8]: Conversion Data Channel 1, Four msbs
These bits are the four msbs of the 12-bit value to convert to channel 1 in right-adjusted mode.

29.10.8.2 Left-adjusted

Bit 7:0 – CHDATA[11:4]: Conversion Data Channel 1, Eight msbs
These bits are the eight msbs of the 12-bit value to convert to channel 1 in left-adjusted mode.

29.10.9 CH1DATAL – Channel 1 Data register Low

	Bit	7	6	5	4	3	2	1	0
Right-adjust	+0x1A	CHDATA[7:0]							
Left-adjust		CHDATA[3:0]				–	–	–	–
Right-adjust	Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Left-adjust	Read/Write	R/W	R/W	R/W	R/W	R	R	R	R
Right-adjust	Initial Value	0	0	0	0	0	0	0	0
Left-adjust	Initial Value	0	0	0	0	0	0	0	0

29.10.9.1Right-adjusted

Bit 7:0 – CHDATA[7:0]: Conversion Data Channel 1, Eight lsbs
These bits are the eight lsbs of the 12-bit value to convert to channel 1 in right-adjusted mode.

29.10.9.2 Left-adjusted

Bits 7:4 – CHDATA[3:0]: Conversion Data Channel 1, Four lsbs
These bits are the four lsbs of the 12-bit value to convert to channel 1 in left-adjusted mode.

Bit 3:0 – Reserved
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

29.10.10 CH0GAINCAL – Gain Calibration register

Bit	7	6	5	4	3	2	1	0
+0x08/+0x0A	CH0GAINCAL[7:0]							
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 7:0 – CH0GAINCAL[7:0]: Gain Calibration value
These bits are used to compensate for the gain error in DAC channel 0. See "Calibration" on page 369 for details.

29.10.11 CH0OFFSETCAL – Offset Calibration register

Bit	7	6	5	4	3	2	1	0
+0x09	CH0OFFSETCAL[7:0]							
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 7:0 – CH0OFFSETCAL[7:0]: Offset Calibration value
These bits are used to compensate for the offset error in DAC channel 0. See "Calibration" on page 369 for details.

29.10.12 CH1GAINCAL – Gain Calibration register

Bit	7	6	5	4	3	2	1	0
+0x0A	CH1GAINCAL[7:0]							
Read/Write	R	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 7:0 – CH1GAINCAL[7:0]: Gain Calibration value
These bits are used to compensate for the gain error in DAC channel 1. See "Calibration" on page 369 for details.

29.10.13 CH1OFFSETCAL – Offset Calibration register

Bit	7	6	5	4	3	2	1	0
+0x0B	CH1OFFSETCAL[7:0]							
Read/Write	R	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 7:0 – CH1OFFSETCAL[7:0]: Offset Calibration value
These bits are used to compensate for the offset error in DAC channel 1. See "Calibration" on page 369 for details.

29.11 Register summary

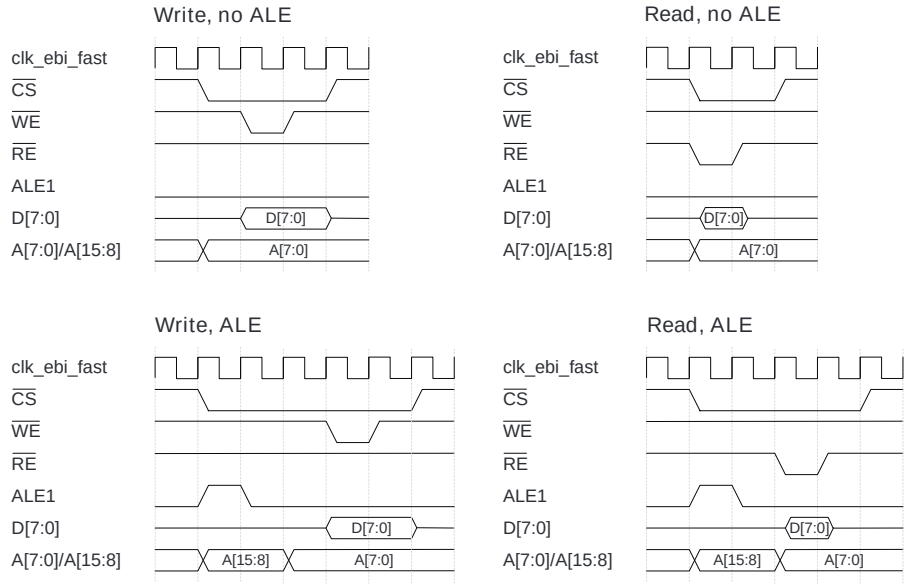
This is the I/O summary when the DAC is configured to give standard 12-bit results. The I/O summary for 12-bit left-adjusted results will be similar, but with some changes in the CHnDATAL and CHnDATAH data registers.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	CTRLA	–	–	–	IDOEN	CH1EN	CH0EN	LPMODE	ENABLE	370
+0x01	CTRLB	–	CHSEL[1:0]		–	–	–	CH1TRIG	CH0TRIG	370
+0x02	CTRLC	–	–	–	REFSEL[1:0]		–	–	LEFTADJ	371
+0x03	EVCTRL	–	–	–	–	EVSEL[3:0]				371
+0x04	Reserved	–	–	–	–	–	–	–	–	
+0x05	STATUS	–	–	–	–	–	–	CH1DRE	CH0DRE	372
+0x06	Reserved	–	–	–	–	–	–	–	–	
+0x07	Reserved	–	–	–	–	–	–	–	–	
+0x08	CH0GAINCAL	CH0GAINCAL[7:0]								374
+0x09	CH0OFFSETCAL	CH0OFFSETCAL[7:0]								375
+0x0A	CH1GAINCAL	CH1GAINCAL[7:0]								375
+0x0B	CH1OFFSETCAL	CH1OFFSETCAL[7:0]								375
+0x12	Reserved	–	–	–	–	–	–	–	–	
+0x13	Reserved	–	–	–	–	–	–	–	–	
+0x14	Reserved	–	–	–	–	–	–	–	–	
+0x15	Reserved	–	–	–	–	–	–	–	–	
+0x16	Reserved	–	–	–	–	–	–	–	–	
+0x17	Reserved	–	–	–	–	–	–	–	–	
+0x18	CH0DATAL	CHDATA[7:0]								373
+0x19	CH0DATAH	–	–	–	–	CHDATA[11:8]				372
+0x1A	CH1DATAL	CHDATA[7:0]								374
+0x1B	CH1DATAH	–	–	–	–	CHDATA[11:8]				374

36. Appendix A: EBI Timing Diagrams

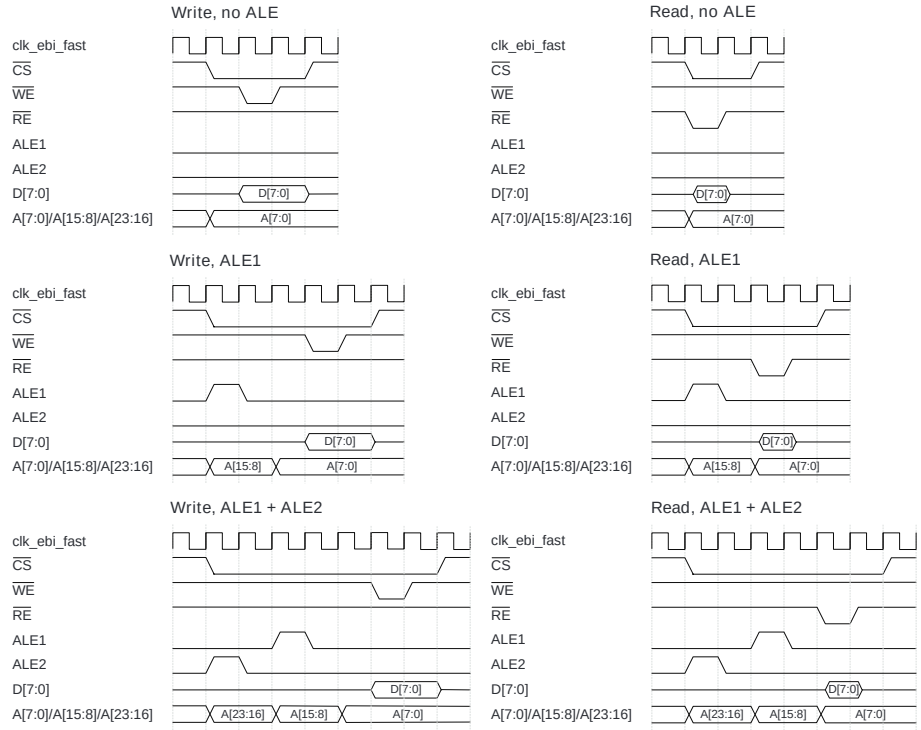
36.1 SRAM 3-Port ALE1 CS

Figure 36-1. SRAM 3- Port ALE1 CS



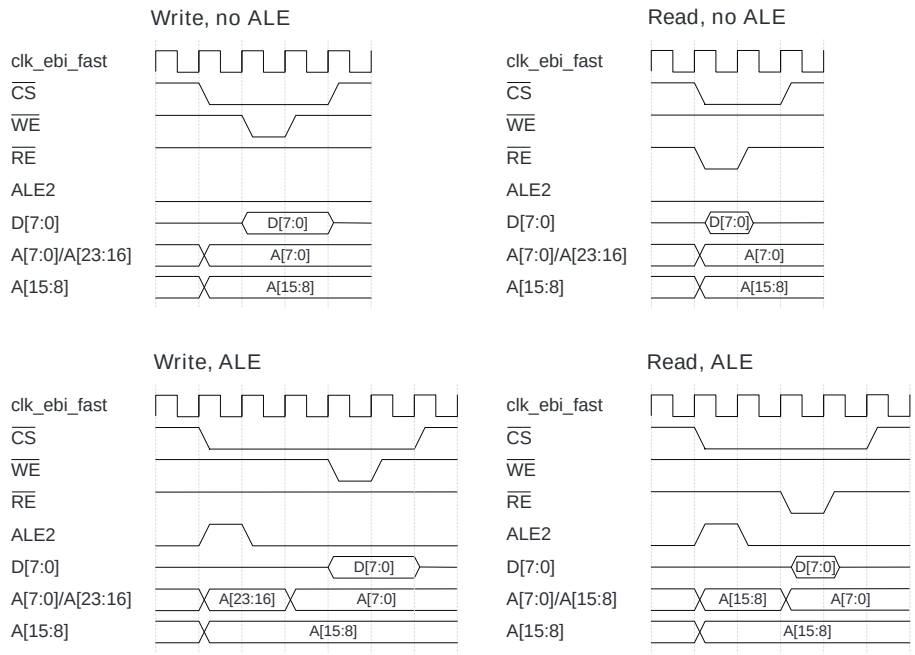
36.2 SRAM 3-Port ALE12 CS

Figure 36-2. SRAM 3- Port ALE12 CS



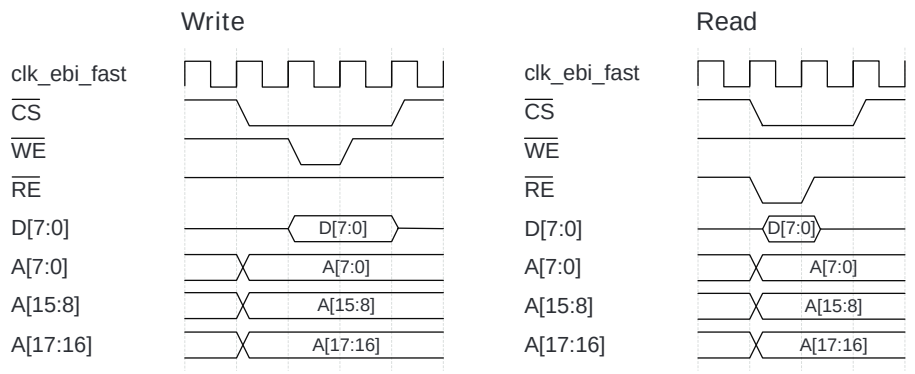
36.3 SRAM 4-Port ALE2 CS

Figure 36-3. SRAM 4- Port ALE2 CS



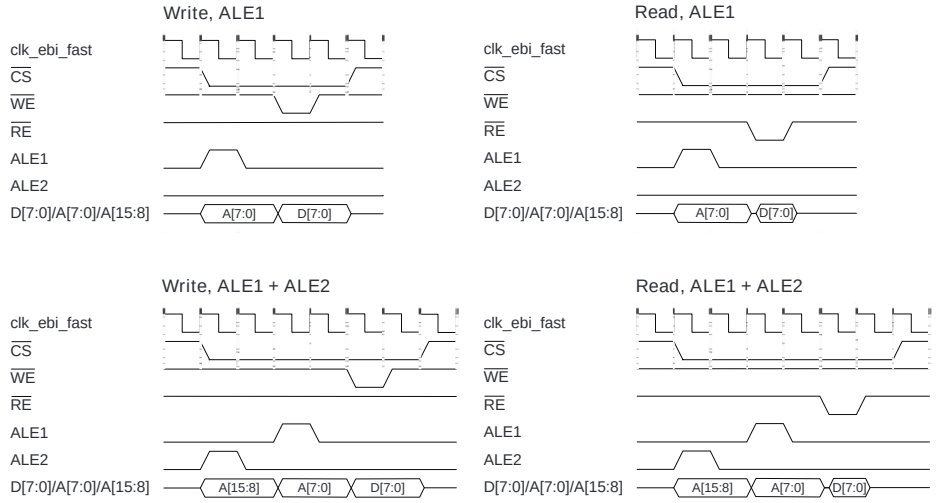
36.4 SRAM 4- Port NOALE CS

Figure 36-4. SRAM 4- Port NOALE CS



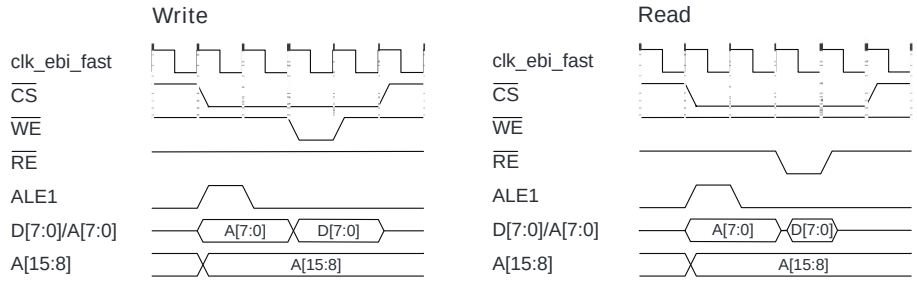
36.5 LPC 2- Port ALE12 CS

Figure 36-5. LPC 2- Port ALE12 CS



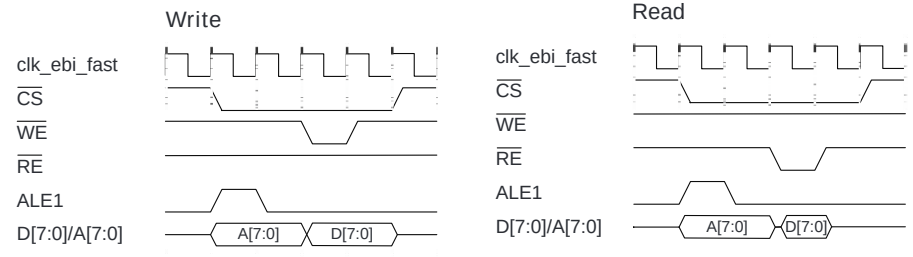
36.6 LPC 3- Port ALE1 CS

Figure 36-6. LPC 3- Port ALE1 CS



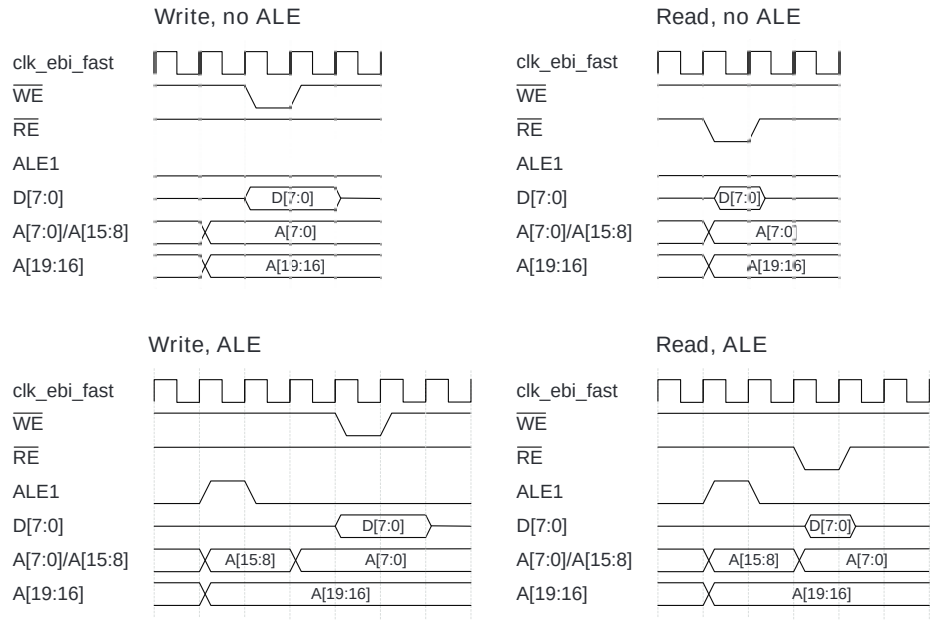
36.7 LPC 2- Port ALE1 CS

Figure 36-7. LPC 2- Port ALE1 CS



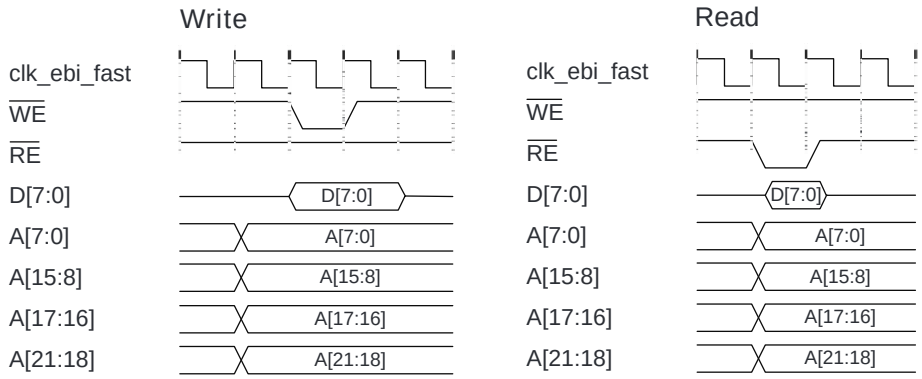
36.8 SRAM 3- Port ALE1 no CS

Figure 36-8. SRAM 3- Port ALE1 no CS



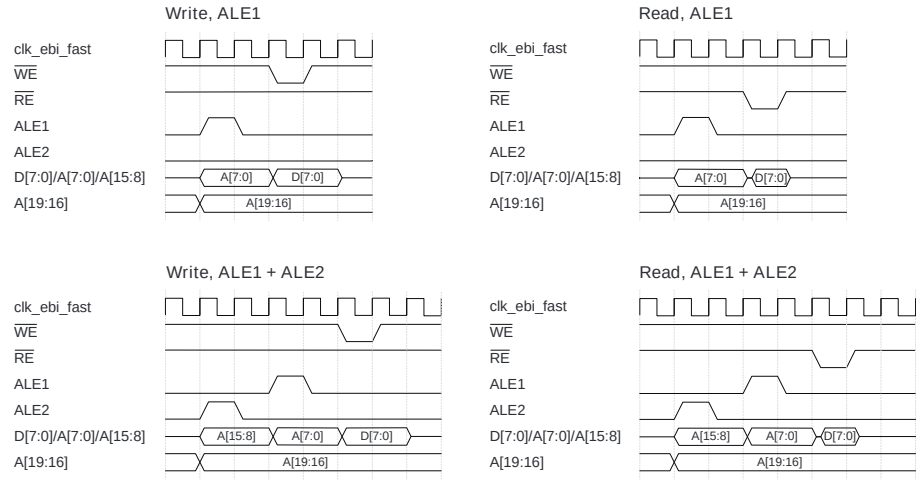
36.9 SRAM 4- Port NOALE no CS

Figure 36-9. SRAM 4- Port NOALE1 no CS



36.10 LPC 2- Port ALE12 no CS

Figure 36-10.LPC 2 - Port ALE12 no CS



37. Nomenclature

37.1 Symbols and operators

Table 37-1. Symbols and operators

+	Addition or logic OR
-	Subtraction
x	Multiplication
/	Division
$\overline{B}$	NOT (Here: NOT B)
⊓	AND
⊕	EXCLUSIVE OR
>	Greater
<	Less
=	Equal
≥	Greater or equal
≤	Less or equal
←	Store into
<<	Shift left

Note: The symbol "+" is used both as arithmetical addition and as a logic OR operand. If it is not apparent what the symbol means from the context, a footnote will clarify the issue.

37.2 Numerical notation

Table 37-2. Numerical notation

165	Decimal number
0b	Binary number (example 0b0101 = 5 decimal)
0x	Hexadecimal number (example 0xE = 14 decimal)
n, m, p, q	Represent numbers.
x, y, z, w	Represent letters.
X	Represents an unknown or don't care value for either a signal or a bus.
Z	Represents a high-impedance (floating) state for either a signal or a bus.
N/A	Not Applicable.

37.3 Memory size and type

Table 37-3. Memory size/type mnemonics

K	Kilo(2^{10} = 1024)
M	Mega(2^{20} = 1024*1024)
G	Giga(2^{30} = 1024*1024*1024)
k	kilo(10^3 = 1000)
b	bit(binary 0 or 1)
B	Byte(Collection of 8 bits)
W	Word(Collection of 2 Bytes)
L	Long(Collection of 2 Words)

37.4 Register and bits

Table 37-4. Register and bit mnemonics

R/W	Read/Write accessible register bit
R	Read-only register bit. (Must always be written to logic zero)
BIT	Bit names are shown in uppercase. (Example PINA1)
BITn..m	A set of bits from bit n down to m. (Example: PINA3..0 = {PINA3, PINA2, PINA1, PINA0}

37.5 Abbreviations

Table 37-5. Abbreviations

Abbreviation	Description
AC	Analog Comparator
ADC	Analog to Digital Converter
ADDR	Address
AES	Advanced Encryption Standard
ALE	Address Latch Enable
ALU	Arithmetic Logic Unit
AREF	Analog Reference
AV_{CC}	Analog supply power
AWeX	Advanced Waveform Extension
BB	Battery Backup
BLB	Boot Lock Bit
BOD	Brown-out Detector
BP	Breakpoint (OCD)
CAL	Calibration
CCP	Configuration Change Protection
CH	Channel
CLK	Clock
CLKSEL	Clock Select
CNT	Counter
COMP	Compare
CRC	Cyclic Redundancy Check
CS	Chip Select
CTRL	Control
DAC	Digital to Analog converter
DES	Data Encryption Standard
DFLL	Digital Frequency Locked Loop
DMA	Direct Memory Access
DMAC	Direct Memory Access Controller
DTI	Dead-time insertion
EBI	External Bus Interface
EEPROM	Electrically Erasable Programmable Read-Only Memory

EVACT	Event Action
EVSEL	Event Select
EVSYS	Event System
FAULT	Fault Control
FIFO	First-in first-out Buffer
FRQ	Frequency Generation
GND	Ground
GPIO	General Purpose Input/Output pin
HIRES	High Resolution
IC	Input Capture
IF	Interrupt Flag
INT	Interrupt
IOBUS	I/O data Bus (8-bit)
IRCOM	Infrared Communication Module
IRDA	Infrared Data Association
IREQ	Interrupt request
IVEC	Interrupt vector
LB	Lock Bit
LPC	Low Pin Count (EBI)
LVL	(Interrupt) Level
NMI	Non-Maskable Interrupt
NVM	Non-Volatile Memory
OC	Output Compare
OCD	On-chip Debug
PC	Program counter
PDI	Program and Debug Interface
PER	Period
PER	Peripheral (when used for as subscript in clock name)
PLL	Phase Lock Loop
PMIC	Programmable Multi-level Interrupt Controller
POR	Power-on reset
PWM	Pulse Width Modulation/Modulator
QDEC	Quadrature Decoder
RAM	Random-access memory

REF	Reference
RMW	Read-modify-write
RR	Round Robin
RTC	Real-Time Counter
RX	(Serial) Receiver
SDRAM	Synchronous Dynamic Random Access Memory
SMBus	System Management Bus
SP	Stack Pointer
SPI	Serial Peripheral Interface
SRAM	Static random-access memory
TC	Timer/Counter
TIF	Test Interface
TOSC	Timer/Counter (crystal) Oscillator
TWI	Two-wire interface
TX	(Serial) Transmitter
ULP	Ultra Low Power (oscillator)
USART	Universal Synchronous and Asynchronous serial Receiver and Transmitter
USB	Universal Serial Bus
V_{BAT}	Power supply Battery Backup
V_{CC}	Digital supply power
VREF	Voltage reference
WDT	Watchdog Timer
WG	Waveform Generator
XOSC	Crystal Oscillator

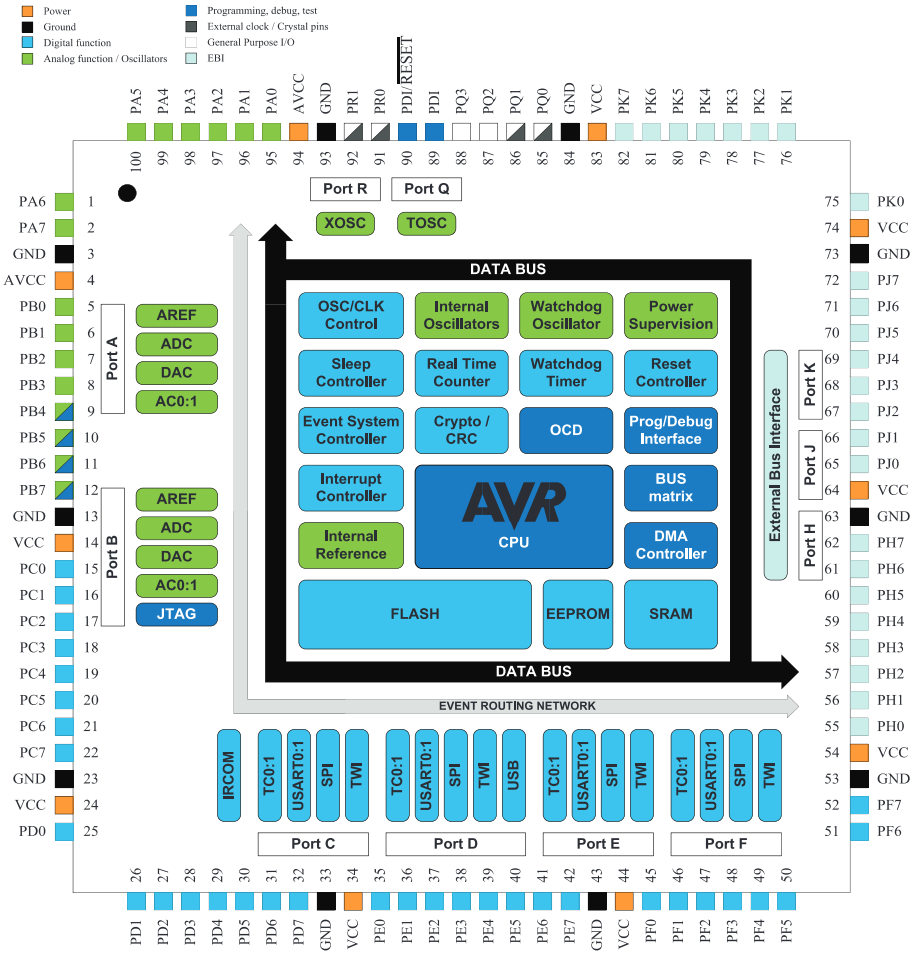
37.6 Conventions

- **Logic level one** is the voltage that corresponds to a Boolean true (1) state.
- **Logic level zero** is the voltage that corresponds to a Boolean false (0) state.
- **Set** refers specifically to establishing logic level one on a bit or bits.
- **Clear** refers specifically to establishing logic level zero on a bit or bits.
- **Asserted** means that a signal is in active logic state. An active low signal changes from logic level one to logic level zero when asserted, and an active high signal changes from logic level zero to logic level one.
- **Negated** means that an asserted signal changes logic state. An active low signal changes from logic level zero to logic level one when negated, and an active high signal changes from logic level one to logic level zero.
- **LSB** means least significant byte or bytes.
- **MSB** means most significant byte or bytes. References to low and high bytes are spelled out.
- **lsb** means least significant bit or bits.
- **msb** means most significant bit or bits.
- **A specific mnemonic** within a range is referred to by mnemonic and number. SP15 is bit 15 of the Stack Pointer; EEAR4 is line 4 of the EEPROM address bus
- **A range of mnemonics** is referred to by mnemonic and the numbers that define the range. EEARL[5:0] are bits 5 to 0 of EEPROM address bus

Table Of Contents

2. Pinout/Block Diagram

Figure 2-1. Block diagram and pinout.



For details on pinout and pin functions refer to "Pinout and Pin Functions" on page 56.

Six of the 32 registers can be used as three 16-bit address register pointers for data space addressing, enabling efficient address calculations. One of these address pointers can also be used as an address pointer for lookup tables in flash program memory.

7. Memories

7.1 Features

- Flash program memory
  - One linear address space
  - In-system programmable
  - Self-programming and boot loader support
  - Application section for application code
  - Application table section for application code or data storage
  - Boot section for application code or boot loader code
  - Separate read/write protection lock bits for all sections
  - Built in fast CRC check of a selectable flash program memory section
- Data memory
  - One linear address space
  - Single-cycle access from CPU
  - SRAM
  - EEPROM
    - Byte and page accessible
    - Optional memory mapping for direct load and store
  - I/O memory
    - Configuration and status registers for all peripherals and modules
    - 16 bit-accessible general purpose register for global variables or flags
  - External memory support
    - SRAM
    - SDRAM
    - Memory mapped external hardware
  - Bus arbitration
    - Deterministic priority handling between CPU, DMA controller, and other bus masters
  - Separate buses for SRAM, EEPROM, I/O memory and external memory
    - Simultaneous bus access for CPU and DMA controller
- Production signature row memory for factory programmed data
  - ID for each microcontroller device type
  - Serial number for each device
  - Calibration bytes for factory calibrated peripherals
- User signature row
  - One flash page in size
  - Can be read and written from software
  - Content is kept after chip erase

7.2 Overview

The Atmel AVR architecture has two main memory spaces, the program memory and the data memory. Executable code can reside only in the program memory, while data can be stored in the program memory and the data memory. The data memory includes the internal SRAM, and EEPROM for nonvolatile data storage. All memory spaces are linear and require no memory bank switching. Nonvolatile memory (NVM) spaces can be locked for further write and read/write operations. This prevents unrestricted access to the application software.

A separate memory section contains the fuse bytes. These are used for configuring important system functions, and can only be written by an external programmer.

The available memory size configurations are shown in “Ordering Information” on page 2. In addition each device has a flash memory signature rows for calibration data, device identification, serial number etc.

7.3 Flash Program Memory

The Atmel AVR XMEGA devices contain on-chip, in-system reprogrammable flash memory for program storage. The flash memory can be accessed for read and write from an external programmer through the PDI or from application software running in the device.

All AVR CPU instructions are 16 or 32 bits wide, and each flash location is 16 bits wide. The flash memory is organized in two main sections, the application section and the boot loader section. The sizes of the different sections are fixed, but device-dependent. These two sections have separate lock bits, and can have different levels of protection. The store program memory (SPM) instruction, which is used to write to the flash from the application software, will only operate when executed from the boot loader section.

The application section contains an application table section with separate lock settings. This enables safe storage of nonvolatile data in the program memory.

Figure 7-1. Flash program memory (Hexadecimal address).

Word Address			
ATxmega128A1U		ATxmega64A1U	
0		0	Application Section (bytes) (128K/64K)
			...
EFFF	/	77FF	
F000	/	7800	Application Table Section (bytes) (8K/4K)
FFFF	/	7FFF	
10000	/	8000	Boot Section (bytes) (8K/4K)
10FFF	/	87FF	

7.3.1 Application Section

The Application section is the section of the flash that is used for storing the executable application code. The protection level for the application section can be selected by the boot lock bits for this section. The application section can not store any boot loader code since the SPM instruction cannot be executed from the application section.

7.3.2 Application Table Section

The application table section is a part of the application section of the flash memory that can be used for storing data. The size is identical to the boot loader section. The protection level for the application table section can be selected by the boot lock bits for this section. The possibilities for different protection levels on the application section and the application table section enable safe parameter storage in the program memory. If this section is not used for data, application code can reside here.

7.3.3 Boot Loader Section

While the application section is used for storing the application code, the boot loader software must be located in the boot loader section because the SPM instruction can only initiate programming when executing from this section. The SPM instruction can access the entire flash, including the boot loader section itself. The protection level for the boot loader section can be selected by the boot loader lock bits. If this section is not used for boot loader software, application code can be stored here.

7.3.4 Production Signature Row

The production signature row is a separate memory section for factory programmed data. It contains calibration data for functions such as oscillators and analog modules. Some of the calibration values will be automatically loaded to the corresponding module or peripheral unit during reset. Other values must be loaded from the signature row and written to the corresponding peripheral registers from software. For details on calibration conditions, refer to "Electrical Characteristics" on page 74.

The production signature row also contains an ID that identifies each microcontroller device type and a serial number for each manufactured device. The serial number consists of the production lot number, wafer number, and wafer coordinates for the device. The device ID for the available devices is shown in Table 7-1.

The production signature row cannot be written or erased, but it can be read from application software and external programmers.

Table 7-1. Device ID bytes.

Device	Device ID bytes		
	Byte 2	Byte 1	Byte 0
ATxmega64A1U	4E	96	1E
ATxmega128A1U	4C	97	1E

7.3.5 User Signature Row

The user signature row is a separate memory section that is fully accessible (read and write) from application software and external programmers. It is one flash page in size, and is meant for static user parameter storage, such as calibration data, custom serial number, identification numbers, random number seeds, etc. This section is not erased by chip erase commands that erase the flash, and requires a dedicated erase command. This ensures parameter storage during multiple program/erase operations and on-chip debug sessions.

7.4 Fuses and Lock bits

The fuses are used to configure important system functions, and can only be written from an external programmer. The application software can read the fuses. The fuses are used to configure reset sources such as brownout detector and watchdog, startup configuration, JTAG enable, and JTAG user ID.

The lock bits are used to set protection levels for the different flash sections (that is, if read and/or write access should be blocked). Lock bits can be written by external programmers and application software, but only to stricter protection levels. Chip erase is the only way to erase the lock bits. To ensure that flash contents are protected even during chip erase, the lock bits are erased after the rest of the flash memory has been erased.

An unprogrammed fuse or lock bit will have the value one, while a programmed fuse or lock bit will have the value zero.

Both fuses and lock bits are reprogrammable like the flash program memory.

7.5 Data Memory

The data memory contains the I/O memory, internal SRAM, optionally memory mapped EEPROM, and external memory if available. The data memory is organized as one continuous memory section, see Figure 7-2 on page 14. To simplify development, I/O Memory, EEPROM and SRAM will always have the same start addresses for all Atmel AVR XMEGA devices. The address space for External Memory will always start at the end of Internal SRAM and end at address 0xFFFFFF.

Figure 7-2. Data memory map (hexadecimal address).

Byte Address	ATxmega64A1U
0	I/O Registers (4K)
FFF	
1000	EEPROM (2K)
17FF	
	RESERVED
2000	Internal SRAM (4K)
2FFF	
4000	External Memory (0 to 16MB)
FFFFFF	

Byte Address	ATxmega128A1U
0	I/O Registers (4KB)
FFF	
1000	EEPROM (2K)
17FF	
	RESERVED
2000	Internal SRAM (8K)
3FFF	
3000	External Memory (0 to 16MB)
FFFFFF	

7.6 EEPROM

XMEGA AU devices have EEPROM for nonvolatile data storage. It is either addressable in a separate data space (default) or memory mapped and accessed in normal data space. The EEPROM supports both byte and page access. Memory mapped EEPROM allows highly efficient EEPROM reading and EEPROM buffer loading. When doing this, EEPROM is accessible using load and store instructions. Memory mapped EEPROM will always start at hexadecimal address 0x1000.

7.7 I/O Memory

The status and configuration registers for peripherals and modules, including the CPU, are addressable through I/O memory locations. All I/O locations can be accessed by the load (LD/LDS/LDD) and store (ST/STS/STD) instructions, which is used to transfer data between the 32 registers in the register file and the I/O memory. The IN and OUT instructions can address I/O memory locations in the range 0x00 - 0x3F directly. In the address range 0x00 - 0x1F, single- cycle instructions for manipulation and checking of individual bits are available.

The I/O memory address for all peripherals and modules in XMEGA A1U is shown in the "Peripheral Module Address Map" on page 63.

7.7.1 General Purpose I/O Registers

The lowest 16 I/O memory addresses are reserved as general purpose I/O registers. These registers can be used for storing global variables and flags, as they are directly bit-accessible using the SBI, CBI, SBIS, and SBIC instructions.

7.8 External Memory

Four ports can be used for external memory, supporting external SRAM, SDRAM, and memory mapped peripherals such as LCD displays. Refer to "EBI – External Bus Interface" on page 48. The external memory address space will always start at the end of internal SRAM.

7.9 Data Memory and Bus Arbitration

Since the data memory is organized as four separate sets of memories, the different bus masters (CPU, DMA controller read and DMA controller write, etc.) can access different memory sections at the same time.

7.10 Memory Timing

Read and write access to the I/O memory takes one CPU clock cycle. A write to SRAM takes one cycle, and a read from SRAM takes two cycles. For burst read (DMA), new data are available every cycle. EEPROM page load (write) takes one cycle, and three cycles are required for read. For burst read, new data are available every second cycle. External memory has multi-cycle read and write. The number of cycles depends on the type of memory and configuration of the external bus interface. Refer to the instruction summary for more details on instructions and instruction timing.

7.11 Device ID and Revision

Each device has a three-byte device ID. This ID identifies Atmel as the manufacturer of the device and the device type. A separate register contains the revision number of the device.

7.12 I/O Memory Protection

Some features in the device are regarded as critical for safety in some applications. Due to this, it is possible to lock the I/O register related to the clock system, the event system, and the advanced waveform extensions. As long as the lock is enabled, all related I/O registers are locked and they can not be written from the application software. The lock registers themselves are protected by the configuration change protection mechanism.

7.13 JTAG Disable

It is possible to disable the JTAG interface from the application software. This will prevent all external JTAG access to the device until the next device reset or until JTAG is enabled again from the application software. As long as JTAG is disabled, the I/O pins required for JTAG can be used as normal I/O pins.

7.14 Flash and EEPROM Page Size

The flash program memory and EEPROM data memory are organized in pages. The pages are word accessible for the flash and byte accessible for the EEPROM.

Table 7-2 on page 15 shows the Flash Program Memory organization. Flash write and erase operations are performed on one page at a time, while reading the Flash is done one byte at a time. For Flash access the Z-pointer (Z[m:n]) is used for addressing. The most significant bits in the address (FPAGE) gives the page number and the least significant address bits (FWORD) gives the word in the page.

Table 7-2. Number of words and Pages in the Flash.

Devices	PC size	Flash	Page Size	FWORD	FPAGE	Application		Boot	
	bits	bytes	words			Size	No of pages	Size	No of pages
ATxmega64A1U	16	64K + 4K	128	Z[7:1]	Z[16:8]	64K	256	4K	16
ATxmega128A1U	17	128K + 8K	256	Z[8:1]	Z[17:9]	128K	256	8K	16

Table 7-3 on page 16 shows EEPROM memory organization for the Atmel AVR XMEGA A1U devices. EEPROM write and erase operations can be performed one page or one byte at a time, while reading the EEPROM is done one byte at a time. For EEPROM access the NVM Address Register (ADDR[m:n]) is used for addressing. The most significant bits in the address (E2PAGE) give the page number and the least significant address bits (E2BYTE) give the byte in the page.

Table 7-3. Number of Bytes and Pages in the EEPROM.

Devices	EEPROM	Page Size	E2BYTE	E2PAGE	No of Pages
	Size	bytes			
ATxmega64A1U	2K	32	ADDR[4:0]	ADDR[10:5]	64
ATxmega128A1U	2K	32	ADDR[4:0]	ADDR[10:5]	64

14. Interrupts and Programmable Multilevel Interrupt Controller

14.1 Features

- Short and predictable interrupt response time
- Separate interrupt configuration and vector address for each interrupt
- Programmable multilevel interrupt controller
  - Interrupt prioritizing according to level and vector address
  - Three selectable interrupt levels for all interrupts: low, medium and high
  - Selectable, round-robin priority scheme within low-level interrupts
  - Non-maskable interrupts for critical functions
- Interrupt vectors optionally placed in the application section or the boot loader section

14.2 Overview

Interrupts signal a change of state in peripherals, and this can be used to alter program execution. Peripherals can have one or more interrupts, and all are individually enabled and configured. When an interrupt is enabled and configured, it will generate an interrupt request when the interrupt condition is present. The programmable multilevel interrupt controller (PMIC) controls the handling and prioritizing of interrupt requests. When an interrupt request is acknowledged by the PMIC, the program counter is set to point to the interrupt vector, and the interrupt handler can be executed.

All peripherals can select between three different priority levels for their interrupts: low, medium, and high. Interrupts are prioritized according to their level and their interrupt vector address. Medium-level interrupts will interrupt low-level interrupt handlers. High-level interrupts will interrupt both medium- and low-level interrupt handlers. Within each level, the interrupt priority is decided from the interrupt vector address, where the lowest interrupt vector address has the highest interrupt priority. Low-level interrupts have an optional round-robin scheduling scheme to ensure that all interrupts are serviced within a certain amount of time.

Non-maskable interrupts (NMI) are also supported, and can be used for system critical functions.

14.3 Interrupt vectors

The interrupt vector is the sum of the peripheral's base interrupt address and the offset address for specific interrupts in each peripheral. The base addresses for the Atmel AVR XMEGA A1U devices are shown in Table 14-1 on page 27. Offset addresses for each interrupt available in the peripheral are described for each peripheral in the XMEGA AU manual. For peripherals or modules that have only one interrupt, the interrupt vector is shown in Table 14-1 on page 27. The program address is the word address.

Table 14-1. Reset and interrupt vectors.

Program address (base address)	Source	Interrupt description
0x000	RESET	
0x002	OSCF_INT_vect	Crystal oscillator failure interrupt vector (NMI)
0x004	PORTC_INT_base	Port C interrupt base
0x008	PORTR_INT_base	Port R interrupt base
0x00C	DMA_INT_base	DMA controller interrupt base
0x014	RTC_INT_base	Real time counter interrupt base
0x018	TWIC_INT_base	Two-Wire interface on port C interrupt base
0x01C	TCC0_INT_base	Timer/counter 0 on port C interrupt base

Program address (base address)	Source	Interrupt description
0x028	TCC1_INT_base	Timer/counter 1 on port C interrupt base
0x030	SPIC_INT_vect	SPI on port C interrupt vector
0x032	USARTC0_INT_base	USART 0 on port C interrupt base
0x038	USARTC1_INT_base	USART 1 on port C interrupt base
0x03E	AES_INT_vect	AES interrupt vector
0x040	NVM_INT_base	Nonvolatile memory interrupt base
0x044	PORTB_INT_base	Port B interrupt base
0x048	ACB_INT_base	Analog comparator on port B interrupt base
0x04E	ADCB_INT_base	Analog to digital converter on port B interrupt base
0x056	PORTE_INT_base	Port E interrupt base
0x05A	TWIE_INT_base	Two-Wire interface on port E interrupt base
0x05E	TCE0_INT_base	Timer/counter 0 on port E interrupt base
0x06A	TCE1_INT_base	Timer/counter 1 on port E interrupt base
0x072	SPIE_INT_vect	SPI on port E interrupt vector
0x074	USARTE0_INT_base	USART 0 on port E interrupt base
0x07A	USARTE1_INT_base	USART 1 on port E interrupt base
0x080	PORTD_INT_base	Port D interrupt base
0x084	PORTA_INT_base	Port A interrupt base
0x088	ACA_INT_base	Analog comparator on Port A interrupt base
0x08E	ADCA_INT_base	Analog to digital converter on Port A interrupt base
0x096	TWID_INT_base	Two-Wire Interface on port D interrupt base
0x09A	TCD0_INT_base	Timer/counter 0 on port D interrupt base
0x0A6	TCD1_INT_base	Timer/counter 1 on port D interrupt base
0x0AE	SPID_INT_vector	SPI on port D interrupt vector
0x0B0	USARTD0_INT_base	USART 0 on port D interrupt base
0x0B6	USARTD1_INT_base	USART 1 on port D interrupt base
0x0BC	PORTQ_INT_base	Port Q INT base
0x0C0	PORTH_INT_base	Port H INT base
0x0C4	PORTJ_INT_base	Port J INT base
0x0C8	PORTK_INT_base	Port K INT base
0x0D0	PORTF_INT_base	Port F INT base
0x0D4	TWIF_INT_base	Two-Wire interface on Port F INT base
0x0D8	TCF0_INT_base	Timer/counter 0 on port F interrupt base

Program address (base address)	Source	Interrupt description
0x0E4	TCF1_INT_base	Timer/counter 1 on port F interrupt base
0x0EC	SPIF_INT_vector	SPI ion port F interrupt base
0x0EE	USARTF0_INT_base	USART 0 on port F interrupt base
0x0F4	USARTF1_INT_base	USART 1 on port F interrupt base
0x0FA	USB_INT_base	USB on port D interrupt base

33. Pinout and Pin Functions

The device pinout is shown in "Pinout/Block Diagram" on page 3. In addition to general purpose I/O functionality, each pin can have several alternate functions. This will depend on which peripheral is enabled and connected to the actual pin. Only one of the pin functions can be used at time.

33.1 Alternate Pin Function Description

The tables below show the notation for all pin functions available and describe its function.

33.1.1 Operation/ Power Supply

V_{CC}	Digital supply voltage
AV_{CC}	Analog supply voltage
GND	Ground

33.1.2 Port Interrupt functions

SYNC	Port pin with full synchronous and limited asynchronous interrupt function
ASYNC	Port pin with full synchronous and full asynchronous interrupt function

33.1.3 Analog functions

ACn	Analog Comparator input pin n
ACnOUT	Analog Comparator n Output
ADCn	Analog to Digital Converter input pin n
DACn	Digital to Analog Converter output pin n
A_{REF}	Analog Reference input pin

33.1.4 External Bus Interface functions

An	Address line n	
Dn	Data line n	
$\overline{CSn}$	Chip Select n	
ALEn	Address Latch Enable pin n	(SRAM)
$\overline{RE}$	Read Enable	(SRAM)
$\overline{WE}$	External Data Memory Write	(SRAM /SDRAM)
BAn	Bank Address	(SDRAM)
$\overline{CAS}$	Column Access Strobe	(SDRAM)
CKE	SDRAM Clock Enable	(SDRAM)
CLK	SDRAM Clock	(SDRAM)
$\overline{DQM}$	Data Mask Signal/Output Enable	(SDRAM)
$\overline{RAS}$	Row Access Strobe	(SDRAM)

33.1.5 Timer/Counter and AWEX functions

OCnxLS	Output Compare Channel x Low Side for Timer/Counter n
OCnxHS	Output Compare Channel x High Side for Timer/Counter n

33.1.6 Communication functions

SCL	Serial Clock for TWI
SDA	Serial Data for TWI
SCLIN	Serial Clock In for TWI when external driver interface is enabled
SCLOUT	Serial Clock Out for TWI when external driver interface is enabled
SDAIN	Serial Data In for TWI when external driver interface is enabled
SDAOUT	Serial Data Out for TWI when external driver interface is enabled
XCKn	Transfer Clock for USART n
RXDn	Receiver Data for USART n
TXDn	Transmitter Data for USART n
$\overline{SS}$	Slave Select for SPI
MOSI	Master Out Slave In for SPI
MISO	Master In Slave Out for SPI
SCK	Serial Clock for SPI
D-	Data- for USB
D+	Data+ for USB

33.1.7 Oscillators, Clock and Event

TOSCn	Timer Oscillator pin n
XTALn	Input/Output for Oscillator pin n
CLKOUT	Peripheral Clock Output
EVOUT	Event Channel 0 Output

33.1.8 Debug/System functions

$\overline{RESET}$	Reset pin
PDI_CLK	Program and Debug Interface Clock pin
PDI_DATA	Program and Debug Interface Data pin
TCK	JTAG Test Clock
TDI	JTAG Test Data In
TDO	JTAG Test Data Out
TMS	JTAG Test Mode Select

33.2 Alternate Pin Functions

The tables below show the primary/default function for each pin on a port in the first column, the pin number in the second column, and then all alternate pin functions in the remaining columns. The head row shows what peripheral that enable and use the alternate pin functions.

For better flexibility, some alternate functions also have selectable pin locations for their functions, this is noted under the the first table where this apply.

Table 33-1. Port A - alternate functions.

PORTA	PIN #	INTERRUPT	ADCA POS /GAINPOS	ADCB POS	ADCA NEG	ADCA GAINNEG	ACA POS	ACA NEG	ACA OUT	DACA	REFA
GND	93										
AVCC	94										
PA0	95	SYNC	ADC0	ADC8	ADC0		AC0	AC0			AREF
PA1	96	SYNC	ADC1	ADC9	ADC1		AC1	AC1			
PA2	97	SYNC/ASYNC	ADC2	ADC10	ADC2		AC2			DAC0	
PA3	98	SYNC	ADC3	ADC11	ADC3		AC3	AC3		DAC1	
PA4	99	SYNC	ADC4	ADC12		ADC4	AC4				
PA5	100	SYNC	ADC5	ADC13		ADC5	AC5	AC5			
PA6	1	SYNC	ADC6	ADC14		ADC6	AC6		AC1OUT		
PA7	2	SYNC	ADC7	ADC15		ADC7		AC7	AC0OUT		

Table 33-2. Port B - alternate functions.

PORT B	PIN#	INTERRUPT	ADCAPOS	ADCBPOS/ GAINPOS	ADCB NEG	ADCB GAIN NEG	ACB POS	ACB NEG	ACB OUT	DACB	REFB	JTAG
GND	3											
AVCC	4											
PB0	5	SYNC	ADC8	ADC0	ADC0		AC0	AC0			AREF	
PB1	6	SYNC	ADC9	ADC1	ADC1		AC1	AC1				
PB2	7	SYNC/ASYNC	ADC10	ADC2	ADC2		AC2			DAC0		
PB3	8	SYNC	ADC11	ADC3	ADC3		AC3	AC3		DAC1		
PB4	9	SYNC	ADC12	ADC4		ADC4	AC4					TMS
PB5	10	SYNC	ADC13	ADC5		ADC5	AC5	AC5				TDI
PB6	11	SYNC	ADC14	ADC6		ADC6	AC6		AC1OUT			TCK
PB7	12	SYNC	ADC15	ADC7		ADC7		AC7	AC0OUT			TDO

Table 33-3. Port C - alternate functions.

PORT C	PIN#	INTERRUPT	TCC0[1][2]	AWEXC	TCC1	USARTC0[3]	USARTC1	SPIC[4]	TWIC	CLOCKOUT[5]	EVENTOUT[6]
GND	13										
VCC	14										
PC0	15	SYNC	OC0A	$\overline{\text{OC0ALS}}$					SDA		
PC1	16	SYNC	OC0B	OC0AHS		XCK0			SCL		
PC2	17	SYNC/ASYNC	OC0C	$\overline{\text{OC0BLS}}$		RXD0					
PC3	18	SYNC	OC0D	OC0BHS		TXD0					
PC4	19	SYNC		$\overline{\text{OC0CLS}}$	OC1A			$\overline{\text{SS}}$			
PC5	20	SYNC		OC0CHS	OC1B		XCK1	MOSI			
PC6	21	SYNC		$\overline{\text{OC0DLS}}$			RXD1	MISO		clk_{RTC}	
PC7	22	SYNC		OC0DHS			TXD1	SCK		clk_{PER}	EVOUT

Notes:
1. Pin mapping of all TC0 can optionally be moved to high nibble of port. Refer to Pin Remap register in I/O Ports in the XMEGA AU Manual.
2. If TC0 is configured as TC2 all eight pins can be used for PWM output. Refer to Pin Remap register in I/O Ports in the XMEGA AU Manual..
3. Pin mapping of all USART0 can optionally be moved to high nibble of port. Refer to Pin Remap register in I/O Ports in the XMEGA AU Manual..
4. Pins MOSI and SCK for all SPI can optionally be swapped.Refer to Pin Remap register in I/O Ports in the XMEGA AU Manual.
5. CLKOUT can optionally be moved between port C, D and E and between pin 4 and 7. Refer to CLKEVOUT register in I/O port configuration in the XMEGA AU Manual.
6. EVOUT can optionally be moved between port C, D and E and between pin 4 and 7. Refer to CLKEVOUT register in I/O port configuration in the XMEGA AU Manual.

Table 33-4. Port D - alternate functions.

PORT D	PIN#	INTERRUPT	TCD0	TCD1	USBD	USARTD0	USARTD1	SPID	TWID	CLOCKOUT	EVENTOUT
GND	23										
VCC	24										
PD0	25	SYNC	OC0A						SDA		
PD1	26	SYNC	OC0B			XCK0			SCL		
PD2	27	SYNC/ASYNC	OC0C			RXD0					
PD3	28	SYNC	OC0D			TXD0					
PD4	29	SYNC		OC1A				$\overline{\text{SS}}$			
PD5	30	SYNC		OC1B			XCK1	MOSI			
PD6	31	SYNC			D-		RXD1	MISO			
PD7	32	SYNC			D+		TXD1	SCK		clk_{PER}	EVOUT

Table 33-5. Port E - alternate functions.

PORT E	PIN #	INTERRUPT	TCE0	AWEXE	TCE1	USARTE0	USARTE1	SPIE	TWIE	CLOCKOUT	EVENTOUT
GND	33										
VCC	34										
PE0	35	SYNC	OC0A	$\overline{\text{OC0ALS}}$					SDA		
PE1	36	SYNC	OC0B	OC0AHS		XCK0			SCL		

PORT E	PIN #	INTERRUPT	TCE0	AWEXE	TCE1	USARTE0	USARTE1	SPIE	TWIE	CLOCKOUT	EVENTOUT
PE2	37	SYNC/ASYNC	OC0C	$\overline{\text{OC0BLS}}$		RXD0					
PE3	38	SYNC	OC0D	OC0BHS		TXD0					
PE4	39	SYNC		$\overline{\text{OC0CLS}}$	OC1A			$\overline{\text{SS}}$			
PE5	40	SYNC		OC0CHS	OC1B		XCK1	MOSI			
PE6	41	SYNC		$\overline{\text{OC0DLS}}$			RXD1	MISO			
PE7	42	SYNC		OC0DHS			TXD1	SCK		clk_{PER}	EVOUT

Notes: 1. All pins on the port can optionally be used for EBI chip select or address lines. Refer to the EBIOUT register description in the XMEGA AU Manual.

Table 33-6. Port F - alternate functions.

PORT F	PIN #	INTERRUPT	TCF0	TCF1	USARTF0	USARTF1	SPIF	TWIF
GND	43							
VCC	44							
PF0	45	SYNC	OC0A					SDA
PF1	46	SYNC	OC0B		XCK0			SCL
PF2	47	SYNC/ASYNC	OC0C		RXD0			
PF3	48	SYNC	OC0D		TXD0			
PF4	49	SYNC		OC1A			$\overline{\text{SS}}$	
PF5	50	SYNC		OC1B		XCK1	MOSI	
PF6	51	SYNC				RXD1	MISO	
PF7	52	SYNC				TXD1	SCK	

Note: 1. All pins on the port can optionally be used for EBI chip select or address lines. Refer to the EBIOUT register description in the XMEGA AU Manual.

Table 33-7. Port H - alternate functions.

PORT H	PIN #	INTERRUPT	SDRAM 3P	SRAM ALE1	SRAM ALE12	LPC3 ALE1	LPC2 ALE1	LPC2 ALE12
GND	53							
VCC	54							
PH0	55	SYNC	$\overline{\text{WE}}$	$\overline{\text{WE}}$	$\overline{\text{WE}}$	$\overline{\text{WE}}$	$\overline{\text{WE}}$	$\overline{\text{WE}}$
PH1	56	SYNC	$\overline{\text{CAS}}$	$\overline{\text{RE}}$	$\overline{\text{RE}}$	$\overline{\text{RE}}$	$\overline{\text{RE}}$	$\overline{\text{RE}}$
PH2	57	SYNC/ASYNC	$\overline{\text{RAS}}$	$\overline{\text{ALE1}}$	$\overline{\text{ALE1}}$	$\overline{\text{ALE1}}$	$\overline{\text{ALE1}}$	$\overline{\text{ALE1}}$
PH3	58	SYNC	$\overline{\text{DQM}}$		$\overline{\text{ALE2}}$			$\overline{\text{ALE2}}$
PH4	59	SYNC	BA0	$\overline{\text{CS0}}$/A16	$\overline{\text{CS0}}$	$\overline{\text{CS0}}$/A16	$\overline{\text{CS0}}$	$\overline{\text{CS0}}$/A16
PH5	60	SYNC	BA1	$\overline{\text{CS1}}$/A17	$\overline{\text{CS1}}$	$\overline{\text{CS1}}$/A17	$\overline{\text{CS1}}$	$\overline{\text{CS1}}$/A17
PH6	61	SYNC	CKE	$\overline{\text{CS2}}$/A18	$\overline{\text{CS2}}$	$\overline{\text{CS2}}$/A18	$\overline{\text{CS2}}$	$\overline{\text{CS2}}$/A18
PH7	62	SYNC	CLK	$\overline{\text{CS3}}$/A19	$\overline{\text{CS3}}$	$\overline{\text{CS3}}$/A19	$\overline{\text{CS3}}$	$\overline{\text{CS3}}$/A19

Notes: 1. CS0 - CS3 can optionally be moved to Port E or F
2. A16-A23 can optionally be moved to Port E or F when EBI configured in 4PORT mode. Refer to the EBIOUT register description in the XMEGA AU Manual.

Table 33-8. Port J - alternate functions.

PORT J	PIN #	INTERRUPT	SDRAM 3P	SRAM ALE1	SRAM ALE12	LPC3 ALE1	LPC2 ALE1	LPC2 ALE12
GND	63							
VCC	64							
PJ0	65	SYNC	D0	D0	D0	D0/A0	D0/A0	D0/A0/A8
PJ1	66	SYNC	D1	D1	D1	D1/A1	D1/A1	D1/A1/A9
PJ2	67	SYNC/ASYNC	D2	D2	D2	D2/A2	D2/A2	D2/A2/A10
PJ3	68	SYNC	D3	D3	D3	D3/A3	D3/A3	D3/A3/A11
PJ4	69	SYNC	A8	D4	D4	D4/A4	D4/A4	D4/A4/A12
PJ5	70	SYNC	A9	D5	D5	D5/A5	D5/A5	D5/A5/A13
PJ6	71	SYNC	A10	D6	D6	D6/A6	D6/A6	D6/A6/A14
PJ7	72	SYNC	A11	D7	D7	D7/A7	D7/A7	D7/A7/A15

Table 33-9. Port K - alternate functions.

PORT K	PIN #	INTERRUPT	SDRAM 3P	SRAM ALE1	SRAM ALE2	LPC3 ALE1
GND	73					
VCC	74					
PK0	75	SYNC	A0	A0/A8	A0/A8/A16	A8
PK1	76	SYNC	A1	A1/A9	A1/A9/A17	A9
PK2	77	SYNC/ASYNC	A2	A2/A10	A2/A10/A18	A10
PK3	78	SYNC	A3	A3/A11	A3/A11/A19	A11
PK4	79	SYNC	A4	A4/A12	A4/A12/A20	A12
PK5	80	SYNC	A5	A5/A13	A5/A13/A21	A13
PK6	81	SYNC	A6	A6/A14	A6/A14/A22	A14
PK7	82	SYNC	A7	A7/A15	A7/A15/A23	A15

Table 33-10. Port Q - alternate functions.

PORT Q	PIN #	INTERRUPT	TOSC
VCC	83		
GND	84		
PQ0	85	SYNC	TOSC1
PQ1	86	SYNC	TOSC2
PQ2	87	SYNC/ASYNC	
PQ3	88	SYNC	

Table 33-11. Port R - alternate functions.

PORT R	PIN #	INTERRUPT	PDI	XTAL
PDI	89		PDI_DATA	
$\overline{\text{RESET}}$	90		PDI_CLOCK	
PR0	91	SYNC		XTAL2
PR1	92	SYNC		XTAL1

35. Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
Arithmetic and Logic Instructions					
ADD	Rd, Rr	Add without Carry	Rd ← Rd + Rr	Z,C,N,V,S,H	1
ADC	Rd, Rr	Add with Carry	Rd ← Rd + Rr + C	Z,C,N,V,S,H	1
ADIW	Rd, K	Add Immediate to Word	Rd ← Rd + 1:Rd + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract without Carry	Rd ← Rd - Rr	Z,C,N,V,S,H	1
SUBI	Rd, K	Subtract Immediate	Rd ← Rd - K	Z,C,N,V,S,H	1
SBC	Rd, Rr	Subtract with Carry	Rd ← Rd - Rr - C	Z,C,N,V,S,H	1
SBCI	Rd, K	Subtract Immediate with Carry	Rd ← Rd - K - C	Z,C,N,V,S,H	1
SBIW	Rd, K	Subtract Immediate from Word	Rd + 1:Rd ← Rd + 1:Rd - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND	Rd ← Rd • Rr	Z,N,V,S	1
ANDI	Rd, K	Logical AND with Immediate	Rd ← Rd • K	Z,N,V,S	1
OR	Rd, Rr	Logical OR	Rd ← Rd v Rr	Z,N,V,S	1
ORI	Rd, K	Logical OR with Immediate	Rd ← Rd v K	Z,N,V,S	1
EOR	Rd, Rr	Exclusive OR	Rd ← Rd ⊕ Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd ← $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd ← $00 - Rd	Z,C,N,V,S,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V,S	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FFh - K)	Z,N,V,S	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement	Rd ← Rd - 1	Z,N,V,S	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V,S	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V,S	1
SER	Rd	Set Register	Rd ← $FF	None	1
MUL	Rd,Rr	Multiply Unsigned	R1:R0 ← Rd x Rr (UU)	Z,C	2
MULS	Rd,Rr	Multiply Signed	R1:R0 ← Rd x Rr (SS)	Z,C	2
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr (SU)	Z,C	2
FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0 ← Rd x Rr<<1 (UU)	Z,C	2
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0 ← Rd x Rr<<1 (SS)	Z,C	2
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← Rd x Rr<<1 (SU)	Z,C	2
DES	K	Data Encryption	if (H = 0) then R15:R0 ← Encrypt(R15:R0, K) else if (H = 1) then R15:R0 ← Decrypt(R15:R0, K)		1/2
Branch instructions					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC(15:0) ← Z, PC(21:16) ← 0	None	2
EIJMP		Extended Indirect Jump to (Z)	PC(15:0) ← Z, PC(21:16) ← EIND	None	2
JMP	k	Jump	PC ← k	None	3
RCALL	k	Relative Call Subroutine	PC ← PC + k + 1	None	2 / 3 (1)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ICALL		Indirect Call to (Z)	PC(15:0) ← Z, PC(21:16) ← 0	None	2 / 3 (1)
EICALL		Extended Indirect Call to (Z)	PC(15:0) ← Z, PC(21:16) ← EIND	None	3 (1)
CALL	k	call Subroutine	PC ← k	None	3 / 4 (1)
RET		Subroutine Return	PC ← STACK	None	4 / 5 (1)
RETI		Interrupt Return	PC ← STACK	I	4 / 5 (1)
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	Rd - Rr	Z,C,N,V,S,H	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,S,H	1
CPI	Rd,K	Compare with Immediate	Rd - K	Z,C,N,V,S,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b) = 0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBRS	Rr, b	Skip if Bit in Register Set	if (Rr(b) = 1) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIC	A, b	Skip if Bit in I/O Register Cleared	if (I/O(A,b) = 0) PC ← PC + 2 or 3	None	2 / 3 / 4
SBIS	A, b	Skip if Bit in I/O Register Set	If (I/O(A,b) =1) PC ← PC + 2 or 3	None	2 / 3 / 4
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, 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
Data transfer instructions					
MOV	Rd, Rr	Copy Register	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Pair	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1

Mnemonics	Operands	Description	Operation	Flags	#Clocks
LDS	Rd, k	Load Direct from data space	Rd ← (k)	None	2 (1)(2)
LD	Rd, X	Load Indirect	Rd ← (X)	None	1 (1)(2)
LD	Rd, X+	Load Indirect and Post-Increment	Rd ← (X) X ← X + 1	None	1 (1)(2)
LD	Rd, -X	Load Indirect and Pre-Decrement	X ← X - 1, Rd ← (X) ← (X)	None	2 (1)(2)
LD	Rd, Y	Load Indirect	Rd ← (Y) ← (Y)	None	1 (1)(2)
LD	Rd, Y+	Load Indirect and Post-Increment	Rd ← (Y) Y ← Y + 1	None	1 (1)(2)
LD	Rd, -Y	Load Indirect and Pre-Decrement	Y ← Y - 1 Rd ← (Y)	None	2 (1)(2)
LDD	Rd, Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2 (1)(2)
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	1 (1)(2)
LD	Rd, Z+	Load Indirect and Post-Increment	Rd ← (Z), Z ← Z+1	None	1 (1)(2)
LD	Rd, -Z	Load Indirect and Pre-Decrement	Z ← Z - 1, Rd ← (Z)	None	2 (1)(2)
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2 (1)(2)
STS	k, Rr	Store Direct to Data Space	(k) ← Rd	None	2 (1)
ST	X, Rr	Store Indirect	(X) ← Rr	None	1 (1)
ST	X+, Rr	Store Indirect and Post-Increment	(X) ← Rr, X ← X + 1	None	1 (1)
ST	-X, Rr	Store Indirect and Pre-Decrement	X ← X - 1, (X) ← Rr	None	2 (1)
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	1 (1)
ST	Y+, Rr	Store Indirect and Post-Increment	(Y) ← Rr, Y ← Y + 1	None	1 (1)
ST	-Y, Rr	Store Indirect and Pre-Decrement	Y ← Y - 1, (Y) ← Rr	None	2 (1)
STD	Y+q, Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2 (1)
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	1 (1)
ST	Z+, Rr	Store Indirect and Post-Increment	(Z) ← Rr Z ← Z + 1	None	1 (1)
ST	-Z, Rr	Store Indirect and Pre-Decrement	Z ← Z - 1	None	2 (1)
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2 (1)
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-Increment	Rd ← (Z), Z ← Z + 1	None	3
ELPM		Extended Load Program Memory	R0 ← (RAMPZ:Z)	None	3
ELPM	Rd, Z	Extended Load Program Memory	Rd ← (RAMPZ:Z)	None	3
ELPM	Rd, Z+	Extended Load Program Memory and Post-Increment	Rd ← (RAMPZ:Z), Z ← Z + 1	None	3
SPM		Store Program Memory	(RAMPZ:Z) ← R1:R0	None	-
SPM	Z+	Store Program Memory and Post-Increment by 2	(RAMPZ:Z) ← R1:R0, Z ← Z + 2	None	-

Mnemonics	Operands	Description	Operation	Flags	#Clocks
IN	Rd, A	In From I/O Location	Rd ← I/O(A)	None	1
OUT	A, Rr	Out To I/O Location	I/O(A) ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	1 (1)
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2 (1)
XCH	Z, Rd	Exchange RAM location	Temp ← Rd, Rd ← (Z), (Z) ← Temp	None	2
LAS	Z, Rd	Load and Set RAM location	Temp ← Rd, Rd ← (Z), (Z) ← Temp v (Z)	None	2
LAC	Z, Rd	Load and Clear RAM location	Temp ← Rd, Rd ← (Z), (Z) ← ($FFh – Rd) ⊡ (Z)	None	2
LAT	Z, Rd	Load and Toggle RAM location	Temp ← Rd, Rd ← (Z), (Z) ← Temp ⊕ (Z)	None	2
		Bit and bit-test instructions			
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0, C ← Rd(7)	Z,C,N,V,H	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0, C ← Rd(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,H	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)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
SBI	A, b	Set Bit in I/O Register	I/O(A, b) ← 1	None	1
CBI	A, b	Clear Bit in I/O Register	I/O(A, b) ← 0	None	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

Mnemonics	Operands	Description	Operation	Flags	#Clocks
SEV		Set Two's Complement Overflow	V ← 1	V	1
CLV		Clear Two's Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
MCU control instructions					
BREAK		Break	(See specific descr. for BREAK)	None	1
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR)	None	1

Notes: 1. Cycle times for Data memory accesses assume internal memory accesses, and are not valid for accesses via the external RAM interface.
2. One extra cycle must be added when accessing Internal SRAM.

37.1.15 SPI Characteristics

Figure 37-5. SPI timing requirements in master mode.

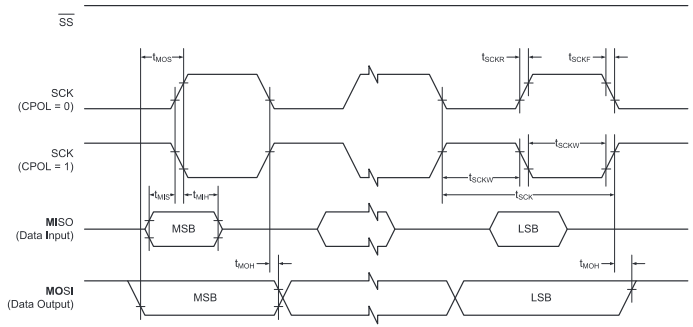


Figure 37-6. SPI timing requirements in slave mode.

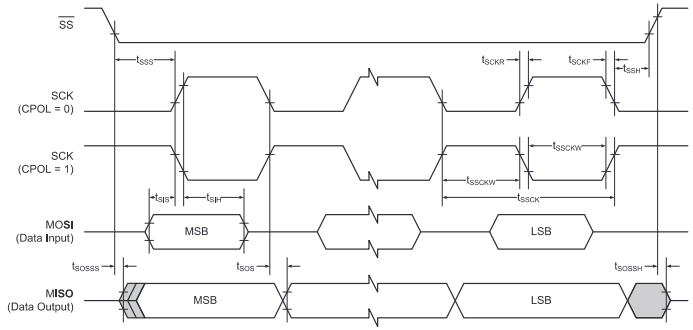


Table 37-31. SPI timing characteristics and requirements.

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
t_{SCK}	SCK period	Master		(See Table 21-4 in XMEGA AU Manual)		ns
t_{SCKW}	SCK high/low width	Master		0.5*SCK		
t_{SCKR}	SCK rise time	Master		2.7		
t_{SCKF}	SCK fall time	Master		2.7		
t_{MIS}	MISO setup to SCK	Master		11		
t_{MIH}	MISO hold after SCK	Master		0		
t_{MOS}	MOSI setup SCK	Master		0.5*SCK		
t_{MOH}	MOSI hold after SCK	Master		1		
	SCK to out high	Master		1		
t_{SSCK}	Slave SCK period	Slave	$4*t_{ClkPER}$			
t_{SSCKW}	SCK high/low width	Slave	$2*t_{ClkPER}$			
t_{SSCKR}	SCK rise time	Slave			1600	
t_{SSCKF}	SCK fall time	Slave			1600	
t_{SIS}	MOSI setup to SCK	Slave	3			
t_{SIH}	MOSI hold after SCK	Slave	t_{SCK}			
t_{SSS}	$\overline{SS}$ setup to SCK	Slave	20			
t_{SSH}	$\overline{SS}$ hold after SCK	Slave	20			
t_{SOS}	MISO setup SCK	Slave		8		
t_{SOH}	MISO hold after SCK	Slave		13		
t_{SOSS}	MISO setup after $\overline{SS}$ low	Slave		11		
t_{SOSH}	MISO hold after $\overline{SS}$ high	Slave		8		

AVR Microcontrollers

AVR Instruction Set Manual

OTHER

Instruction Set Nomenclature

Status Register (SREG)

SREG	Status Register
C	Carry Flag
Z	Zero Flag
N	Negative Flag
V	Two's complement overflow indicator
S	N ⊕ V, for signed tests
H	Half Carry Flag
T	Transfer bit used by BLD and BST instructions
I	Global Interrupt Enable/Disable Flag

Registers and Operands

Rd:	Destination (and source) register in the Register File
Rr:	Source register in the Register File
R:	Result after instruction is executed
K:	Constant data
k:	Constant address
b:	Bit in the Register File or I/O Register (3-bit)
s:	Bit in the Status Register (3-bit)
X,Y,Z:	Indirect Address Register (X=R27:R26, Y=R29:R28, and Z=R31:R30)
A:	I/O location address
q:	Displacement for direct addressing (6-bit)

Table of Contents

1. I/O Registers

1.1. RAMPX, RAMPY, and RAMPZ

Registers concatenated with the X-, Y-, and Z-registers enabling indirect addressing of the whole data space on MCUs with more than 64KB data space, and constant data fetch on MCUs with more than 64KB program space.

1.2. RAMPD

Register concatenated with the Z-register enabling direct addressing of the whole data space on MCUs with more than 64KB data space.

1.3. EIND

Register concatenated with the Z-register enabling indirect jump and call to the whole program space on MCUs with more than 64K words (128KB) program space.

1.4. Stack

STACK	Stack for return address and pushed registers
SP	Stack Pointer to STACK

1.5. Flags

⇔	Flag affected by instruction
0	Flag cleared by instruction
1	Flag set by instruction
-	Flag not affected by instruction

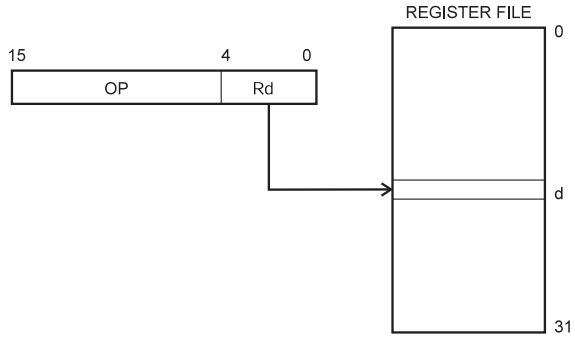
2. The Program and Data Addressing Modes

The AVR® Enhanced RISC microcontroller supports powerful and efficient addressing modes for access to the Program memory (Flash) and Data memory (SRAM, Register file, I/O Memory, and Extended I/O Memory). This chapter describes the various addressing modes supported by the AVR architecture. In the following figures, OP means the operation code part of the instruction word. To simplify, not all figures show the exact location of the addressing bits. To generalize, the abstract terms RAMEND and FLASHEND have been used to represent the highest location in data and program space, respectively.

Note: Not all addressing modes are present in all devices. Refer to the device specific instruction summary.

2.1. Register Direct, Single Register Rd

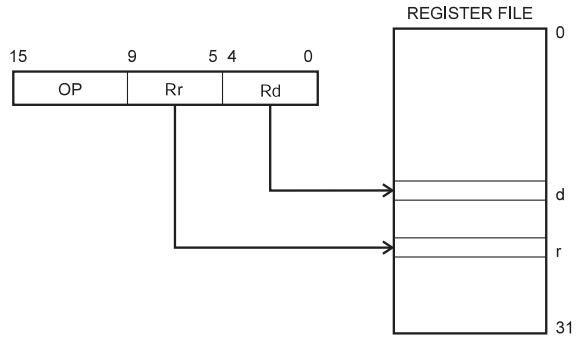
Figure 2-1. Direct Single Register Addressing



The operand is contained in register d (Rd).

2.2. Register Direct - Two Registers, Rd and Rr

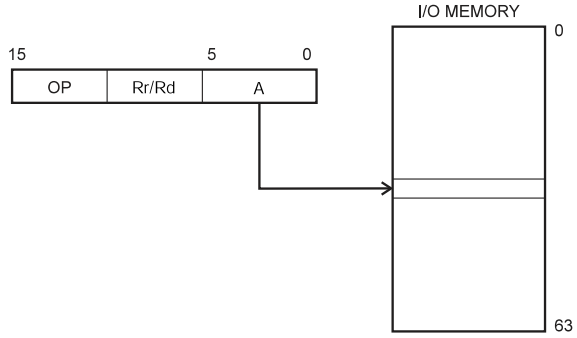
Figure 2-2. Direct Register Addressing, Two Registers



Operands are contained in register r (Rr) and d (Rd). The result is stored in register d (Rd).

2.3. I/O Direct

Figure 2-3. I/O Direct Addressing

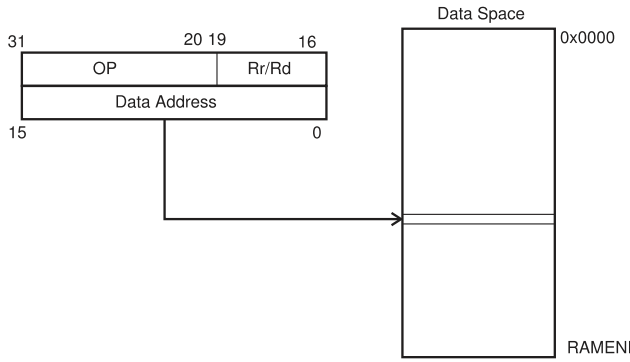


Operand address is contained in six bits of the instruction word. n is the destination or source register address.

Note: Some complex AVR Microcontrollers have more peripheral units than can be supported within the 64 locations reserved in the opcode for I/O direct addressing. The extended I/O memory from address 64 to 255 can only be reached by data addressing, not I/O addressing.

2.4. Data Direct

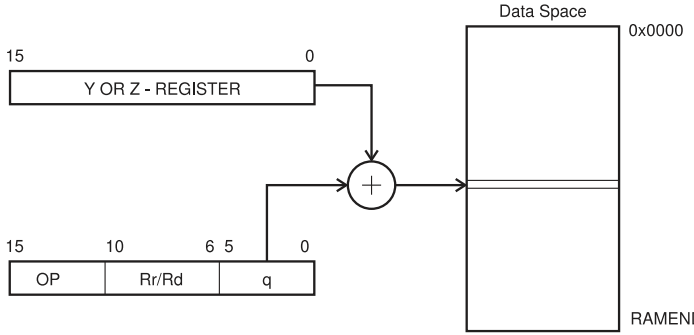
Figure 2-4. Direct Data Addressing



A 16-bit Data Address is contained in the 16 LSBs of a two-word instruction. Rd/Rr specify the destination or source register.

2.5. Data Indirect with Displacement

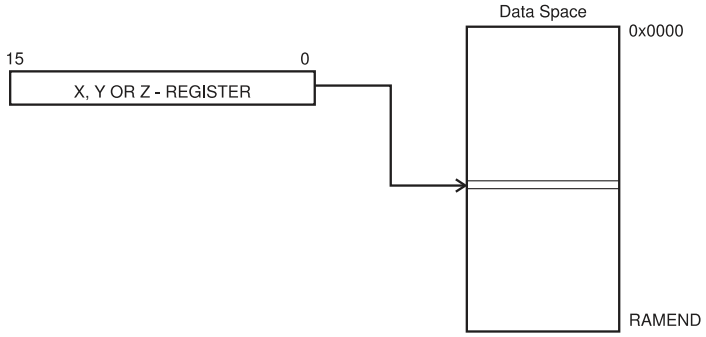
Figure 2-5. Data Indirect with Displacement



Operand address is the result of the Y- or Z-register contents added to the address contained in six bits of the instruction word. Rd/Rr specify the destination or source register.

2.6. Data Indirect

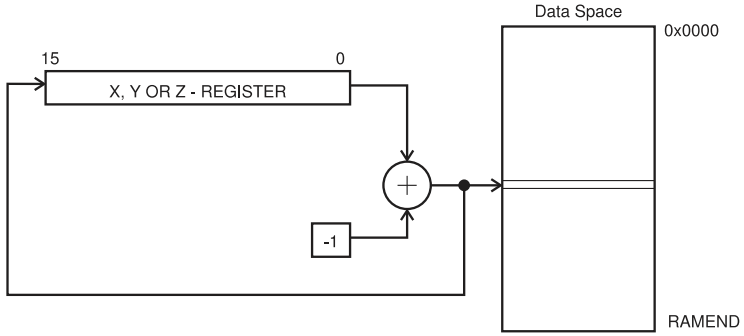
Figure 2-6. Data Indirect Addressing



Operand address is the contents of the X-, Y-, or the Z-register. In AVR devices without SRAM, Data Indirect Addressing is called Register Indirect Addressing. Register Indirect Addressing is a subset of Data Indirect Addressing since the data space form 0 to 31 is the Register File.

2.7. Data Indirect with Pre-decrement

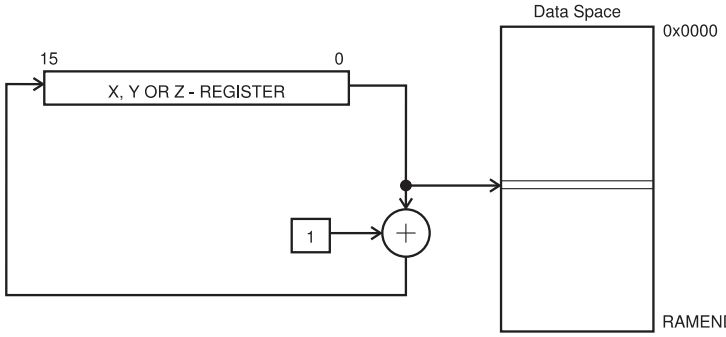
Figure 2-7. Data Indirect Addressing with Pre-decrement



The X,- Y-, or the Z-register is decremented before the operation. Operand address is the decremented contents of the X-, Y-, or the Z-register.

2.8. Data Indirect with Post-increment

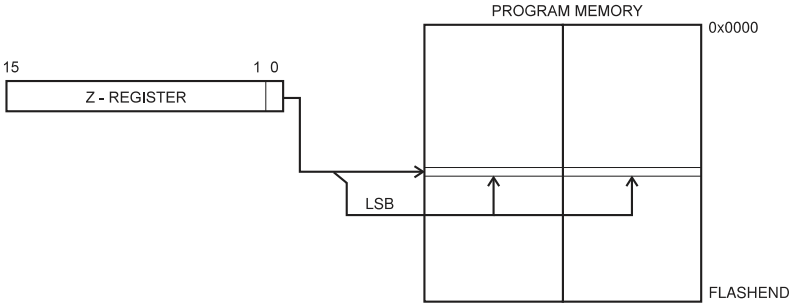
Figure 2-8. Data Indirect Addressing with Post-increment



The X-, Y-, or the Z-register is incremented after the operation. Operand address is the content of the X-, Y-, or the Z-register prior to incrementing.

2.9. Program Memory Constant Addressing using the LPM, ELPM, and SPM Instructions

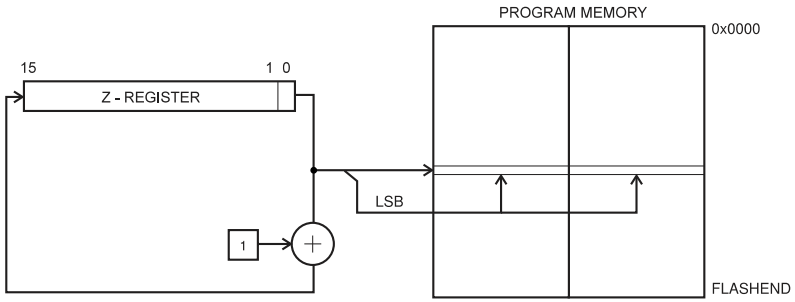
Figure 2-9. Program Memory Constant Addressing



Constant byte address is specified by the Z-register contents. The 15 MSBs select word address. For LPM, the LSB selects low byte if cleared (LSB = 0) or high byte if set (LSB = 1). For SPM, the LSB should be cleared. If ELPM is used, the RAMPZ Register is used to extend the Z-register.

2.10. Program Memory with Post-increment using the LPM Z+ and ELPM Z+ Instruction

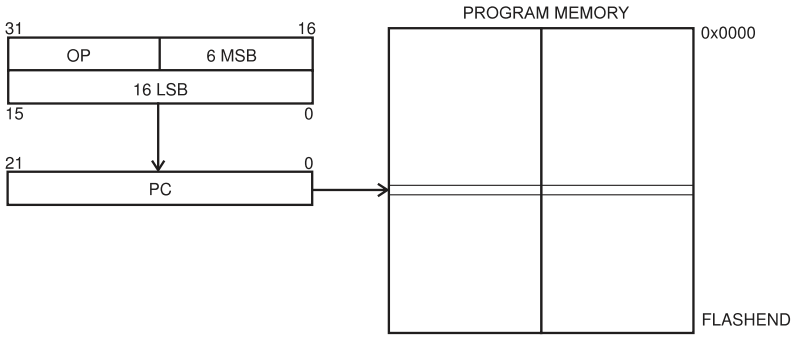
Figure 2-10. Program Memory Addressing with Post-increment



Constant byte address is specified by the Z-register contents. The 15 MSBs select word address. The LSB selects low byte if cleared (LSB = 0) or high byte if set (LSB = 1). If ELPM Z+ is used, the RAMPZ Register is used to extend the Z-register.

2.11. Direct Program Addressing, JMP and CALL

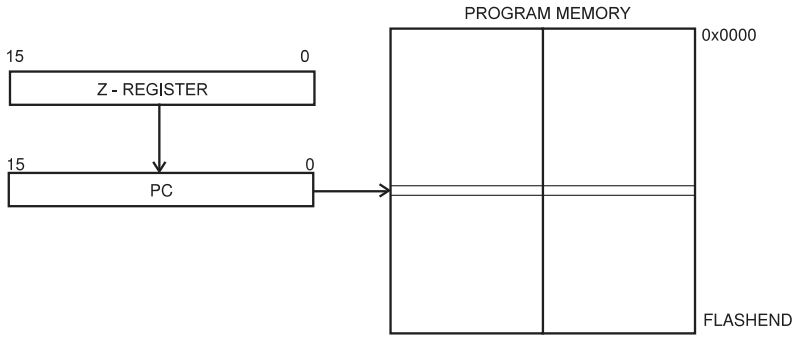
Figure 2-11. Direct Program Memory Addressing



Program execution continues at the address immediate in the instruction word.

2.12. Indirect Program Addressing, IJMP and ICALL

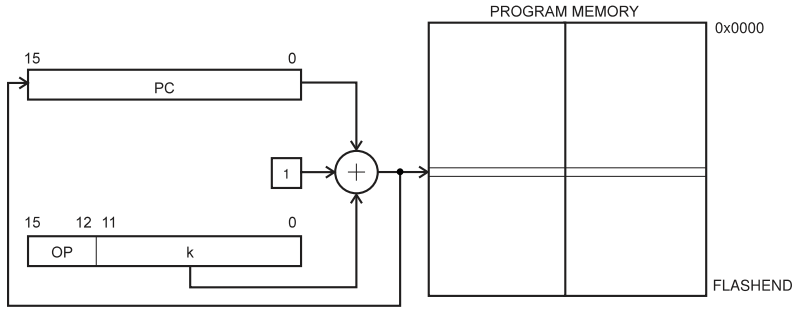
Figure 2-12. Indirect Program Memory Addressing



Program execution continues at address contained by the Z-register (i.e., the PC is loaded with the contents of the Z-register).

2.13. Relative Program Addressing, RJMP and RCALL

Figure 2-13. Relative Program Memory Addressing



Program execution continues at address PC + k + 1. The relative address k is from -2048 to 2047.

3. Conditional Branch Summary

Test	Boolean	Mnemonic	Complementary	Boolean	Mnemonic	Comment
Rd > Rr	Z•(N ⊕ V) = 0	BRLT[1]	Rd ≤ Rr	Z+(N ⊕ V) = 1	BRGE*	Signed
Rd ≥ Rr	(N ⊕ V) = 0	BRGE	Rd < Rr	(N ⊕ V) = 1	BRLT	Signed
Rd = Rr	Z = 1	BREQ	Rd ≠ Rr	Z = 0	BRNE	Signed
Rd ≤ Rr	Z+(N ⊕ V) = 1	BRGE[1]	Rd > Rr	Z•(N ⊕ V) = 0	BRLT*	Signed
Rd < Rr	(N ⊕ V) = 1	BRLT	Rd ≥ Rr	(N ⊕ V) = 0	BRGE	Signed
Rd > Rr	C + Z = 0	BRLO[1]	Rd ≤ Rr	C + Z = 1	BRSH*	Unsigned
Rd ≥ Rr	C = 0	BRSH/ BRCC	Rd < Rr	C = 1	BRLO/BRCS	Unsigned
Rd = Rr	Z = 1	BREQ	Rd ≠ Rr	Z = 0	BRNE	Unsigned
Rd ≤ Rr	C + Z = 1	BRSH[1]	Rd > Rr	C + Z = 0	BRLO*	Unsigned
Rd < Rr	C = 1	BRLO/BRCS	Rd ≥ Rr	C = 0	BRSH/BRCC	Unsigned
Carry	C = 1	BRCS	No carry	C = 0	BRCC	Simple
Negative	N = 1	BRMI	Positive	N = 0	BRPL	Simple
Overflow	V = 1	BRVS	No overflow	V = 0	BRVC	Simple
Zero	Z = 1	BREQ	Not zero	Z = 0	BRNE	Simple

Note: 1. Interchange Rd and Rr in the operation before the test, i.e., CP Rd,Rr → CP Rr,Rd.

4. Instruction Set Summary

Several updates of the AVR CPU during its lifetime has resulted in different flavors of the instruction set, especially for the timing of the instructions. Machine code level of compatibility is intact for all CPU versions with a very few exceptions related to the Reduced Core (AVRrc), though not all instructions are included in the instruction set for all devices. The table below contains the major versions of the AVR 8-bit CPUs. In addition to the different versions, there are differences dependent of the size of the device memory map. Typically these differences are handled by a C/EC++ compiler, but users that are porting code should be aware that the code execution can vary slightly in number of clock cycles.

Table 4-1. Versions of AVR 8-bit CPU

Name	Device Series	Description
AVR	AT90	Original instruction set from 1995.
AVRe	megaAVR®	Multiply (xMULxx), Move Word (MOVW), and enhanced Load Program Memory (LPM) added to the AVR instruction set. No timing differences.
AVRe	tinyAVR®	Multiply not included, but else equal to AVRe for megaAVR.
AVRxm	XMEGA®	Significantly different timing compared to AVR(e). The Read Modify Write (RMW) and DES encryption instructions are unique to this version.
AVRxt	(AVR)	AVR 2016 and onwards. This variant is based on AVRe and AVRxm. Closer related to AVRe, but with improved timing.
AVRrc	tinyAVR	The Reduced Core AVR CPU was developed for ultra-low pinout (6-pin) size constrained devices. The AVRrc therefore only has a 16 registers register-file (R31-R16) and a limited instruction set.

Table 4-2. Arithmetic and Logic Instructions

Mnemonic	Operands	Description		Op		Flags	#Clocks AVR	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
ADD	Rd, Rr	Add without Carry	Rd	←	Rd + Rr	Z,C,N,V,S,H	1	1	1	1
ADC	Rd, Rr	Add with Carry	Rd	←	Rd + Rr + C	Z,C,N,V,S,H	1	1	1	1
ADIW	Rd, K	Add Immediate to Word	Rd	←	Rd + 1:Rd + K	Z,C,N,V,S	2	2	2	N/A
SUB	Rd, Rr	Subtract without Carry	Rd	←	Rd - Rr	Z,C,N,V,S,H	1	1	1	1
SUBI	Rd, K	Subtract Immediate	Rd	←	Rd - K	Z,C,N,V,S,H	1	1	1	1
SBC	Rd, Rr	Subtract with Carry	Rd	←	Rd - Rr - C	Z,C,N,V,S,H	1	1	1	1
SBCI	Rd, K	Subtract Immediate with Carry	Rd	←	Rd - K - C	Z,C,N,V,S,H	1	1	1	1
SBIW	Rd, K	Subtract Immediate from Word	Rd + 1:Rd	←	Rd + 1:Rd - K	Z,C,N,V,S	2	2	2	N/A
AND	Rd, Rr	Logical AND	Rd	←	Rd • Rr	Z,N,V,S	1	1	1	1

Mnemonic	Operands	Description		Op		Flags	#Clocks AVR	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
ANDI	Rd, K	Logical AND with Immediate	Rd	←	Rd • K	Z,N,V,S	1	1	1	1
OR	Rd, Rr	Logical OR	Rd	←	Rd v Rr	Z,N,V,S	1	1	1	1
ORI	Rd, K	Logical OR with Immediate	Rd	←	Rd v K	Z,N,V,S	1	1	1	1
EOR	Rd, Rr	Exclusive OR	Rd	←	Rd ⊕ Rr	Z,N,V,S	1	1	1	1
COM	Rd	One's Complement	Rd	←	$FF - Rd	Z,C,N,V,S	1	1	1	1
NEG	Rd	Two's Complement	Rd	←	$00 - Rd	Z,C,N,V,S,H	1	1	1	1
SBR	Rd,K	Set Bit(s) in Register	Rd	←	Rd v K	Z,N,V,S	1	1	1	1
CBR	Rd,K	Clear Bit(s) in Register	Rd	←	Rd • ($FFh - K)	Z,N,V,S	1	1	1	1
INC	Rd	Increment	Rd	←	Rd + 1	Z,N,V,S	1	1	1	1
DEC	Rd	Decrement	Rd	←	Rd - 1	Z,N,V,S	1	1	1	1
TST	Rd	Test for Zero or Minus	Rd	←	Rd • Rd	Z,N,V,S	1	1	1	1
CLR	Rd	Clear Register	Rd	←	Rd ⊕ Rd	Z,N,V,S	1	1	1	1
SER	Rd	Set Register	Rd	←	$FF	None	1	1	1	1
MUL	Rd,Rr	Multiply Unsigned	R1:R0	←	Rd x Rr (UU)	Z,C	2	2	2	N/A
MULS	Rd,Rr	Multiply Signed	R1:R0	←	Rd x Rr (SS)	Z,C	2	2	2	N/A
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0	←	Rd x Rr (SU)	Z,C	2	2	2	N/A
FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0	←	Rd x Rr<<1 (UU)	Z,C	2	2	2	N/A
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0	←	Rd x Rr<<1 (SS)	Z,C	2	2	2	N/A
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0	←	Rd x Rr<<1 (SU)	Z,C	2	2	2	N/A
DES	K	Data Encryption	if (H = 0) then R15:R0 else if (H = 1) then R15:R0	← ←	Encrypt(R15:R0, K) Decrypt(R15:R0, K)		N/A	1/2	N/A	N/A

Table 4-3. Branch Instructions

Mnemonic	Operands	Description		Op		Flags	#Clocks AVR	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
RJMP	k	Relative Jump	PC	←	PC + k + 1	None	2	2	2	2
IJMP		Indirect Jump to (Z)	PC(15:0) PC(21:16)	← ←	Z 0	None	2	2	2	2

Mnemonic	Operands	Description		Op		Flags	#Clocks AVR	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
EIJMP		Extended Indirect Jump to (Z)	PC(15:0) PC(21:16)	← ←	Z EIND	None	2	2	2	N/A
JMP	k	Jump	PC	←	k	None	3	3	3	N/A
RCALL	k	Relative Call Subroutine	PC	←	PC + k + 1	None	3 / 4(1)	2 / 3(1)	2 / 3	3(1)
ICALL		Indirect Call to (Z)	PC(15:0) PC(21:16)	← ←	Z 0	None	3 / 4(1)	2 / 3(1)	2 / 3	3(1)
EICALL		Extended Indirect Call to (Z)	PC(15:0) PC(21:16)	← ←	Z EIND	None	4(1)	3(1)	2 / 3	N/A
CALL	k	Call Subroutine	PC	←	k	None	4 / 5(1)	3 / 4(1)	3 / 4	N/A
RET		Subroutine Return	PC	←	STACK	None	4 / 5(1)	4 / 5(1)	4 / 5	6(1)
RETI		Interrupt Return	PC	←	STACK	I	4 / 5(1)	4 / 5(1)	4 / 5	6(1)
CPSE	Rd,Rr	Compare, skip if Equal	if (Rd = Rr) PC	←	PC + 2 or 3	None	1 / 2 / 3	1 / 2 / 3	1 / 2 / 3	1 / 2
CP	Rd,Rr	Compare	Rd - Rr			Z,C,N,V,S,H	1	1	1	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C			Z,C,N,V,S,H	1	1	1	1
CPI	Rd,K	Compare with Immediate	Rd - K			Z,C,N,V,S,H	1	1	1	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b) = 0) PC	←	PC + 2 or 3	None	1 / 2 / 3	1 / 2 / 3	1 / 2 / 3	1 / 2
SBRS	Rr, b	Skip if Bit in Register Set	if (Rr(b) = 1) PC	←	PC + 2 or 3	None	1 / 2 / 3	1 / 2 / 3	1 / 2 / 3	1 / 2
SBIC	A, b	Skip if Bit in I/O Register Cleared	if (I/O(A,b) = 0) PC	←	PC + 2 or 3	None	1 / 2 / 3	2 / 3 / 4	1 / 2 / 3	1 / 2
SBIS	A, b	Skip if Bit in I/O Register Set	If (I/O(A,b) =1) PC	←	PC + 2 or 3	None	1 / 2 / 3	2 / 3 / 4	1 / 2 / 3	1 / 2
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2

Mnemonic	Operands	Description		Op		Flags	#Clocks AVR	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
BRLO	k	Branch if Lower	if (C = 1) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRLT	k	Branch if Less Than, Signed	if (N ⊕ V= 1) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC	←	PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2

Table 4-4. Data Transfer Instructions

Mnemonic	Operands	Description		Op		Flags	#Clocks AVR	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
MOV	Rd, Rr	Copy Register	Rd	←	Rr	None	1	1	1	1
MOVW	Rd, Rr	Copy Register Pair	Rd+1:Rd	←	Rr+1:Rr	None	1	1	1	N/A
LDI	Rd, K	Load Immediate	Rd	←	K	None	1	1	1	1
LDS	Rd, k	Load Direct from data space	Rd	←	(k)	None	2[1]	2[1]	3[1]	2
LD	Rd, X	Load Indirect	Rd	←	(X)	None	2[1]	1[1]	2[1]	1 / 2
LD	Rd, X+	Load Indirect and Post-Increment	Rd X	← ←	(X) X + 1	None	2[1]	1[1]	2[1]	2 / 3
LD	Rd, -X	Load Indirect and Pre-Decrement	X Rd	← ←	X - 1 (X)	None	2[1]	2[1]	2[1]	2 / 3
LD	Rd, Y	Load Indirect	Rd	←	(Y)	None	2[1]	1[1]	2[1]	1 / 2

Mnemonic	Operands	Description		Op		Flags	#Clocks AVR	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
LD	Rd, Y+	Load Indirect and Post-Increment	Rd Y	← ←	(Y) Y + 1	None	2[1]	1[1]	2[1]	2 / 3
LD	Rd, -Y	Load Indirect and Pre-Decrement	Y Rd	← ←	Y - 1 (Y)	None	2[1]	2[1]	2[1]	2 / 3
LDD	Rd, Y+q	Load Indirect with Displacement	Rd	←	(Y + q)	None	2[1]	2[1]	2[1]	N/A
LD	Rd, Z	Load Indirect	Rd	←	(Z)	None	2[1]	1[1]	2[1]	1 / 2
LD	Rd, Z+	Load Indirect and Post-Increment	Rd Z	← ←	(Z) Z+1	None	2[1]	1[1]	2[1]	2 / 3
LD	Rd, -Z	Load Indirect and Pre-Decrement	Z Rd	← ←	Z - 1 (Z)	None	2[1]	2[1]	2[1]	2 / 3
LDD	Rd, Z+q	Load Indirect with Displacement	Rd	←	(Z + q)	None	2[1]	2[1]	2[1]	N/A
STS	k, Rr	Store Direct to Data Space	(k)	←	Rd	None	2[1][2]	2[1][2]	2[1][2]	1
ST	X, Rr	Store Indirect	(X)	←	Rr	None	1[1][2]	1[1][2]	1[1][2]	1
ST	X+, Rr	Store Indirect and Post-Increment	(X) X	← ←	Rr X + 1	None	1[1][2]	1[1][2]	1[1][2]	1
ST	-X, Rr	Store Indirect and Pre-Decrement	X (X)	← ←	X - 1 Rr	None	2[1][2]	2[1][2]	1[1][2]	2
ST	Y, Rr	Store Indirect	(Y)	←	Rr	None	2[1][2]	1[1][2]	1[1][2]	1
ST	Y+, Rr	Store Indirect and Post-Increment	(Y) Y	← ←	Rr Y + 1	None	2[1][2]	1[1][2]	1[1][2]	1
ST	-Y, Rr	Store Indirect and Pre-Decrement	Y (Y)	← ←	Y - 1 Rr	None	2[1][2]	2[1][2]	1[1][2]	2
STD	Y+q, Rr	Store Indirect with Displacement	(Y + q)	←	Rr	None	2[1][2]	2[1][2]	1[1][2]	N/A
ST	Z, Rr	Store Indirect	(Z)	←	Rr	None	2[1][2]	1[1][2]	1[1][2]	1
ST	Z+, Rr	Store Indirect and Post-Increment	(Z) Z	← ←	Rr Z + 1	None	2[1][2]	1[1][2]	1[1][2]	1
ST	-Z, Rr	Store Indirect and Pre-Decrement	Z	←	Z - 1	None	2[1][2]	2[1][2]	1[1][2]	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q)	←	Rr	None	2[1][2]	2[1][2]	1[1][2]	N/A
LPM		Load Program Memory	R0	←	(Z)	None	3	3	3	N/A
LPM	Rd, Z	Load Program Memory	Rd	←	(Z)	None	3	3	3	N/A

Mnemonic	Operands	Description		Op		Flags	#Clocks AVR	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
LPM	Rd, Z+	Load Program Memory and Post-Increment	Rd Z	← ←	(Z) Z + 1	None	3	3	3	N/A
ELPM		Extended Load Program Memory	R0	←	(RAMPZ:Z)	None	3	3	3	N/A
ELPM	Rd, Z	Extended Load Program Memory	Rd	←	(RAMPZ:Z)	None	3	3	3	N/A
ELPM	Rd, Z+	Extended Load Program Memory and Post-Increment	Rd (RAMPZ:Z)	← ←	(RAMPZ:Z) (RAMPZ:Z) + 1	None	3	3	3	N/A
SPM		Store Program Memory	(RAMPZ:Z)	←	R1:R0	None	(4)	(4)	4(3)	N/A
SPM	Z+	Store Program Memory and Post-Increment by 2	(RAMPZ:Z) Z	← ←	R1:R0 Z + 2	None	(4)	(4)	4(3)	N/A
IN	Rd, A	In From I/O Location	Rd	←	I/O(A)	None	1	1	1	1
OUT	A, Rr	Out To I/O Location	I/O(A)	←	Rr	None	1	1	1	1
PUSH	Rr	Push Register on Stack	STACK	←	Rr	None	2	1(1)	1	1(1)
POP	Rd	Pop Register from Stack	Rd	←	STACK	None	2	2(1)	2	3(1)
XCH	Z, Rd	Exchange	(Z) Rd	← ←	Rd (Z)	None	N/A	1	N/A	N/A
LAS	Z, Rd	Load and Set	(Z) Rd	← ←	Rd v (Z) (Z)	None	N/A	1	N/A	N/A
LAC	Z, Rd	Load and Clear	(Z) Rd	← ←	($FF – Rd) • (Z) (Z)	None	N/A	1	N/A	N/A
LAT	Z, Rd	Load and Toggle	(Z) Rd	← ←	Rd ⊕ (Z) (Z)	None	N/A	1	N/A	N/A

Table 4-5. Bit and Bit-test Instructions

Mnemonic	Operands	Description		Op		Flags	#Clocks AVR	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
LSL	Rd	Logical Shift Left	Rd(n+1) Rd(0) C	← ← ←	Rd(n) 0 Rd(7)	Z,C,N,V,H	1	1	1	1
LSR	Rd	Logical Shift Right	Rd(n) Rd(7) C	← ← ←	Rd(n+1) 0 Rd(0)	Z,C,N,V	1	1	1	1

Mnemonic	Operands	Description		Op		Flags	#Clocks AVR	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
ROL	Rd	Rotate Left Through Carry	Rd(0) Rd(n+1) C	← ← ←	C Rd(n) Rd(7)	Z,C,N,V,H	1	1	1	1
ROR	Rd	Rotate Right Through Carry	Rd(7) Rd(n) C	← ← ←	C Rd(n+1) Rd(0)	Z,C,N,V	1	1	1	1
ASR	Rd	Arithmetic Shift Right	Rd(n)	←	Rd(n+1), n=0..6	Z,C,N,V	1	1	1	1
SWAP	Rd	Swap Nibbles	Rd(3..0)	↔	Rd(7..4)	None	1	1	1	1
SBI	A, b	Set Bit in I/O Register	I/O(A, b)	←	1	None	2	1	1	1
CBI	A, b	Clear Bit in I/O Register	I/O(A, b)	←	0	None	2	1	1	1
BST	Rr, b	Bit Store from Register to T	T	←	Rr(b)	T	1	1	1	1
BLD	Rd, b	Bit load from T to Register	Rd(b)	←	T	None	1	1	1	1
BSET	s	Flag Set	SREG(s)	←	1	SREG(s)	1	1	1	1
BCLR	s	Flag Clear	SREG(s)	←	0	SREG(s)	1	1	1	1
SEC		Set Carry	C	←	1	C	1	1	1	1
CLC		Clear Carry	C	←	0	C	1	1	1	1
SEN		Set Negative Flag	N	←	1	N	1	1	1	1
CLN		Clear Negative Flag	N	←	0	N	1	1	1	1
SEZ		Set Zero Flag	Z	←	1	Z	1	1	1	1
CLZ		Clear Zero Flag	Z	←	0	Z	1	1	1	1
SEI		Global Interrupt Enable	I	←	1	I	1	1	1	1
CLI		Global Interrupt Disable	I	←	0	I	1	1	1	1
SES		Set Signed Test Flag	S	←	1	S	1	1	1	1
CLS		Clear Signed Test Flag	S	←	0	S	1	1	1	1
SEV		Set Two's Complement Overflow	V	←	1	V	1	1	1	1
CLV		Clear Two's Complement Overflow	V	←	0	V	1	1	1	1
SET		Set T in SREG	T	←	1	T	1	1	1	1
CLT		Clear T in SREG	T	←	0	T	1	1	1	1

Mnemonic	Operands	Description		Op		Flags	#Clocks AVR	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
SEH		Set Half Carry Flag in SREG	H	←	1	H	1	1	1	1
CLH		Clear Half Carry Flag in SREG	H	←	0	H	1	1	1	1

Table 4-6. MCU Control Instructions

Mnemonic	Operands	Description	Operation	Flags	#Clocks AVR	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
BREAK		Break	(See also in Debug interface description)	None	1	1	1	1
NOP		No Operation		None	1	1	1	1
SLEEP		Sleep	(see also power management and sleep description)	None	1	1	1	1
WDR		Watchdog Reset	(see also Watchdog Controller description)	None	1	1	1	1

Note:

1. Cycle time for data memory accesses assume internal RAM access, and are not valid for accesses through the NVM controller. A minimum of one extra cycle must be added when accessing memory through the NVM controller (such as Flash and EEPROM), but depending on simultaneous accesses by other masters or the NVM controller state, there may be more than one extra cycle.
2. One extra cycle must be added when accessing lower (64 bytes of) I/O space.
3. The instruction is not available on all devices.
4. Device dependent. See the device specific datasheet.

5. ADC – Add with Carry

5.1. Description

Adds two registers and the contents of the C Flag and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd + Rr + C

Syntax:	Operands:	Program Counter:
(i) ADC Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0001	11rd	dddd	rrrr

5.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H $Rd3 \cdot Rr3 + Rr3 \cdot \overline{R3} + \overline{R3} \cdot Rd3$

Set if there was a carry from bit 3; cleared otherwise.

S N ⊕ V, for signed tests.

V $Rd7 \cdot Rr7 \cdot \overline{R7} + \overline{Rd7} \cdot \overline{Rr7} \cdot R7$

Set if two's complement overflow resulted from the operation; cleared otherwise.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Set if the result is $00; cleared otherwise.

C $Rd7 \cdot Rr7 + Rr7 \cdot \overline{R7} + \overline{R7} \cdot Rd7$

Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
            ; Add R1:R0 to R3:R2
add r2,r0   ; Add low byte
adc r3,r1   ; Add with carry high byte
```

Words 1 (2 bytes)

Cycles	1

6. ADD – Add without Carry

6.1. Description

Adds two registers without the C Flag and places the result in the destination register Rd.

	Operation:		
(i)	(i) Rd ← Rd + Rr		
	Syntax:	Operands:	Program Counter:
(i)	ADD Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0000	11rd	dddd	rrrr

6.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H $Rd3 \cdot Rr3 + Rr3 \cdot \overline{R3} + \overline{R3} \cdot Rd3$

Set if there was a carry from bit 3; cleared otherwise.

S $N \oplus V$, for signed tests.

V $Rd7 \cdot Rr7 \cdot \overline{R7} + \overline{Rd7} \cdot \overline{Rr7} \cdot R7$

Set if two's complement overflow resulted from the operation; cleared otherwise.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Set if the result is $00; cleared otherwise.

C $Rd7 \cdot Rr7 + Rr7 \cdot \overline{R7} + \overline{R7} \cdot Rd7$

Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
add r1,r2 ; Add r2 to r1 (r1=r1+r2)
add r28,r28 ; Add r28 to itself (r28=r28+r28)
```

Words	1 (2 bytes)
Cycles	1

7. ADIW – Add Immediate to Word

7.1. Description

Adds an immediate value (0 - 63) to a register pair and places the result in the register pair. This instruction operates on the upper four register pairs, and is well suited for operations on the pointer registers.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) Rd+1:Rd ← Rd+1:Rd + K

	Syntax:	Operands:	Program Counter:
(i)	ADIW Rd+1:Rd,K	d ∈ {24,26,28,30}, 0 ≤ K ≤ 63	PC ← PC + 1

16-bit Opcode:

1001	0110	KKdd	KKKK

7.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S N ⊕ V, for signed tests.

V $\overline{Rdh7}$ • R15

Set if two's complement overflow resulted from the operation; cleared otherwise.

N R15

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R15}$ • $\overline{R14}$ • $\overline{R13}$ • $\overline{R12}$ • $\overline{R11}$ • $\overline{R10}$ • $\overline{R9}$ • $\overline{R8}\overline{R7}$ • $\overline{R6}$ • $\overline{R5}$ • $\overline{R4}$ • $\overline{R3}$ • $\overline{R2}$ • $\overline{R1}$ • $\overline{R0}$

Set if the result is $0000; cleared otherwise.

C $\overline{R15}$ • Rdh7

Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rdh:Rdl after the operation (Rdh7-Rdh0 = R15-R8, Rdl7-Rdl0=R7-R0).

Example:

```
adiw r25:24,1 ; Add 1 to r25:r24
adiw ZH:ZL,63 ; Add 63 to the Z-pointer(r31:r30)
```

Words	1 (2 bytes)
Cycles	1

8. AND – Logical AND

8.1. Description

Performs the logical AND between the contents of register Rd and register Rr, and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd • Rr

Syntax:	Operands:	Program Counter:
(i) AND Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0010	00rd	dddd	rrrr

8.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S N ⊕ V, for signed tests.

V 0

Cleared.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
and r2,r3 ; Bitwise and r2 and r3, result in r2
ldi r16,1 ; Set bitmask 0000 0001 in r16
and r2,r16 ; Isolate bit 0 in r2
```

Words	1 (2 bytes)
Cycles	1

9. ANDI – Logical AND with Immediate

9.1. Description

Performs the logical AND between the contents of register Rd and a constant, and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd • K

Syntax:	Operands:	Program Counter:
(i) ANDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

0111	KKKK	dddd	KKKK

9.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S N ⊕ V, for signed tests.

V 0

Cleared.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
andi r17,$0F ; Clear upper nibble of r17
andi r18,$10 ; Isolate bit 4 in r18
andi r19,$AA ; Clear odd bits of r19
```

Words	1 (2 bytes)
Cycles	1

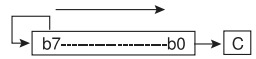
10. ASR – Arithmetic Shift Right

10.1. Description

Shifts all bits in Rd one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C Flag of the SREG. This operation effectively divides a signed value by two without changing its sign. The Carry Flag can be used to round the result.

Operation:

(i)



	Syntax:	Operands:	Program Counter:
(i)	ASR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0101

10.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S N ⊕ V, for signed tests.

V N ⊕ C, for N and C after the shift.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

C Rd0

Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
ldi r16,$10 ; Load decimal 16 into r16
asr r16 ; r16=r16 / 2
ldi r17,$FC ; Load -4 in r17
asr r17 ; r17=r17/2
```

Words	1 (2 bytes)
Cycles	1

11. BCLR – Bit Clear in SREG

11.1. Description

Clears a single Flag in SREG.

Operation:

(i) SREG(s) ← 0

	Syntax:	Operands:	Program Counter:
(i)	BCLR s	0 ≤ s ≤ 7	PC ← PC + 1

16-bit Opcode:

1001	0100	1sss	1000

11.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
⇔	⇔	⇔	⇔	⇔	⇔	⇔	⇔

I 0 if s = 7; Unchanged otherwise.

T 0 if s = 6; Unchanged otherwise.

H 0 if s = 5; Unchanged otherwise.

S 0 if s = 4; Unchanged otherwise.

V 0 if s = 3; Unchanged otherwise.

N 0 if s = 2; Unchanged otherwise.

Z 0 if s = 1; Unchanged otherwise.

C 0 if s = 0; Unchanged otherwise.

Example:

```
bclr 0 ; Clear Carry Flag
bclr 7 ; Disable interrupts
```

Words	1 (2 bytes)
Cycles	1

12. BLD – Bit Load from the T Flag in SREG to a Bit in Register

12.1. Description

Copies the T Flag in the SREG (Status Register) to bit b in register Rd.

Operation:

(i) Rd(b) ← T

Syntax:	Operands:	Program Counter:
(i) BLD Rd,b	0 ≤ d ≤ 31, 0 ≤ b ≤ 7	PC ← PC + 1

16 bit Opcode:

1111	100d	dddd	0bbb

12.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
        ; Copy bit
bst r1,2 ; Store bit 2 of r1 in T Flag
bld r0,4 ; Load T Flag into bit 4 of r0
```

Words	1 (2 bytes)
Cycles	1

13. BRBC – Branch if Bit in SREG is Cleared

13.1. Description

Conditional relative branch. Tests a single bit in SREG and branches relatively to PC if the bit is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form.

Operation:

(i) If SREG(s) = 0 then PC ← PC + k + 1, else PC ← PC + 1

Syntax:	Operands:	Program Counter:
(i) BRBC s,k	0 ≤ s ≤ 7, -64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	ksss

13.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
cpi r20,5 ; Compare r20 to the value 5
brbc 1,noteq ; Branch if Zero Flag cleared
...
noteq: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

14. BRBS – Branch if Bit in SREG is Set

14.1. Description

Conditional relative branch. Tests a single bit in SREG and branches relatively to PC if the bit is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form.

Operation:

(i) If SREG(s) = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRBS s,k	0 ≤ s ≤ 7, -64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	ksss

14.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
  bst r0,3 ; Load T bit with bit 3 of r0
  brbs 6,bitset ; Branch T bit was set
  ...
bitset: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

15. BRCC – Branch if Carry Cleared

15.1. Description

Conditional relative branch. Tests the Carry Flag (C) and branches relatively to PC if C is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 0,k.)

Operation:

(i) If C = 0 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRCC k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k000

15.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
  add r22,r23 ; Add r23 to r22
  brcc nocarry ; Branch if carry cleared
  ...
nocarry: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

16. BRCS – Branch if Carry Set

16.1. Description

Conditional relative branch. Tests the Carry Flag (C) and branches relatively to PC if C is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 0,k.)

Operation:

(i) If C = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRCS k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k000

16.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
  cpi r26,$56 ; Compare r26 with $56
  brcs carry ; Branch if carry set
  ...
carry: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

17. BREAK – Break

17.1. Description

The BREAK instruction is used by the On-chip Debug system, and is normally not used in the application software. When the BREAK instruction is executed, the AVR CPU is set in the Stopped Mode. This gives the On-chip Debugger access to internal resources.

If any Lock bits are set, or either the JTAGEN or OCDEN Fuses are unprogrammed, the CPU will treat the BREAK instruction as a NOP and will not enter the Stopped mode.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) On-chip Debug system break.

	Syntax:	Operands:	Program Counter:
(i)	BREAK	None	PC ← PC + 1

16-bit Opcode:

1001	0101	1001	1000

17.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	1 (2 bytes)
Cycles	1

18. BREQ – Branch if Equal

18.1. Description

Conditional relative branch. Tests the Zero Flag (Z) and branches relatively to PC if Z is set. If the instruction is executed immediately after any of the instructions CP, CPI, SUB, or SUBI, the branch will occur if and only if the unsigned or signed binary number represented in Rd was equal to the unsigned or signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 1,k.)

Operation:

(i) If Rd = Rr (Z = 1) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BREQ k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k001

18.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
  cp r1,r0 ; Compare registers r1 and r0
  breq equal ; Branch if registers equal
  ...
equal: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

19. BRGE – Branch if Greater or Equal (Signed)

19.1. Description

Conditional relative branch. Tests the Signed Flag (S) and branches relatively to PC if S is cleared. If the instruction is executed immediately after any of the instructions CP, CPI, SUB, or SUBI, the branch will occur if and only if the signed binary number represented in Rd was greater than or equal to the signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 4,k.)

Operation:

(i) If Rd ≥ Rr (N ⊕ V = 0) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRGE k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k100

19.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
  cp r11,r12 ; Compare registers r11 and r12
  brge greateq ; Branch if r11 ≥ r12 (signed)
  ...
greateq: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

20. BRHC – Branch if Half Carry Flag is Cleared

20.1. Description

Conditional relative branch. Tests the Half Carry Flag (H) and branches relatively to PC if H is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 5,k.)

Operation:

(i) If H = 0 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRHC k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k101

20.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 brhc hclear ; Branch if Half Carry Flag cleared
...
hclear: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

21. BRHS – Branch if Half Carry Flag is Set

21.1. Description

Conditional relative branch. Tests the Half Carry Flag (H) and branches relatively to PC if H is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 5,k.)

Operation:

(i) If H = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRHS k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k101

21.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 brhs hset ; Branch if Half Carry Flag set
...
hset: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

22. BRID – Branch if Global Interrupt is Disabled

22.1. Description

Conditional relative branch. Tests the Global Interrupt Flag (I) and branches relatively to PC if I is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 7,k.)

Operation:

(i) If I = 0 then PC ← PC + k + 1, else PC ← PC + 1

Syntax:	Operands:	Program Counter:
(i) BRID k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k111

22.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 brid intdis ; Branch if interrupt disabled
  ...
intdis: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

23. BRIE – Branch if Global Interrupt is Enabled

23.1. Description

Conditional relative branch. Tests the Global Interrupt Flag (I) and branches relatively to PC if I is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 7,k.)

Operation:

(i) If I = 1 then PC ← PC + k + 1, else PC ← PC + 1

Syntax:	Operands:	Program Counter:
(i) BRIE k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k111

23.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 brie inten ; Branch if interrupt enabled
  ...
inten: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

24. BRLO – Branch if Lower (Unsigned)

24.1. Description

Conditional relative branch. Tests the Carry Flag (C) and branches relatively to PC if C is set. If the instruction is executed immediately after any of the instructions CP, CPI, SUB, or SUBI, the branch will occur if and only if, the unsigned binary number represented in Rd was smaller than the unsigned binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 0,k.)

Operation:

(i) If Rd < Rr (C = 1) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRLO k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k000

24.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 eor r19,r19 ; Clear r19
loop: inc r19 ; Increase r19
 ...
 cpi r19,$10 ; Compare r19 with $10
 brlo loop ; Branch if r19 < $10 (unsigned)
 nop ; Exit from loop (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

25. BRLT – Branch if Less Than (Signed)

25.1. Description

Conditional relative branch. Tests the Signed Flag (S) and branches relatively to PC if S is set. If the instruction is executed immediately after any of the instructions CP, CPI, SUB, or SUBI, the branch will occur if and only if, the signed binary number represented in Rd was less than the signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 4,k.)

Operation:

(i) If Rd < Rr (N ⊕ V = 1) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRLT k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k100

25.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 cp r16,r1 ; Compare r16 to r1
 brlt less ; Branch if r16 < r1 (signed)
 ...
less: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

26. BRMI – Branch if Minus

26.1. Description

Conditional relative branch. Tests the Negative Flag (N) and branches relatively to PC if N is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 2,k.)

Operation:

(i) If N = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRMI k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k010

26.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 subi r18,4 ; Subtract 4 from r18
 brmi negative ; Branch if result negative
 ...
negative: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

27. BRNE – Branch if Not Equal

27.1. Description

Conditional relative branch. Tests the Zero Flag (Z) and branches relatively to PC if Z is cleared. If the instruction is executed immediately after any of the instructions CP, CPI, SUB, or SUBI, the branch will occur if and only if, the unsigned or signed binary number represented in Rd was not equal to the unsigned or signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 1,k.)

Operation:

(i) If Rd ≠ Rr (Z = 0) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRNE k	-64 ≤ k ≤ +6	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k001

27.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 eor r27,r27 ; Clear r27
loop: inc r27 ; Increase r27
 ...
 cpi r27,5 ; Compare r27 to 5
 brne loop ; Branch if r27<>5
 nop ; Loop exit (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

28. BRPL – Branch if Plus

28.1. Description

Conditional relative branch. Tests the Negative Flag (N) and branches relatively to PC if N is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 2,k.)

Operation:

(i) If N = 0 then PC ← PC + k + 1, else PC ← PC + 1

Syntax:	Operands:	Program Counter:
(i) BRPL k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k010

28.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 subi r26,$50 ; Subtract $50 from r26
 brpl positive ; Branch if r26 positive
 ...
positive: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

29. BRSH – Branch if Same or Higher (Unsigned)

29.1. Description

Conditional relative branch. Tests the Carry Flag (C) and branches relatively to PC if C is cleared. If the instruction is executed immediately after execution of any of the instructions CP, CPI, SUB, or SUBI, the branch will occur if and only if, the unsigned binary number represented in Rd was greater than or equal to the unsigned binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 0,k.)

Operation:

(i) If Rd ≥Rr (C = 0) then PC ← PC + k + 1, else PC ← PC + 1

Syntax:	Operands:	Program Counter:
(i) BRSH k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k000

29.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 subi r19,4 ; Subtract 4 from r19
 brsh highsm ; Branch if r19 >= 4 (unsigned)
 ...
highsm: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

30. BRTC – Branch if the T Flag is Cleared

30.1. Description

Conditional relative branch. Tests the T Flag and branches relatively to PC if T is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 6,k.)

Operation:

(i) If T = 0 then PC ← PC + k + 1, else PC ← PC + 1

Syntax:	Operands:	Program Counter:
(i) BRTC k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k110

30.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 bst r3,5 ; Store bit 5 of r3 in T Flag
 brtc tclear ; Branch if this bit was cleared
 ...
tclear: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

31. BRTS – Branch if the T Flag is Set

31.1. Description

Conditional relative branch. Tests the T Flag and branches relatively to PC if T is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 6,k.)

Operation:

(i) If T = 1 then PC ← PC + k + 1, else PC ← PC + 1

Syntax:	Operands:	Program Counter:
(i) BRTS k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k110

31.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 bst r3,5 ; Store bit 5 of r3 in T Flag
 brts tset ; Branch if this bit was set
 ...
tset: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

32. BRVC – Branch if Overflow Cleared

32.1. Description

Conditional relative branch. Tests the Overflow Flag (V) and branches relatively to PC if V is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 3,k.)

Operation:

(i) If V = 0 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRVC k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k011

32.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 add r3,r4 ; Add r4 to r3
 brvc noover ; Branch if no overflow
 ...
noover: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

33. BRVS – Branch if Overflow Set

33.1. Description

Conditional relative branch. Tests the Overflow Flag (V) and branches relatively to PC if V is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). Parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 3,k.)

Operation:

(i) If V = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRVS k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k011

33.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 add r3,r4 ; Add r4 to r3
 brvs overfl ; Branch if overflow
 ...
overfl: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

34. BSET – Bit Set in SREG

34.1. Description

Sets a single Flag or bit in SREG.

Operation:

(i) SREG(s) ← 1

Syntax:	Operands:	Program Counter:
(i) BSET s	0 ≤ s ≤ 7	PC ← PC + 1

16-bit Opcode:

1001	0100	0sss	1000

34.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
⇔	⇔	⇔	⇔	⇔	⇔	⇔	⇔

I 1 if s = 7; Unchanged otherwise.

T 1 if s = 6; Unchanged otherwise.

H 1 if s = 5; Unchanged otherwise.

S 1 if s = 4; Unchanged otherwise.

V 1 if s = 3; Unchanged otherwise.

N 1 if s = 2; Unchanged otherwise.

Z 1 if s = 1; Unchanged otherwise.

C 1 if s = 0; Unchanged otherwise.

Example:

```
bset 6 ; Set T Flag
bset 7 ; Enable interrupt
```

Words	1 (2 bytes)
Cycles	1

35. BST – Bit Store from Bit in Register to T Flag in SREG

35.1. Description

Stores bit b from Rd to the T Flag in SREG (Status Register).

Operation:

(i) T ← Rd(b)

Syntax:	Operands:	Program Counter:
(i) BST Rd,b	0 ≤ d ≤ 31, 0 ≤ b ≤ 7	PC ← PC + 1

16-bit Opcode:

1111	101d	dddd	0bbb

35.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	⇔	–	–	–	–	–	–

T 0 if bit b in Rd is cleared. Set to 1 otherwise.

Example:

```
 ; Copy bit
bst r1,2 ; Store bit 2 of r1 in T Flag
bld r0,4 ; Load T into bit 4 of r0
```

Words	1 (2 bytes)
Cycles	1

36. CALL – Long Call to a Subroutine

36.1. Description

Calls to a subroutine within the entire Program memory. The return address (to the instruction after the CALL) will be stored onto the Stack. (See also RCALL). The Stack Pointer uses a post-decrement scheme during CALL.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) PC ← k Devices with 16-bit PC, 128KB Program memory maximum.

(ii) PC ← k Devices with 22-bit PC, 8MB Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i)	CALL k	0 ≤ k < 64K	PC ← k	STACK ← PC+2 SP ← SP-2, (2 bytes, 16 bits)
(ii)	CALL k	0 ≤ k < 4M	PC ← k	STACK ← PC+2 SP ← SP-3 (3 bytes, 22 bits)

32-bit Opcode:

1001	010k	kkkk	111k
kkkk	kkkk	kkkk	kkkk

36.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 mov r16,r0 ; Copy r0 to r16
 call check ; Call subroutine
 nop ; Continue (do nothing)
 ...
check: cpi r16,$42 ; Check if r16 has a special value
 breq error ; Branch if equal
 ret ; Return from subroutine
 ...
error: rjmp error ; Infinite loop
```

Words	2 (4 bytes)
Cycles	4 devices with 16-bit PC

	5 devices with 22-bit PC
Cycles XMEGA	3 devices with 16-bit PC
	4 devices with 22-bit PC

37. CBI – Clear Bit in I/O Register

37.1. Description

Clears a specified bit in an I/O register. This instruction operates on the lower 32 I/O registers – addresses 0-31.

Operation:

(i) I/O(A,b) ← 0

	Syntax:	Operands:	Program Counter:
(i)	CBI A,b	0 ≤ A ≤ 31, 0 ≤ b ≤ 7	PC ← PC + 1

16-bit Opcode:

1001	1000	AAAA	Abbb

37.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
cbi $12,7 ; Clear bit 7 in Port D
```

Words	1 (2 bytes)
Cycles	2
Cycles XMEGA	1
Cycles Reduced Core tinyAVR	1

38. CBR – Clear Bits in Register

38.1. Description

Clears the specified bits in register Rd. Performs the logical AND between the contents of register Rd and the complement of the constant mask K. The result will be placed in register Rd.

Operation:

(i) Rd ← Rd • ($FF - K)

	Syntax:	Operands:	Program Counter:
(i)	CBR Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode: (see ANDI with K complemented)

38.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S N ⊕ V, for signed tests.

V 0

Cleared.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
cbr r16,$F0 ; Clear upper nibble of r16
cbr r18,1 ; Clear bit 0 in r18
```

Words	1 (2 bytes)
Cycles	1

39. CLC – Clear Carry Flag

39.1. Description

Clears the Carry Flag (C) in SREG (Status Register).

Operation:

(i) C ← 0

	Syntax:	Operands:	Program Counter:
(i)	CLC	None	PC ← PC + 1

16-bit Opcode:

1001	0100	1000	1000

39.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	0

C 0

Carry Flag cleared.

Example:

```
add r0,r0 ; Add r0 to itself
clc ; Clear Carry Flag
```

Words	1 (2 bytes)
Cycles	1

40. CLH – Clear Half Carry Flag

40.1. Description

Clears the Half Carry Flag (H) in SREG (Status Register).

Operation:

(i) H ← 0

	Syntax:	Operands:	Program Counter:
(i)	CLH	None	PC ← PC + 1

16-bit Opcode:

1001	0100	1101	1000

40.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	0	–	–	–	–	–

H 0

Half Carry Flag cleared.

Example:

```
clh ; Clear the Half Carry Flag
```

Words	1 (2 bytes)
Cycles	1

41. CLI – Clear Global Interrupt Flag

41.1. Description

Clears the Global Interrupt Flag (I) in SREG (Status Register). The interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction.

Operation:

(i) I ← 0

Syntax:	Operands:	Program Counter:
(i) CLI	None	PC ← PC + 1

16-bit Opcode:

1001	0100	1111	1000

41.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
0	–	–	–	–	–	–	–

I 0

Global Interrupt Flag cleared.

Example:

```
in temp, SREG ; Store SREG value (temp must be defined by user)
cli ; Disable interrupts during timed sequence
sbi EECR, EEMWE ; Start EEPROM write
sbi EECR, EEWE
out SREG, temp ; Restore SREG value (I-Flag)
```

Words	1 (2 bytes)
Cycles	1

42. CLN – Clear Negative Flag

42.1. Description

Clears the Negative Flag (N) in SREG (Status Register).

Operation:

(i) N ← 0

Syntax:	Operands:	Program Counter:
(i) CLN	None	PC ← PC + 1

16-bit Opcode:

1001	0100	1010	1000

42.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	0	–	–

N 0

Negative Flag cleared.

Example:

```
add r2,r3 ; Add r3 to r2
cln ; Clear Negative Flag
```

Words	1 (2 bytes)
Cycles	1

43. CLR – Clear Register

43.1. Description

Clears a register. This instruction performs an Exclusive OR between a register and itself. This will clear all bits in the register.

Operation:

(i) Rd ← Rd ⊕ Rd

	Syntax:	Operands:	Program Counter:
(i)	CLR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode: (see EOR Rd,Rd)

0010	01dd	dddd	dddd

43.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	0	0	0	1	–

S 0
Cleared.

V 0
Cleared.

N 0
Cleared.

Z 1
Set.

R (Result) equals Rd after the operation.

Example:

```
 clr r18 ; clear r18
loop: inc r18 ; increase r18
 ...
 cpi r18,$50 ; Compare r18 to $50
 brne loop
```

Words	1 (2 bytes)
Cycles	1

44. CLS – Clear Signed Flag

44.1. Description

Clears the Signed Flag (S) in SREG (Status Register).

Operation:

(i) S ← 0

	Syntax:	Operands:	Program Counter:
(i)	CLS	None	PC ← PC + 1

16-bit Opcode:

1001	0100	1100	1000

44.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	0	–	–	–	–

S 0
Signed Flag cleared.

Example:

```
add r2,r3 ; Add r3 to r2
cls ; Clear Signed Flag
```

Words	1 (2 bytes)
Cycles	1

45. CLT – Clear T Flag

45.1. Description

Clears the T Flag in SREG (Status Register).

Operation:

(i) T ← 0

	Syntax:	Operands:	Program Counter:
(i)	CLT	None	PC ← PC + 1

16-bit Opcode:

1001	0100	1110	1000

45.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	0	–	–	–	–	–	–

T 0

T Flag cleared.

Example:

```
clt ; Clear T Flag
```

Words	1 (2 bytes)
Cycles	1

46. CLV – Clear Overflow Flag

46.1. Description

Clears the Overflow Flag (V) in SREG (Status Register).

Operation:

(i) V ← 0

	Syntax:	Operands:	Program Counter:
(i)	CLV	None	PC ← PC + 1

16-bit Opcode:

1001	0100	1011	1000

46.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	0	–	–	–

V 0

Overflow Flag cleared.

Example:

```
add r2,r3 ; Add r3 to r2
clv ; Clear Overflow Flag
```

Words	1 (2 bytes)
Cycles	1

47. CLZ – Clear Zero Flag

47.1. Description

Clears the Zero Flag (Z) in SREG (Status Register).

Operation:

(i) Z ← 0

Syntax:	Operands:	Program Counter:
(i) CLZ	None	PC ← PC + 1

16-bit Opcode:

1001	0100	1001	1000

47.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	0	–

Z 0

Zero Flag cleared.

Example:

```
add r2,r3 ; Add r3 to r2
clz ; Clear zero
```

Words	1 (2 bytes)
Cycles	1

48. COM – One's Complement

48.1. Description

This instruction performs a One's Complement of register Rd.

Operation:

(i) Rd ← $FF - Rd

Syntax:	Operands:	Program Counter:
(i) COM Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0000

48.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	1

S N ⊕ V, for signed tests.

V 0

Cleared.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

C 1

Set.

R (Result) equals Rd after the operation.

Example:

```
com r4 ; Take one's complement of r4
breq zero ; Branch if zero
...
zero: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1

49. CP – Compare

49.1. Description

This instruction performs a compare between two registers Rd and Rr. None of the registers are changed. All conditional branches can be used after this instruction.

Operation:

(i) Rd - Rr

	Syntax:	Operands:	Program Counter:
(i)	CP Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0001	01rd	dddd	rrrr

49.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$

Set if there was a borrow from bit 3; cleared otherwise.

S N ⊕ V, for signed tests.

V $Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$

Set if two's complement overflow resulted from the operation; cleared otherwise.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

C $\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$

Set if the absolute value of the contents of Rr is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

Example:

```
  cp r4,r19 ; Compare r4 with r19
  brne noteq ; Branch if r4 <> r19
  ...
noteq: nop ; Branch destination (do nothing)
```


Words	1 (2 bytes)
Cycles	1

50. CPC – Compare with Carry

50.1. Description

This instruction performs a compare between two registers Rd and Rr and also takes into account the previous carry. None of the registers are changed. All conditional branches can be used after this instruction.

Operation:

(i) Rd - Rr - C

	Syntax:	Operands:	Program Counter:
(i)	CPC Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0000	01rd	dddd	rrrr

50.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$

Set if there was a borrow from bit 3; cleared otherwise.

S N ⊕ V, for signed tests.

V $Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$

Set if two's complement overflow resulted from the operation; cleared otherwise.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0} \bullet Z$

Previous value remains unchanged when the result is zero; cleared otherwise.

C $\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$

Set if the absolute value of the contents of Rr plus previous carry is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

Example:

```
; Compare r3:r2 with r1:r0
cp r2,r0 ; Compare low byte
cpc r3,r1 ; Compare high byte
 brne noteq ; Branch if not equal
 ...
noteq: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1

51. CPI – Compare with Immediate

51.1. Description

This instruction performs a compare between register Rd and a constant. The register is not changed. All conditional branches can be used after this instruction.

Operation:

(i) Rd - K

	Syntax:	Operands:	Program Counter:
(i)	CPI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

0011	KKKK	dddd	KKKK

51.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H $\overline{Rd3} \bullet K3 + K3 \bullet R3 + R3 \bullet \overline{Rd3}$

Set if there was a borrow from bit 3; cleared otherwise.

S N ⊕ V, for signed tests.

V $Rd7 \bullet \overline{K7} \bullet \overline{R7} + \overline{Rd7} \bullet K7 \bullet R7$

Set if two's complement overflow resulted from the operation; cleared otherwise.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

C $\overline{Rd7} \bullet K7 + K7 \bullet R7 + R7 \bullet \overline{Rd7}$

Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

Example:

```
  cpi r19,3 ; Compare r19 with 3
  brne error ; Branch if r19<>3
  ...
error: nop ; Branch destination (do nothing)
```


Words	1 (2 bytes)
Cycles	1

52. CPSE – Compare Skip if Equal

52.1. Description

This instruction performs a compare between two registers Rd and Rr, and skips the next instruction if Rd = Rr.

Operation:

(i) If Rd = Rr then PC ← PC + 2 (or 3) else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	CPSE Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1, Condition false - no skip PC ← PC + 2, Skip a one word instruction PC ← PC + 3, Skip a two word instruction

16-bit Opcode:

0001	00rd	dddd	rrrr

52.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
inc r4 ; Increase r4
cpse r4,r0 ; Compare r4 to r0
neg r4 ; Only executed if r4<>r0
nop ; Continue (do nothing)
```

Words 1 (2 bytes)

Cycles 1 if condition is false (no skip)

2 if condition is true (skip is executed) and the instruction skipped is 1 word

3 if condition is true (skip is executed) and the instruction skipped is 2 words

53. DEC – Decrement

53.1. Description

Subtracts one -1- from the contents of register Rd and places the result in the destination register Rd.

The C Flag in SREG is not affected by the operation, thus allowing the DEC instruction to be used on a loop counter in multiple-precision computations.

When operating on unsigned values, only BREQ and BRNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

Operation:

(i) Rd ← Rd - 1

	Syntax:	Operands:	Program Counter:
(i)	DEC Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	1010

53.2. Status Register and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	–

S N ⊕ V, for signed tests.

V $\overline{R7} \bullet R6 \bullet R5 \bullet R4 \bullet R3 \bullet R2 \bullet R1 \bullet R0$

Set if two's complement overflow resulted from the operation; cleared otherwise. Two's complement overflow occurs if and only if Rd was $80 before the operation.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
 ldi r17,$10 ; Load constant in r17
loop: add r1,r2 ; Add r2 to r1
 dec r17 ; Decrement r17
 brne loop ; Branch if r17<>0
 nop ; Continue (do nothing)
```

Words	1 (2 bytes)
Cycles	1

54. DES – Data Encryption Standard

54.1. Description

The module is an instruction set extension to the AVR CPU, performing DES iterations. The 64-bit data block (plaintext or ciphertext) is placed in the CPU register file, registers R0-R7, where LSB of data is placed in LSB of R0 and MSB of data is placed in MSB of R7. The full 64-bit key (including parity bits) is placed in registers R8-R15, organized in the register file with LSB of key in LSB of R8 and MSB of key in MSB of R15. Executing one DES instruction performs one round in the DES algorithm. Sixteen rounds must be executed in increasing order to form the correct DES ciphertext or plaintext. Intermediate results are stored in the register file (R0-R15) after each DES instruction. The instruction's operand (K) determines which round is executed, and the half carry flag (H) determines whether encryption or decryption is performed.

The DES algorithm is described in “Specifications for the Data Encryption Standard” (Federal Information Processing Standards Publication 46). Intermediate results in this implementation differ from the standard because the initial permutation and the inverse initial permutation are performed in each iteration. This does not affect the result in the final ciphertext or plaintext, but reduces the execution time.

Operation:

(i) If H = 0 then Encrypt round (R7-R0, R15-R8, K)
If H = 1 then Decrypt round (R7-R0, R15-R8, K)

	Syntax:	Operands:	Program Counter:
(i)	DES K	0x00≤K≤0x0F	PC ← PC + 1

16-bit Opcode:

1001	0100	KKKK	1011

Example:

```
DES 0x00
DES 0x01
…
DES 0x0E
DES 0x0F
```

Words 1 (2 bytes)

Cycles 1
Note: If the DES instruction is succeeding a non-DES instruction, an extra cycle is inserted.

55. EICALL – Extended Indirect Call to Subroutine

55.1. Description

Indirect call of a subroutine pointed to by the Z (16 bits) Pointer Register in the Register File and the EIND Register in the I/O space. This instruction allows for indirect calls to the entire 4M (words) Program memory space. See also ICALL. The Stack Pointer uses a post-decrement scheme during EICALL.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) PC(15:0) ← Z(15:0)
PC(21:16) ← EIND

	Syntax:	Operands:	Program Counter:	Stack:
(i)	EICALL	None	See Operation	STACK ← PC + 1 SP ← SP - 3 (3 bytes, 22 bits)

16-bit Opcode:

1001	0101	0001	1001

55.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
ldi r16,$05 ; Set up EIND and Z-pointer
out EIND,r16
ldi r30,$00
ldi r31,$10
eicall ; Call to $051000
```

Words	1 (2 bytes)
Cycles	4 (only implemented in devices with 22-bit PC)
Cycles XMEGA	3 (only implemented in devices with 22-bit PC)

56. EIJMP – Extended Indirect Jump

56.1. Description

Indirect jump to the address pointed to by the Z (16 bits) Pointer Register in the Register File and the EIND Register in the I/O space. This instruction allows for indirect jumps to the entire 4M (words) Program memory space. See also IJMP.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) PC(15:0) ← Z(15:0)
PC(21:16) ← EIND

	Syntax:	Operands:	Program Counter:	Stack:
(i)	EIJMP	None	See Operation	Not Affected

16-bit Opcode:

1001	0100	0001	1001

56.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
ldi r16,$05 ; Set up EIND and Z-pointer
out EIND,r16
ldi r30,$00
ldi r31,$10
eijmp ; Jump to $051000
```

Words	1 (2 bytes)
Cycles	2

57. ELPM – Extended Load Program Memory

57.1. Description

Loads one byte pointed to by the Z-register and the RAMPZ Register in the I/O space, and places this byte in the destination register Rd. This instruction features a 100% space effective constant initialization or constant data fetch. The Program memory is organized in 16-bit words while the Z-pointer is a byte address. Thus, the least significant bit of the Z-pointer selects either low byte (Z_{LSB} = 0) or high byte (Z_{LSB} = 1). This instruction can address the entire Program memory space. The Z-pointer Register can either be left unchanged by the operation, or it can be incremented. The incrementation applies to the entire 24-bit concatenation of the RAMPZ and Z-pointer Registers.

Devices with Self-Programming capability can use the ELPM instruction to read the Fuse and Lock bit value. Refer to the device documentation for a detailed description.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

The result of these combinations is undefined:

ELPM r30, Z+

ELPM r31, Z+

	Operation:		Comment:
(i)	R0 ← (RAMPZ:Z)		RAMPZ:Z: Unchanged, R0 implied destination register
(ii)	Rd ← (RAMPZ:Z)		RAMPZ:Z: Unchanged
(iii)	Rd ← (RAMPZ:Z)		(RAMPZ:Z) ← (RAMPZ:Z) + 1 RAMPZ:Z: Post incremented
	Syntax:	Operands:	Program Counter:
(i)	ELPM	None, R0 implied	PC ← PC + 1
(ii)	ELPM Rd, Z	0 ≤ d ≤ 31	PC ← PC + 1
(iii)	ELPM Rd, Z+	0 ≤ d ≤ 31	PC ← PC + 1

16 bit Opcode:

(i)	1001	0101	1101	1000
(ii)	1001	000d	dddd	0110
(iii)	1001	000d	dddd	0111

57.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 ldi ZL, byte3(Table_1<<1) ; Initialize Z-pointer
 out RAMPZ, ZL
 ldi ZH, byte2(Table_1<<1)
 ldi ZL, byte1(Table_1<<1)
 elpm r16, Z+ ; Load constant from Program
 ; memory pointed to by RAMPZ:Z (Z is r31:r30)
...
Table_1:
.dw 0x3738 ; 0x38 is addressed when Z_LSB = 0
 ; 0x37 is addressed when Z_LSB = 1
...
```

Words	1 (2 bytes)
Cycles	3

58. EOR – Exclusive OR

58.1. Description

Performs the logical EOR between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd ⊕ Rr

Syntax:	Operands:	Program Counter:
(i) EOR Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 3	PC ← PC + 1

16-bit Opcode:

0010	01rd	dddd	rrrr

58.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S N ⊕ V, for signed tests.

V 0

Cleared.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
eor r4,r4 ; Clear r4
eor r0,r22 ; Bitwise exclusive or between r0 and r22
```

Words	1 (2 bytes)
Cycles	1

59. FMUL – Fractional Multiply Unsigned

59.1. Description

This instruction performs 8-bit × 8-bit → 16-bit unsigned multiplication and shifts the result one bit left.

Let (N.Q) denote a fractional number with N binary digits left of the radix point, and Q binary digits right of the radix point. A multiplication between two numbers in the formats (N1.Q1) and (N2.Q2) results in the format ((N1+N2).(Q1+Q2)). For signal processing applications, the format (1.7) is widely used for the inputs, resulting in a (2.14) format for the product. A left shift is required for the high byte of the product to be in the same format as the inputs. The FMUL instruction incorporates the shift operation in the same number of cycles as MUL.

The (1.7) format is most commonly used with signed numbers, while FMUL performs an unsigned multiplication. This instruction is therefore most useful for calculating one of the partial products when performing a signed multiplication with 16-bit inputs in the (1.15) format, yielding a result in the (1.31) format. Note: the result of the FMUL operation may suffer from a 2's complement overflow if interpreted as a number in the (1.15) format. The MSB of the multiplication before shifting must be taken into account, and is found in the carry bit. See the following example.

The multiplicand Rd and the multiplier Rr are two registers containing unsigned fractional numbers where the implicit radix point lies between bit 6 and bit 7. The 16-bit unsigned fractional product with the implicit radix point between bit 14 and bit 15 is placed in R1 (high byte) and R0 (low byte).

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) R1:R0 ← Rd × Rr (unsigned (1.15) ← unsigned (1.7) × unsigned (1.7))

Syntax:	Operands:	Program Counter:
(i) FMUL Rd,Rr	16 ≤ d ≤ 23, 16 ≤ r ≤ 23	PC ← PC + 1

(i) PC ← PC + 1

16-bit Opcode:

0000	0011	0ddd	1rrr

59.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	⇔	⇔

C R16

Set if bit 15 of the result before left shift is set; cleared otherwise.

Z $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $0000; cleared otherwise.

R (Result) equals R1,R0 after the operation.

Example:

```
;***************************************************************************
;* DESCRIPTION
;* Signed fractional multiply of two 16-bit numbers with 32-bit result.
;* USAGE
;* r19:r18:r17:r16 = ( r23:r22 * r21:r20 ) << 1
;***************************************************************************
fmuls16x16_32:
 clr r2
 fmuls r23, r21 ;((signed)ah * (signed)bh) << 1
 movw r19:r18, r1:r0
 fmul r22, r20 ;(al * bl) << 1
 adc r18, r2
 movw r17:r16, r1:r0
 fmulsu r23, r20 ;((signed)ah * bl) << 1
 sbc r19, r2
 add r17, r0
 adc r18, r1
 adc r19, r2
 fmulsu r21, r22 ;((signed)bh * al) << 1
 sbc r19, r2
 add r17, r0
 adc r18, r1
 adc r19, r2
```

Words	1 (2 bytes)
Cycles	2

60. FMULS – Fractional Multiply Signed

60.1. Description

This instruction performs 8-bit × 8-bit → 16-bit signed multiplication and shifts the result one bit left.

Rd		Rr		R1	R0
Multiplicand	×	Multiplier	→	Product High	Product Low
8		8		16	

Let (N.Q) denote a fractional number with N binary digits left of the radix point, and Q binary digits right of the radix point. A multiplication between two numbers in the formats (N1.Q1) and (N2.Q2) results in the format ((N1+N2).(Q1+Q2)). For signal processing applications, the format (1.7) is widely used for the inputs, resulting in a (2.14) format for the product. A left shift is required for the high byte of the product to be in the same format as the inputs. The FMULS instruction incorporates the shift operation in the same number of cycles as MULS.

The multiplicand Rd and the multiplier Rr are two registers containing signed fractional numbers where the implicit radix point lies between bit 6 and bit 7. The 16-bit signed fractional product with the implicit radix point between bit 14 and bit 15 is placed in R1 (high byte) and R0 (low byte).

Note that when multiplying 0x80 (-1) with 0x80 (-1), the result of the shift operation is 0x8000 (-1). The shift operation thus gives a two's complement overflow. This must be checked and handled by software.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) R1:R0 ← Rd × Rr (signed (1.15) ← signed (1.7) × signed (1.7))

	Syntax:	Operands:	Program Counter:
(i)	FMULS Rd,Rr	16 ≤ d ≤ 23, 16 ≤ r ≤ 23	PC ← PC + 1

16-bit Opcode:

0000	0011	1ddd	0rrr

60.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	⇔	⇔

C R16

Set if bit 15 of the result before left shift is set; cleared otherwise.

Z $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $0000; cleared otherwise.

R (Result) equals R1,R0 after the operation.

Example:

```
fmuls r23,r22 ; Multiply signed r23 and r22 in (1.7) format, result in (1.15) format
mov w r23:r22,r1:r0 ; Copy result back in r23:r22
```

Words	1 (2 bytes)
Cycles	2

61. FMULSU – Fractional Multiply Signed with Unsigned

61.1. Description

This instruction performs 8-bit × 8-bit → 16-bit signed multiplication and shifts the result one bit left.

Let (N.Q) denote a fractional number with N binary digits left of the radix point, and Q binary digits right of the radix point. A multiplication between two numbers in the formats (N1.Q1) and (N2.Q2) results in the format ((N1+N2).(Q1+Q2)). For signal processing applications, the format (1.7) is widely used for the inputs, resulting in a (2.14) format for the product. A left shift is required for the high byte of the product to be in the same format as the inputs. The FMULSU instruction incorporates the shift operation in the same number of cycles as MULSU.

The (1.7) format is most commonly used with signed numbers, while FMULSU performs a multiplication with one unsigned and one signed input. This instruction is therefore most useful for calculating two of the partial products when performing a signed multiplication with 16-bit inputs in the (1.15) format, yielding a result in the (1.31) format. Note: the result of the FMULSU operation may suffer from a 2's complement overflow if interpreted as a number in the (1.15) format. The MSB of the multiplication before shifting must be taken into account, and is found in the carry bit. See the following example.

The multiplicand Rd and the multiplier Rr are two registers containing fractional numbers where the implicit radix point lies between bit 6 and bit 7. The multiplicand Rd is a signed fractional number, and the multiplier Rr is an unsigned fractional number. The 16-bit signed fractional product with the implicit radix point between bit 14 and bit 15 is placed in R1 (high byte) and R0 (low byte).

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) R1:R0 ← Rd × Rr (signed (1.15) ← signed (1.7) × unsigned (1.7))

	Syntax:	Operands:	Program Counter:
(i)	FMULSU Rd,Rr	16 ≤ d ≤ 23, 16 ≤ r ≤ 23	PC ← PC + 1

16-bit Opcode:

0000	0011	1ddd	1rrr

61.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	⇔	⇔

C R16

Set if bit 15 of the result before left shift is set; cleared otherwise.

Z $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $0000; cleared otherwise.

R (Result) equals R1,R0 after the operation.

Example:

```
;*****************************************************************************
;* DESCRIPTION
;* Signed fractional multiply of two 16-bit numbers with 32-bit result.
;* USAGE
;* r19:r18:r17:r16 = ( r23:r22 * r21:r20 ) << 1
;*****************************************************************************
fmuls16x16_32:
 clr r2
 fmuls r23, r21 ;((signed)ah * (signed)bh) << 1
 movw r19:r18, r1:r0
 fmul r22, r20 ;(al * bl) << 1
 adc r18, r2
 movw r17:r16, r1:r0
 fmulsu r23, r20 ;((signed)ah * bl) << 1
 sbc r19, r2
 add r17, r0
 adc r18, r1
 adc r19, r2
 fmulsu r21, r22 ;((signed)bh * al) << 1
 sbc r19, r2
 add r17, r0
 adc r18, r1
 adc r19, r2
```

Words	1 (2 bytes)
Cycles	2

62. ICALL – Indirect Call to Subroutine

62.1. Description

Calls to a subroutine within the entire 4M (words) Program memory. The return address (to the instruction after the CALL) will be stored onto the Stack. See also RCALL. The Stack Pointer uses a post-decrement scheme during CALL.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

	Operation:	Comment:
(i)	PC(15:0) ← Z(15:0)	Devices with 16-bit PC, 128KB Program memory maximum.
(ii)	PC(15:0) ← Z(15:0) PC(21:16) ← 0	Devices with 22-bit PC, 8MB Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i)	ICALL	None	See Operation	STACK ← PC + 1 SP ← SP - 2 (2 bytes, 16 bits)
(ii)	ICALL	None	See Operation	STACK ← PC + 1 SP ← SP - 3 (3 bytes, 22 bits)

16-bit Opcode:

1001	0101	0000	1001

62.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
mov r30,r0 ; Set offset to call table
icall ; Call routine pointed to by r31:r30
```

Words	1 (2 bytes)
Cycles	3 devices with 16-bit PC
	4 devices with 22-bit PC
Cycles XMEGA®	2 devices with 16-bit PC
	3 devices with 22-bit PC

63. IJMP – Indirect Jump

63.1. Description

Indirect jump to the address pointed to by the Z (16 bits) Pointer Register in the Register File. The Z-pointer Register is 16 bits wide and allows jump within the lowest 64K words (128KB) section of Program memory.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

	Operation:	Comment:
(i)	PC ← Z(15:0)	Devices with 16-bit PC, 128KB Program memory maximum.
(ii)	PC(15:0) ← Z(15:0) PC(21:16) ← 0	Devices with 22-bit PC, 8MB Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i), (ii)	IJMP	None	See Operation	Not Affected

16-bit Opcode:

1001	0100	0000	1001

63.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
mov r30,r0 ; Set offset to jump table
ijmp ; Jump to routine pointed to by r31:r30
```

Words	1 (2 bytes)
Cycles	2

64. IN - Load an I/O Location to Register

64.1. Description

Loads data from the I/O Space (Ports, Timers, Configuration Registers, etc.) into register Rd in the Register File.

	Operation:
(i)	Rd ← I/O(A)

	Syntax:	Operands:	Program Counter:
(i)	IN Rd,A	0 ≤ d ≤ 31, 0 ≤ A ≤ 63	PC ← PC + 1

16-bit Opcode:

1011	0AAd	dddd	AAAA

64.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
in r25,$16 ; Read Port B
cpi r25,4 ; Compare read value to constant
breq exit ; Branch if r25=4
...
exit: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1

65. INC – Increment

65.1. Description

Adds one -1- to the contents of register Rd and places the result in the destination register Rd.

The C Flag in SREG is not affected by the operation, thus allowing the INC instruction to be used on a loop counter in multiple-precision computations.

When operating on unsigned numbers, only BREQ and BRNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

Operation:

(i) Rd ← Rd + 1

	Syntax:	Operands:	Program Counter:
(i)	INC Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0011

65.2. Status Register and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	–

S N ⊕ V, for signed tests.

V $R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if two's complement overflow resulted from the operation; cleared otherwise. Two's complement overflow occurs if and only if Rd was $7F before the operation.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
 clr r22 ; clear r22
loop: inc r22 ; increment r22
 ...
 cpi r22,$4F ; Compare r22 to $4f
 brne loop ; Branch if not equal
 nop ; Continue (do nothing)
```


Words	1 (2 bytes)
Cycles	1

66. JMP – Jump

66.1. Description

Jump to an address within the entire 4M (words) Program memory. See also RJMP.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) PC ← k

	Syntax:	Operands:	Program Counter:	Stack:
(i)	JMP k	0 ≤ k < 4M	PC ← k	Unchanged

32-bit Opcode:

1001	010k	kkkk	110k
kkkk	kkkk	kkkk	kkkk

66.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
mov r1,r0 ; Copy r0 to r1
jmp farplc ; Unconditional jump
...
farplc: nop ; Jump destination (do nothing)
```

Words	2 (4 bytes)
Cycles	3

67. LAC – Load and Clear

67.1. Description

Load one byte indirect from data space to register and stores and clear the bits in data space specified by the register. The instruction can only be used towards internal SRAM.

The data location is pointed to by the Z (16 bits) Pointer Register in the Register File. Memory access is limited to the current data segment of 64KB. To access another data segment in devices with more than 64KB data space, the RAMPZ in register in the I/O area has to be changed.

The Z-pointer Register is left unchanged by the operation. This instruction is especially suited for clearing status bits stored in SRAM.

Operation:

(i) (Z) ← ($FF – Rd) • (Z), Rd ← (Z)

	Syntax:	Operands:	Program Counter:
(i)	LAC Z,Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	001r	rrrr	0110

67.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	1 (2 bytes)
Cycles	2

68. LAS – Load and Set

68.1. Description

Load one byte indirect from data space to register and set bits in data space specified by the register. The instruction can only be used towards internal SRAM.

The data location is pointed to by the Z (16 bits) Pointer Register in the Register File. Memory access is limited to the current data segment of 64KB. To access another data segment in devices with more than 64KB data space, the RAMPZ in register in the I/O area has to be changed.

The Z-pointer Register is left unchanged by the operation. This instruction is especially suited for setting status bits stored in SRAM.

Operation:

(i) (Z) ← Rd v (Z), Rd ← (Z)

	Syntax:	Operands:	Program Counter:
(i)	LAS Z,Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	001r	rrrr	0101

68.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	1 (2 bytes)
Cycles	2

69. LAT – Load and Toggle

69.1. Description

Load one byte indirect from data space to register and toggles bits in the data space specified by the register. The instruction can only be used towards SRAM.

The data location is pointed to by the Z (16 bits) Pointer Register in the Register File. Memory access is limited to the current data segment of 64KB. To access another data segment in devices with more than 64KB data space, the RAMPZ in register in the I/O area has to be changed.

The Z-pointer Register is left unchanged by the operation. This instruction is especially suited for changing status bits stored in SRAM.

Operation:

(i) (Z) ← Rd ⊕ (Z), Rd ← (Z)

	Syntax:	Operands:	Program Counter:
(i)	LAT Z,Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	001r	rrrr	0111

69.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	1 (2 bytes)
Cycles	2

70. LD – Load Indirect from Data Space to Register using Index X

70.1. Description

Loads one byte indirect from the data space to a register. For parts with SRAM, the data space consists of the Register File, I/O memory, and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. In some parts the Flash Memory has been mapped to the data space and can be read using this command. The EEPROM has a separate address space.

The data location is pointed to by the X (16 bits) Pointer Register in the Register File. Memory access is limited to the current data segment of 64KB. To access another data segment in devices with more than 64KB data space, the RAMPX in register in the I/O area has to be changed.

The X-pointer Register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for accessing arrays, tables, and Stack Pointer usage of the X-pointer Register. Note that only the low byte of the X-pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPX Register in the I/O area is updated in parts with more than 64KB data space or more than 64KB Program memory, and the increment/decrement is added to the entire 24-bit address on such devices.

Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

In the Reduced Core tinyAVR the LD instruction can be used to achieve the same operation as LPM since the program memory is mapped to the data memory space.

The result of these combinations is undefined:

LD r26, X+

LD r27, X+

LD r26, -X

LD r27, -X

Using the X-pointer:

	Operation:		Comment:
(i)	Rd ← (X)		X: Unchanged
(ii)	Rd ← (X) X ← X + 1		X: Post incremented
(iii)	X ← X - 1 Rd ← (X)		X: Pre decremented
	Syntax:	Operands:	Program Counter:
(i)	LD Rd, X	0 ≤ d ≤ 31	PC ← PC + 1
(ii)	LD Rd, X+	0 ≤ d ≤ 31	PC ← PC + 1
(iii)	LD Rd, -X	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

(i)	1001	000d	dddd	1100
(ii)	1001	000d	dddd	1101
(iii)	1001	000d	dddd	1110

70.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr r27 ; Clear X high byte
ldi r26,$60 ; Set X low byte to $60
ld r0,X+ ; Load r0 with data space loc. $60(X post inc)
ld r1,X ; Load r1 with data space loc. $61
ldi r26,$63 ; Set X low byte to $63
ld r2,X ; Load r2 with data space loc. $63
ld r3,-X ; Load r3 with data space loc. $62(X pre dec)
```

Words	1 (2 bytes)
Cycles	(i): 1[2]
	(ii): 2
	(iii): 3[2]
Cycles XMEGA	(i): 1[1]
	(ii): 1[1]
	(iii): 2[1]

1. Note: If the LD instruction is accessing internal SRAM, one extra cycle is inserted.
2. Note: LD instruction can load data from program memory since the flash is memory mapped. Loading data from the data memory takes one clock cycle, and loading from the program memory takes two clock cycles. But if an interrupt occur (before the last clock cycle) no additional clock cycles are necessary when loading from the program memory. Hence, the instruction takes only one clock cycle to execute.

   LD instruction with pre-decrement can load data from program memory since the flash is memory mapped. Loading data from the data memory takes two clock cycles, and loading from the program memory takes three clock cycles. But if an interrupt occur (before the last clock cycle) no additional clock cycles are necessary when loading from the program memory. Hence, the instruction takes only one clock cycle to execute.

71. LD (LDD) – Load Indirect from Data Space to Register using Index Y

71.1. Description

Loads one byte indirect with or without displacement from the data space to a register. For parts with SRAM, the data space consists of the Register File, I/O memory, and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. In some parts the Flash Memory has been mapped to the data space and can be read using this command. The EEPROM has a separate address space.

The data location is pointed to by the Y (16 bits) Pointer Register in the Register File. Memory access is limited to the current data segment of 64KB. To access another data segment in devices with more than 64KB data space, the RAMPY in register in the I/O area has to be changed.

The Y-pointer Register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for accessing arrays, tables, and Stack Pointer usage of the Y-pointer Register. Note that only the low byte of the Y-pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPY Register in the I/O area is updated in parts with more than 64KB data space or more than 64KB Program memory, and the increment/decrement/displacement is added to the entire 24-bit address on such devices.

Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

In the Reduced Core tinyAVR the LD instruction can be used to achieve the same operation as LPM since the program memory is mapped to the data memory space.

The result of these combinations is undefined:

LD r28, Y+

LD r29, Y+

LD r28, -Y

LD r29, -Y

Using the Y-pointer:

	Operation:		Comment:
(i)	Rd ← (Y)		Y: Unchanged
(ii)	Rd ← (Y), Y ← Y + 1		Y: Post incremented
(iii)	Y ← Y - 1, Rd ← (Y)		Y: Pre decremented
(iv)	Rd ← (Y+q)		Y: Unchanged, q: Displacement
	Syntax:	Operands:	Program Counter:
(i)	LD Rd, Y	0 ≤ d ≤ 31	PC ← PC + 1
(ii)	LD Rd, Y+	0 ≤ d ≤ 31	PC ← PC + 1

(iii)	LD Rd, -Y	0 ≤ d ≤ 31	PC ← PC + 1
(iv)	LDD Rd, Y+q	0 ≤ d ≤ 31, 0 ≤ q ≤ 63	PC ← PC + 1

16-bit Opcode:

(i)	1000	000d	dddd	1000
(ii)	1001	000d	dddd	1001
(iii)	1001	000d	dddd	1010
(iv)	10q0	qq0d	dddd	1qqq

71.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr r29 ; Clear Y high byte
ldi r28,$60 ; Set Y low byte to $60
ld r0,Y+ ; Load r0 with data space loc. $60(Y post inc)
ld r1,Y ; Load r1 with data space loc. $61
ldi r28,$63 ; Set Y low byte to $63
ld r2,Y ; Load r2 with data space loc. $63
ld r3,-Y ; Load r3 with data space loc. $62(Y pre dec)
ldd r4,Y+2 ; Load r4 with data space loc. $64
```

Words	1 (2 bytes)
Cycles	(i): 1[2]
	(ii): 2
	(iii): 3[2]
Cycles XMEGA	(i): 1[1]
	(ii): 1[1]
	(iii): 2[1]
	(iv): 2[1]

1. Note: If the LD instruction is accessing internal SRAM, one extra cycle is inserted.
2. Note: LD instruction can load data from program memory since the flash is memory mapped. Loading data from the data memory takes one clock cycle, and loading from the program memory takes two clock cycles. But if an interrupt occur (before the last clock cycle) no additional clock cycles are necessary when loading from the program memory. Hence, the instruction takes only one clock cycle to execute.

   LD instruction with pre-decrement can load data from program memory since the flash is memory mapped. Loading data from the data memory takes two clock cycles, and loading from the program memory takes three clock cycles. But if an interrupt occur (before the last clock cycle) no additional

clock cycles are necessary when loading from the program memory. Hence, the instruction takes only one clock cycle to execute.

72. LD (LDD) – Load Indirect From Data Space to Register using Index Z

72.1. Description

Loads one byte indirect with or without displacement from the data space to a register. For parts with SRAM, the data space consists of the Register File, I/O memory, and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. In some parts the Flash Memory has been mapped to the data space and can be read using this command. The EEPROM has a separate address space.

The data location is pointed to by the Z (16 bits) Pointer Register in the Register File. Memory access is limited to the current data segment of 64KB. To access another data segment in devices with more than 64KB data space, the RAMPZ in register in the I/O area has to be changed.

The Z-pointer Register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for Stack Pointer usage of the Z-pointer Register, however, because the Z-pointer Register can be used for indirect subroutine calls, indirect jumps and table look-up, it is often more convenient to use the X- or Y-pointer as a dedicated Stack Pointer. Note that only the low byte of the Z-pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPZ Register in the I/O area is updated in parts with more than 64KB data space or more than 64KB Program memory, and the increment/decrement/displacement is added to the entire 24-bit address on such devices.

Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

In the tinyAVR reduced core the LD instruction can be used to achieve the same operation as LPM since the program memory is mapped to the data memory space.

For using the Z-pointer for table look-up in Program memory see the LPM and ELPM instructions.

The result of these combinations is undefined:

LD r30, Z+

LD r31, Z+

LD r30, -Z

LD r31, -Z

Using the Z-pointer:

	Operation:	Comment:
(i)	Rd ← (Z)	Z: Unchanged
(ii)	Rd ← (Z), Z ← Z + 1	Z: Post incremented
(iii)	Z ← Z - 1, Rd ← (Z)	Z: Pre decremented
(iv)	Rd ← (Z+q)	Z: Unchanged, q: Displacement

Syntax:	Operands:	Program Counter:

(i)	LD Rd, Z	0 ≤ d ≤ 31	PC ← PC + 1
(ii)	LD Rd, Z+	0 ≤ d ≤ 31	PC ← PC + 1
(iii)	LD Rd, -Z	0 ≤ d ≤ 31	PC ← PC + 1
(iv)	LDD Rd, Z+q	0 ≤ d ≤ 31, 0 ≤ q ≤ 63	PC ← PC + 1

16-bit Opcode:

(i)	1000	000d	dddd	0000
(ii)	1001	000d	dddd	0001
(iii)	1001	000d	dddd	0010
(iv)	10q0	qq0d	dddd	0qqq

72.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr r31 ; Clear Z high byte
ldi r30,$60 ; Set Z low byte to $60
ld r0,Z+ ; Load r0 with data space loc. $60(Z post inc)
ld r1,Z ; Load r1 with data space loc. $61
ldi r30,$63 ; Set Z low byte to $63
ld r2,Z ; Load r2 with data space loc. $63
ld r3,-Z ; Load r3 with data space loc. $62(Z pre dec)
ldd r4,Z+2 ; Load r4 with data space loc. $64
```

Words	1 (2 bytes)
Cycles	(i): 1[2]
	(ii): 2
	(iii): 3[2]
Cycles XMEGA	(i): 1[1]
	(ii): 1[1]
	(iii): 2[1]
	(iv): 2[1]

1. Note: If the LD instruction is accessing internal SRAM, one extra cycle is inserted.
2. Note: LD instruction can load data from program memory since the flash is memory mapped. Loading data from the data memory takes one clock cycle, and loading from the program memory takes two clock cycles. But if an interrupt occur (before the last clock cycle) no additional clock cycles are necessary when loading from the program memory. Hence, the instruction takes only one clock cycle to execute.

LD instruction with pre-decrement can load data from program memory since the flash is memory mapped. Loading data from the data memory takes two clock cycles, and loading from the program memory takes three clock cycles. But if an interrupt occur (before the last clock cycle) no additional clock cycles are necessary when loading from the program memory. Hence, the instruction takes only one clock cycle to execute.

73. LDI – Load Immediate

73.1. Description

Loads an 8-bit constant directly to register 16 to 31.

	Operation:		
(i)	Rd ← K		
	Syntax:	Operands:	Program Counter:
(i)	LDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

1110	KKKK	dddd	KKKK

73.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr r31 ; Clear Z high byte
ldi r30,$F0 ; Set Z low byte to $F0
lpm ; Load constant from Program
; memory pointed to by Z
```

Words	1 (2 bytes)
Cycles	1

74. LDS – Load Direct from Data Space

74.1. Description

Loads one byte from the data space to a register. For parts with SRAM, the data space consists of the Register File, I/O memory, and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the register file only. The EEPROM has a separate address space.

A 16-bit address must be supplied. Memory access is limited to the current data segment of 64KB. The LDS instruction uses the RAMPD Register to access memory above 64KB. To access another data segment in devices with more than 64KB data space, the RAMPD in register in the I/O area has to be changed.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

	Operation:		
(i)	Rd ← (k)		
	Syntax:	Operands:	Program Counter:
(i)	LDS Rd,k	0 ≤ d ≤ 31, 0 ≤ k ≤ 65535	PC ← PC + 2

32-bit Opcode:

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

74.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
lds r2,$FF00 ; Load r2 with the contents of data space location $FF00
add r2,r1 ; add r1 to r2
sts $FF00,r2 ; Write back
```

Words	2 (4 bytes)
Cycles	2
Cycles XMEGA	2 If the LDS instruction is accessing internal SRAM, one extra cycle is inserted

75. LDS (16-bit) – Load Direct from Data Space

75.1. Description

Loads one byte from the data space to a register. For parts with SRAM, the data space consists of the Register File, I/O memory, and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the register file only. In some parts the Flash memory has been mapped to the data space and can be read using this command. The EEPROM has a separate address space.

A 7-bit address must be supplied. The address given in the instruction is coded to a data space address as follows:

ADDR[7:0] = ($\overline{\text{INST[8]}}$, INST[8], INST[10], INST[9], INST[3], INST[2], INST[1], INST[0])

Memory access is limited to the address range 0x40...0xbf.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

	Operation:
(i)	Rd ← (k)

	Syntax:	Operands:	Program Counter:
(i)	LDS Rd,k	16 ≤ d ≤ 31, 0 ≤ k ≤ 127	PC ← PC + 1

16-bit Opcode:

1010	0kkk	dddd	kkkk

75.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
lds r16,$00 ; Load r16 with the contents of data space location $00
add r16,r17 ; add r17 to r16
sts $00,r16 ; Write result to the same address it was fetched from
```

Words	1 (2 bytes)
Cycles	1

Note: Registers r0...r15 are remapped to r16...r31.

76. LPM – Load Program Memory

76.1. Description

Loads one byte pointed to by the Z-register into the destination register Rd. This instruction features a 100% space effective constant initialization or constant data fetch. The Program memory is organized in 16-bit words while the Z-pointer is a byte address. Thus, the least significant bit of the Z-pointer selects either low byte (Z_{LSB} = 0) or high byte (Z_{LSB} = 1). This instruction can address the first 64KB (32K words) of Program memory. The Z-pointer Register can either be left unchanged by the operation, or it can be incremented. The incrementation does not apply to the RAMPZ Register.

Devices with Self-Programming capability can use the LPM instruction to read the Fuse and Lock bit values. Refer to the device documentation for a detailed description.

The LPM instruction is not available in all devices. Refer to the device specific instruction set summary.

The result of these combinations is undefined:

LPM r30, Z+

LPM r31, Z+

	Operation:	Comment:
(i)	R0 ← (Z)	Z: Unchanged, R0 implied destination register
(ii)	Rd ← (Z)	Z: Unchanged
(iii)	Rd ← (Z) Z ← Z + 1	Z: Post incremented

	Syntax:	Operands:	Program Counter:
(i)	LPM	None, R0 implied	PC ← PC + 1
(ii)	LPM Rd, Z	0 ≤ d ≤ 31	PC ← PC + 1
(iii)	LPM Rd, Z+	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

(i)	1001	0101	1100	1000
(ii)	1001	000d	dddd	0100
(iii)	1001	000d	dddd	0101

76.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
  ldi ZH, high(Table_1<<1) ; Initialize Z-pointer
  ldi ZL, low(Table_1<<1)
  lpm r16, Z ; Load constant from Program
  ; Memory pointed to by Z (r31:r30)
...
Table_1:
.dw 0x5876 ; 0x76 is addresses when ZLSB = 0
  ; 0x58 is addresses when ZLSB = 1
...
```

Words	1 (2 bytes)
Cycles	3

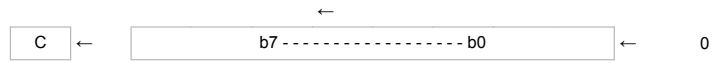
77. LSL – Logical Shift Left

77.1. Description

Shifts all bits in Rd one place to the left. Bit 0 is cleared. Bit 7 is loaded into the C Flag of the SREG. This operation effectively multiplies signed and unsigned values by two.

Operation:

(i)



	Syntax:	Operands:	Program Counter:
(i)	LSL Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode: (see ADD Rd,Rd)

0000	11dd	dddd	dddd

77.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H Rd3

S N ⊕ V, for signed tests.

V N ⊕ C, for N and C after the shift.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

C Rd7

Set if, before the shift, the MSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
add r0,r4 ; Add r4 to r0
lsl r0 ; Multiply r0 by 2
```

Words	1 (2 bytes)

Cycles	1

78. LSR – Logical Shift Right

78.1. Description

Shifts all bits in Rd one place to the right. Bit 7 is cleared. Bit 0 is loaded into the C Flag of the SREG. This operation effectively divides an unsigned value by two. The C Flag can be used to round the result.

Operation:

(i)

	Syntax:	Operands:	Program Counter:
(i)	LSR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0110

78.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	0	⇔	⇔

S N ⊕ V, for signed tests.

V N ⊕ C, for N and C after the shift.

N 0

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

C Rd0

Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
add r0,r4 ; Add r4 to r0
lsr r0 ; Divide r0 by 2
```

Words	1 (2 bytes)
Cycles	1

79. MOV – Copy Register

79.1. Description

This instruction makes a copy of one register into another. The source register Rr is left unchanged, while the destination register Rd is loaded with a copy of Rr.

Operation:

(i) Rd ← Rr

	Syntax:	Operands:	Program Counter:
(i)	MOV Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0010	11rd	dddd	rrrr

79.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
mov r16,r0 ; Copy r0 to r16
call check ; Call subroutine
...
check: cpi r16,$11 ; Compare r16 to $11
...
ret ; Return from subroutine
```

Words	1 (2 bytes)
Cycles	1

80. MOVW – Copy Register Word

80.1. Description

This instruction makes a copy of one register pair into another register pair. The source register pair Rr +1:Rr is left unchanged, while the destination register pair Rd+1:Rd is loaded with a copy of Rr + 1:Rr.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) Rd+1:Rd ← Rr+1:Rr

	Syntax:	Operands:	Program Counter:
(i)	MOVW Rd+1:Rd,Rr+1:Rr	d ∈ {0,2,...,30}, r ∈ {0,2,...,30}	PC ← PC + 1

16-bit Opcode:

0000	0001	dddd	rrrr

80.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
mov w r17:16,r1:r0 ; Copy r1:r0 to r17:r16
call check ; Call subroutine
...
check: cpi r16,$11 ; Compare r16 to $11
...
cpi r17,$32 ; Compare r17 to $32
...
ret ; Return from subroutine
```

Words	1 (2 bytes)
Cycles	1

81. MUL – Multiply Unsigned

81.1. Description

This instruction performs 8-bit × 8-bit → 16-bit unsigned multiplication.

Rd		Rr		R1	R0
Multiplicand	×	Multiplier	→	Product High	Product Low
8		8		16	

The multiplicand Rd and the multiplier Rr are two registers containing unsigned numbers. The 16-bit unsigned product is placed in R1 (high byte) and R0 (low byte). Note that if the multiplicand or the multiplier is selected from R0 or R1 the result will overwrite those after multiplication.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) R1:R0 ← Rd × Rr (unsigned ← unsigned × unsigned)

	Syntax:	Operands:	Program Counter:
(i)	MUL Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	11rd	dddd	rrrr

81.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	⇔	⇔

C R15

Z $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8}R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $0000; cleared otherwise.

R (Result) equals R1,R0 after the operation.

Example:

```
mul r5,r4 ; Multiply unsigned r5 and r4
mov w r4,r0 ; Copy result back in r5:r4
```

Words	1 (2 bytes)
Cycles	1

82. MULS – Multiply Signed

82.1. Description

This instruction performs 8-bit × 8-bit → 16-bit signed multiplication.

Rd		Rr		R1	R0
Multiplicand	×	Multiplier	→	Product High	Product Low
8		8		16	

The multiplicand Rd and the multiplier Rr are two registers containing signed numbers. The 16-bit signed product is placed in R1 (high byte) and R0 (low byte).

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) R1:R0 ← Rd × Rr (signed ← signed × signed)

	Syntax:	Operands:	Program Counter:
(i)	MULS Rd,Rr	16 ≤ d ≤ 31, 16 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0000	0010	dddd	rrrr

82.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	⇔	⇔

C R15

Z $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8}R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $0000; cleared otherwise.

R (Result) equals R1,R0 after the operation.

Example:

```
muls r21,r20 ; Multiply signed r21 and r20
mov w r20,r0 ; Copy result back in r21:r20
```

Words	1 (2 bytes)
Cycles	1

83. MULSU – Multiply Signed with Unsigned

83.1. Description

This instruction performs 8-bit × 8-bit → 16-bit multiplication of a signed and an unsigned number.

Rd	Rr		R1	R0
Multiplicand	Multiplier	→	Product High	Product Low
8	8		16	

The multiplicand Rd and the multiplier Rr are two registers. The multiplicand Rd is a signed number, and the multiplier Rr is unsigned. The 16-bit signed product is placed in R1 (high byte) and R0 (low byte).

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) R1:R0 ← Rd × Rr (signed ← signed × unsigned)

	Syntax:	Operands:	Program Counter:
(i)	MULSU Rd,Rr	16 ≤ d ≤ 23, 16 ≤ r ≤ 23	PC ← PC + 1

16-bit Opcode:

0000	0011	0ddd	0rrr

83.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	⇔	⇔

C R15

Z $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8}R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $0000; cleared otherwise.

R (Result) equals R1,R0 after the operation.

Example:

```
;***************************************************************************
;* DESCRIPTION
;* Signed multiply of two 16-bit numbers with 32-bit result.
;* USAGE
;* r19:r18:r17:r16 = r23:r22 * r21:r20
;***************************************************************************
muls16x16_32:
 clr r2
 muls r23, r21 ; (signed)ah * (signed)bh
 movw r19:r18, r1:r0
 mul r22, r20 ; al * bl
 movw r17:r16, r1:r0
 mulsu r23, r20 ; (signed)ah * bl
```



```
 sbc r19, r2
 add r17, r0
 adc r18, r1
 adc r19, r2
 mulsu r21, r22 ; (signed)bh * al
 sbc r19, r2
 add r17, r0
 adc r18, r1
 adc r19, r2
 ret
```

Words	1 (2 bytes)
Cycles	2

84. NEG – Two's Complement

84.1. Description

Replaces the contents of register Rd with its two's complement; the value $80 is left unchanged.

Operation:

(i) Rd ← $00 - Rd

	Syntax:	Operands:	Program Counter:
(i)	NEG Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0001

84.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H $P3 + \overline{Rd3}$

Set if there was a borrow from bit 3; cleared otherwise.

S N ⊕ V, for signed tests.

V $R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if there is a two's complement overflow from the implied subtraction from zero; cleared otherwise. A two's complement overflow will occur if and only if the contents of the Register after operation (Result) is $80.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

C R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0

Set if there is a borrow in the implied subtraction from zero; cleared otherwise. The C Flag will be set in all cases except when the contents of Register after operation is $00.

R (Result) equals Rd after the operation.

Example:

```
sub r11,r0 ; Subtract r0 from r11
brpl positive ; Branch if result positive
```



```
neg r11 ; Take two's complement of r11
positive: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1

85. NOP – No Operation

85.1. Description

This instruction performs a single cycle No Operation.

	Operation:		
(i)	No		
	Syntax:	Operands:	Program Counter:
(i)	NOP	None	PC ← PC + 1

16-bit Opcode:

0000	0000	0000	0000

85.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr r16 ; Clear r16
ser r17 ; Set r17
out $18,r16 ; Write zeros to Port B
nop ; Wait (do nothing)
out $18,r17 ; Write ones to Port B
```

Words	1 (2 bytes)
Cycles	1

86. OR – Logical OR

86.1. Description

Performs the logical OR between the contents of register Rd and register Rr, and places the result in the destination register Rd.

	Operation:		
(i)	Rd ← Rd v Rr		
	Syntax:	Operands:	Program Counter:
(i)	OR Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0010	10rd	dddd	rrrr

86.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S N ⊕ V, for signed tests.

V 0

Cleared.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
or r15,r16 ; Do bitwise or between registers
bst r15,6 ; Store bit 6 of r15 in T Flag
brts ok ; Branch if T Flag set
...
ok: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1

87. ORI – Logical OR with Immediate

87.1. Description

Performs the logical OR between the contents of register Rd and a constant, and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd v K

Syntax:	Operands:	Program Counter:
(i) ORI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

0110	KKKK	dddd	KKKK

87.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S N ⊕ V, for signed tests.

V 0

Cleared.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
ori r16,$F0 ; Set high nibble of r16
ori r17,1 ; Set bit 0 of r17
```

Words	1 (2 bytes)
Cycles	1

88. OUT – Store Register to I/O Location

88.1. Description

Stores data from register Rr in the Register File to I/O Space (Ports, Timers, Configuration Registers, etc.).

Operation:

(i) I/O(A) ← Rr

Syntax:	Operands:	Program Counter:
(i) OUT A,Rr	0 ≤ r ≤ 31, 0 ≤ A ≤ 63	PC ← PC + 1

16-bit Opcode:

1011	1AAr	rrrr	AAAA

88.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr r16 ; Clear r16
ser r17 ; Set r17
out $18,r16 ; Write zeros to Port B
nop ; Wait (do nothing)
out $18,r17 ; Write ones to Port B
```

Words	1 (2 bytes)
Cycles	1

89. POP – Pop Register from Stack

89.1. Description

This instruction loads register Rd with a byte from the STACK. The Stack Pointer is pre-incremented by 1 before the POP.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) Rd ← STACK

	Syntax:	Operands:	Program Counter:	Stack:
(i)	POP Rd	0 ≤ d ≤ 31	PC ← PC + 1	SP ← SP + 1

16-bit Opcode:

1001	000d	dddd	1111

89.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 call routine ; Call subroutine
 ...
routine: push r14 ; Save r14 on the Stack
 push r13 ; Save r13 on the Stack
 ...
 pop r13 ; Restore r13
 pop r14 ; Restore r14
 ret ; Return from subroutine
```

Words	1 (2 bytes)
Cycles	2

90. PUSH – Push Register on Stack

90.1. Description

This instruction stores the contents of register Rr on the STACK. The Stack Pointer is post-decremented by 1 after the PUSH.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) STACK ← Rr

	Syntax:	Operands:	Program Counter:	Stack:
(i)	PUSH Rr	0 ≤ r ≤ 31	PC ← PC + 1	SP ← SP - 1

16-bit Opcode:

1001	001d	dddd	1111

90.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 call routine ; Call subroutine
 ...
routine: push r14 ; Save r14 on the Stack
 push r13 ; Save r13 on the Stack
 ...
 pop r13 ; Restore r13
 pop r14 ; Restore r14
 ret ; Return from subroutine
```

Words	1 (2 bytes)
Cycles	2
Cycles XMEGA	1

91. RCALL – Relative Call to Subroutine

91.1. Description

Relative call to an address within PC - 2K + 1 and PC + 2K (words). The return address (the instruction after the RCALL) is stored onto the Stack. See also CALL. For AVR microcontrollers with Program memory not exceeding 4K words (8KB) this instruction can address the entire memory from every address location. The Stack Pointer uses a post-decrement scheme during RCALL.

	Operation:	Comment:
(i)	PC ← PC + k + 1	Devices with 16-bit PC, 128KB Program memory maximum.
(ii)	PC ← PC + k + 1	Devices with 22-bit PC, 8MB Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i)	RCALL k	-2K ≤ k < 2K	PC ← PC + k + 1	STACK ← PC + 1 SP ← SP - 2 (2 bytes, 16 bits)
(ii)	RCALL k	-2K ≤ k < 2K	PC ← PC + k + 1	STACK ← PC + 1 SP ← SP - 3 (3 bytes, 22 bits)

16-bit Opcode:

1101	kkkk	kkkk	kkkk

91.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 rcall routine ; Call subroutine
 ...
routine: push r14 ; Save r14 on the Stack
 ...
 pop r14 ; Restore r14
 ret ; Return from subroutine
```

Words	1 (2 bytes)
Cycles	3 devices with 16-bit PC
	4 devices with 22-bit PC
Cycles XMEGA	2 devices with 16-bit PC
	3 devices with 22-bit PC
Cycles Reduced Core tinyAVR	4

92. RET – Return from Subroutine

92.1. Description

Returns from subroutine. The return address is loaded from the STACK. The Stack Pointer uses a pre-increment scheme during RET.

Operation:

	Operation:	Comment:
(i)	PC(15:0) ← STACK	Devices with 16-bit PC, 128KB Program memory maximum.
(ii)	PC(21:0) ← STACK	Devices with 22-bit PC, 8MB Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i)	RET	None	See Operation	SP ← SP + 2, (2 bytes, 16 bits)
(ii)	RET	None	See Operation	SP ← SP + 3, (3 bytes, 22 bits)

16-bit Opcode:

1001	0101	0000	1000

92.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 call routine ; Call subroutine
 ...
routine: push r14 ; Save r14 on the Stack
 ...
 pop r14 ; Restore r14
 ret ; Return from subroutine
```

Words	1 (2 bytes)
Cycles	4 devices with 16-bit PC
	5 devices with 22-bit PC

93. RETI – Return from Interrupt

93.1. Description

Returns from interrupt. The return address is loaded from the STACK and the Global Interrupt Flag is set.

Note that the Status Register is not automatically stored when entering an interrupt routine, and it is not restored when returning from an interrupt routine. This must be handled by the application program. The Stack Pointer uses a pre-increment scheme during RETI.

	Operation:	Comment:
(i)	PC(15:0) ← STACK	Devices with 16-bit PC, 128KB Program memory maximum.
(ii)	PC(21:0) ← STACK	Devices with 22-bit PC, 8MB Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i)	RETI	None	See Operation	SP ← SP + 2 (2 bytes, 16 bits)
(ii)	RETI	None	See Operation	SP ← SP + 3 (3 bytes, 22 bits)

16-bit Opcode:

1001	0101	0001	1000

93.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
1	–	–	–	–	–	–	–

I 1

The I Flag is set.

Example:

```
 ...
extint: push r0 ; Save r0 on the Stack
 ...
 pop r0 ; Restore r0
 reti ; Return and enable interrupts
```

Words	1 (2 bytes)
Cycles	4 devices with 16-bit PC
	5 devices with 22-bit PC

Note: RETI behaves differently in megaAVR and AVR XMEGA devices. In the megaAVR series of devices, the global interrupt flag is cleared by hardware once an interrupt occurs and this bit is set when RETI is executed. In the AVR XMEGA devices, RETI will not modify the global interrupt flag in SREG

since it is not cleared by hardware while entering ISR. This bit should be modified using SEI and CLI instructions when needed.

94. RJMP – Relative Jump

94.1. Description

Relative jump to an address within PC - 2K +1 and PC + 2K (words). For AVR microcontrollers with Program memory not exceeding 4K words (8KB) this instruction can address the entire memory from every address location. See also JMP.

	Operation:			
(i)	PC ← PC + k + 1			
	Syntax:	Operands:	Program Counter:	Stack:
(i)	RJMP	k -2K ≤ k < 2K	PC ← PC + k + 1	Unchanged

16-bit Opcode:

1100	kkkk	kkkk	kkkk

94.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
 cpi r16,$42 ; Compare r16 to $42
 brne error ; Branch if r16 <> $42
 rjmp ok ; Unconditional branch
error: add r16,r17 ; Add r17 to r16
 inc r16 ; Increment r16
ok: nop ; Destination for rjmp (do nothing)
```

Words	1 (2 bytes)
Cycles	2

95. ROL – Rotate Left trough Carry

95.1. Description

Shifts all bits in Rd one place to the left. The C Flag is shifted into bit 0 of Rd. Bit 7 is shifted into the C Flag. This operation, combined with LSL, effectively multiplies multi-byte signed and unsigned values by two.

Operation:

	Syntax:	Operands:	Program Counter:
(i)	ROL Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode: (see ADC Rd,Rd)

0001	11dd	dddd	dddd

95.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H Rd3

S N ⊕ V, for signed tests.

V N ⊕ C, for N and C after the shift.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

C Rd7

Set if, before the shift, the MSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
  lsl r18 ; Multiply r19:r18 by two
  rol r19 ; r19:r18 is a signed or unsigned two-byte integer
  brcs oneenc ; Branch if carry set
  ...
oneenc: nop ; Branch destination (do nothing)
```


Words	1 (2 bytes)
Cycles	1

96. ROR – Rotate Right through Carry

96.1. Description

Shifts all bits in Rd one place to the right. The C Flag is shifted into bit 7 of Rd. Bit 0 is shifted into the C Flag. This operation, combined with ASR, effectively divides multi-byte signed values by two. Combined with LSR it effectively divides multi-byte unsigned values by two. The Carry Flag can be used to round the result.

Operation:

	Syntax:	Operands:	Program Counter:
(i)	ROR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0111

96.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S N ⊕ V, for signed tests.

V N ⊕ C, for N and C after the shift.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

C Rd0

Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
lsr r19 ; Divide r19:r18 by two
ror r18 ; r19:r18 is an unsigned two-byte integer
brcc zeroenc1 ; Branch if carry cleared
asr r17 ; Divide r17:r16 by two
ror r16 ; r17:r16 is a signed two-byte integer
brcc zeroenc2 ; Branch if carry cleared
...
```



```
zeroenc1: nop ; Branch destination (do nothing)
 ...
zeroenc1: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1

97. SBC – Subtract with Carry

97.1. Description

Subtracts two registers and subtracts with the C Flag, and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd - Rr - C

	Syntax:	Operands:	Program Counter:
(i)	SBC Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0000	10rd	dddd	rrrr

97.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$

Set if there was a borrow from bit 3; cleared otherwise.

S N ⊕ V, for signed tests.

V $Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$

Set if two's complement overflow resulted from the operation; cleared otherwise.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0} \bullet Z$

Previous value remains unchanged when the result is zero; cleared otherwise.

C $\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$

Set if the absolute value of the contents of Rr plus previous carry is larger than the absolute value of the Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
; Subtract r1:r0 from r3:r2
sub r2,r0 ; Subtract low byte
sbc r3,r1 ; Subtract with carry high byte
```


Words	1 (2 bytes)
Cycles	1

98. SBCI – Subtract Immediate with Carry SBI – Set Bit in I/O Register

98.1. Description

Subtracts a constant from a register and subtracts with the C Flag, and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd - K - C

	Syntax:	Operands:	Program Counter:
(i)	SBCI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

0100	KKKK	dddd	KKKK

98.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H $\overline{Rd3} \bullet K3 + K3 \bullet R3 + R3 \bullet \overline{Rd3}$

Set if there was a borrow from bit 3; cleared otherwise.

S N ⊕ V, for signed tests.

V $Rd7 \bullet \overline{K7} \bullet \overline{R7} + \overline{Rd7} \bullet K7 \bullet R7$

Set if two's complement overflow resulted from the operation; cleared otherwise.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0} \bullet Z$

Previous value remains unchanged when the result is zero; cleared otherwise.

C $\overline{Rd7} \bullet K7 + K7 \bullet R7 + R7 \bullet \overline{Rd7}$

Set if the absolute value of the constant plus previous carry is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
; Subtract $4F23 from r17:r16
subi r16,$23 ; Subtract low byte
sbci r17,$4F ; Subtract with carry high byte
```


Words	1 (2 bytes)
Cycles	1

99. SBI – Set Bit in I/O Register

99.1. Description

Sets a specified bit in an I/O Register. This instruction operates on the lower 32 I/O Registers – addresses 0-31.

Operation:

(i) I/O(A,b) ← 1

	Syntax:	Operands:	Program Counter:
(i)	SBI A,b	0 ≤ A ≤ 31, 0 ≤ b ≤ 7	PC ← PC + 1

16-bit Opcode:

1001	1010	AAAA	Abbb

99.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
out $1E,r0 ; Write EEPROM address
sbi $1C,0 ; Set read bit in EECR
in r1,$1D ; Read EEPROM data
```

Words	1 (2 bytes)
Cycles	2
Cycles XMEGA	1
Cycles Reduced Core tinyAVR	1

100. SBIC – Skip if Bit in I/O Register is Cleared

100.1. Description

This instruction tests a single bit in an I/O Register and skips the next instruction if the bit is cleared. This instruction operates on the lower 32 I/O Registers – addresses 0-31.

Operation:

(i) If I/O(A,b) = 0 then PC ← PC + 2 (or 3) else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	SBIC A,b	0 ≤ A ≤ 31, 0 ≤ b ≤ 7	PC ← PC + 1, Condition false - no skip PC ← PC + 2, Skip a one word instruction PC ← PC + 3, Skip a two word instruction

16-bit Opcode:

1001	1001	AAAA	Abbb

100.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
e2wait: sbic $1C,1 ; Skip next inst. if EEWE cleared
rjmp e2wait ; EEPROM write not finished
nop ; Continue (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false (no skip) 2 if condition is true (skip is executed) and the instruction skipped is 1 word 3 if condition is true (skip is executed) and the instruction skipped is 2 words
Cycles XMEGA	2 if condition is false (no skip) 3 if condition is true (skip is executed) and the instruction skipped is 1 word 4 if condition is true (skip is executed) and the instruction skipped is 2 words

101. SBIS – Skip if Bit in I/O Register is Set

101.1. Description

This instruction tests a single bit in an I/O Register and skips the next instruction if the bit is set. This instruction operates on the lower 32 I/O Registers – addresses 0-31.

Operation:

(i) If I/O(A,b) = 1 then PC ← PC + 2 (or 3) else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	SBIS A,b	0 ≤ A ≤ 31, 0 ≤ b ≤ 7	PC ← PC + 1, Condition false - no skip PC ← PC + 2, Skip a one word instruction PC ← PC + 3, Skip a two word instruction

16-bit Opcode:

1001	1011	AAAA	Abbb

101.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
waitset: sbis $10,0 ; Skip next inst. if bit 0 in Port D set
 rjmp waitset ; Bit not set
 nop ; Continue (do nothing)
```

Words	1 (2 bytes)
Cycles	1 if condition is false (no skip)
	2 if condition is true (skip is executed) and the instruction skipped is 1 word
	3 if condition is true (skip is executed) and the instruction skipped is 2 words
Cycles XMEGA	2 if condition is false (no skip)
	3 if condition is true (skip is executed) and the instruction skipped is 1 word
	4 if condition is true (skip is executed) and the instruction skipped is 2 words

102. SBIW – Subtract Immediate from Word

102.1. Description

Subtracts an immediate value (0-63) from a register pair and places the result in the register pair. This instruction operates on the upper four register pairs, and is well suited for operations on the Pointer Registers.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) Rd+1:Rd ← Rd+1:Rd - K

	Syntax:	Operands:	Program Counter:
(i)	SBIW Rd+1:Rd,K	d ∈ {24,26,28,30}, 0 ≤ K ≤ 63	PC ← PC + 1

16-bit Opcode:

1001	0111	KKdd	KKKK

102.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S N ⊕ V, for signed tests.

V R15 • $\overline{Rdh7}$

Set if two's complement overflow resulted from the operation; cleared otherwise.

N R15

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R15}$ • $\overline{R14}$ • $\overline{R13}$ • $\overline{R12}$ • $\overline{R11}$ • $\overline{R10}$ • $\overline{R9}$ • $\overline{R8R7}$ • $\overline{R6}$ • $\overline{R5}$ • $\overline{R4}$ • $\overline{R3}$ • $\overline{R2}$ • $\overline{R1}$ • $\overline{R0}$

Set if the result is $0000; cleared otherwise.

C R15 • $\overline{Rdh7}$

Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rdh:Rdl after the operation (Rdh7-Rdh0 = R15-R8, Rdl7-Rdl0=R7-R0).

Example:

```
sbiw r25:r24,1 ; Subtract 1 from r25:r24
sbiw YH:YL,63 ; Subtract 63 from the Y-pointer(r29:r28)
```

Words	1 (2 bytes)

Cycles 2

103. SBR – Set Bits in Register

103.1. Description

Sets specified bits in register Rd. Performs the logical ORI between the contents of register Rd and a constant mask K, and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd v K

Syntax:	Operands:	Program Counter:
(i) SBR Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

0110	KKKK	dddd	KKKK

103.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S N ⊕ V, for signed tests.

V 0

Cleared.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
sbr r16,3 ; Set bits 0 and 1 in r16
sbr r17,$F0 ; Set 4 MSB in r17
```

Words	1 (2 bytes)
Cycles	1

104. SBRC – Skip if Bit in Register is Cleared

104.1. Description

This instruction tests a single bit in a register and skips the next instruction if the bit is cleared.

Operation:

Operation:

(i) If Rr(b) = 0 then PC ← PC + 2 (or 3) else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	SBRC Rr,b	0 ≤ r ≤ 31, 0 ≤ b ≤ 7	PC ← PC + 1, Condition false - no skip PC ← PC + 2, Skip a one word instruction PC ← PC + 3, Skip a two word instruction

16-bit Opcode:

1111	110r	rrrr	0bbb

104.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
sub r0,r1 ; Subtract r1 from r0
sbrc r0,7 ; Skip if bit 7 in r0 cleared
sub r0,r1 ; Only executed if bit 7 in r0 not cleared
nop ; Continue (do nothing)
```

Words 1 (2 bytes)

Cycles 1 if condition is false (no skip)

2 if condition is true (skip is executed) and the instruction skipped is 1 word

3 if condition is true (skip is executed) and the instruction skipped is 2 words

105. SBRS – Skip if Bit in Register is Set

105.1. Description

This instruction tests a single bit in a register and skips the next instruction if the bit is set.

Operation:

(i) If Rr(b) = 1 then PC ← PC + 2 (or 3) else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	SBRS Rr,b	0 ≤ r ≤ 31, 0 ≤ b ≤ 7	PC ← PC + 1, Condition false - no skip PC ← PC + 2, Skip a one word instruction PC ← PC + 3, Skip a two word instruction

16-bit Opcode:

1111	111r	rrrr	0bbb

105.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
sub r0,r1 ; Subtract r1 from r0
sbrs r0,7 ; Skip if bit 7 in r0 set
neg r0 ; Only executed if bit 7 in r0 not set
nop ; Continue (do nothing)
```

Words 1 (2 bytes)

Cycles 1 if condition is false (no skip)

2 if condition is true (skip is executed) and the instruction skipped is 1 word

3 if condition is true (skip is executed) and the instruction skipped is 2 words

106. SEC – Set Carry Flag

106.1. Description

Sets the Carry Flag (C) in SREG (Status Register).

	Operation:		
(i)	C ← 1		
	Syntax:	**Operands:**	**Program Counter:**
(i)	SEC	None	PC ← PC + 1

16-bit Opcode:

1001	0100	0000	1000

106.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	1

C 1

Carry Flag set.

Example:

```
sec ; Set Carry Flag
adc r0,r1 ; r0=r0+r1+1
```

Words	1 (2 bytes)
Cycles	1

107. SEH – Set Half Carry Flag

107.1. Description

Sets the Half Carry (H) in SREG (Status Register).

	Operation:		
(i)	H ← 1		
	Syntax:	**Operands:**	**Program Counter:**
(i)	SEH	None	PC ← PC + 1

16-bit Opcode:

1001	0100	0101	1000

107.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	1	–	–	–	–	–

H 1

Half Carry Flag set.

Example:

```
seh ; Set Half Carry Flag
```

Words	1 (2 bytes)
Cycles	1

108. SEI – Set Global Interrupt Flag

108.1. Description

Sets the Global Interrupt Flag (I) in SREG (Status Register). The instruction following SEI will be executed before any pending interrupts.

Operation:

(i) I ← 1

	Syntax:	Operands:	Program Counter:
(i)	SEI	None	PC ← PC + 1

16-bit Opcode:

1001	0100	0111	1000

108.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
1	–	–	–	–	–	–	–

I 1

Global Interrupt Flag set.

Example:

```
sei ; set global interrupt enable
sleep ; enter sleep, waiting for interrupt
; note: will enter sleep before any pending interrupt(s)
```

Words	1 (2 bytes)
Cycles	1

109. SEN – Set Negative Flag

109.1. Description

Sets the Negative Flag (N) in SREG (Status Register).

Operation:

(i) N ← 1

	Syntax:	Operands:	Program Counter:
(i)	SEN	None	PC ← PC + 1

16-bit Opcode:

1001	0100	0010	1000

109.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	1	–	–

N 1

Negative Flag set.

Example:

```
add r2,r19 ; Add r19 to r2
sen ; Set Negative Flag
```

Words	1 (2 bytes)
Cycles	1

110. SER – Set all Bits in Register

110.1. Description

Loads $FF directly to register Rd.

Operation:

(i) Rd ← $FF

	Syntax:	Operands:	Program Counter:
(i)	SER Rd	16 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1110	1111	dddd	1111

110.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr r16 ; Clear r16
ser r17 ; Set r17
out $18,r16 ; Write zeros to Port B
nop ; Delay (do nothing)
out $18,r17 ; Write ones to Port B
```

Words	1 (2 bytes)
Cycles	1

111. SES – Set Signed Flag

111.1. Description

Sets the Signed Flag (S) in SREG (Status Register).

Operation:

(i) S ← 1

	Syntax:	Operands:	Program Counter:
(i)	SES	None	PC ← PC + 1

16-bit Opcode:

1001	0100	0100	1000

111.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	1	–	–	–	–

S 1

Signed Flag set.

Example:

```
add r2,r19 ; Add r19 to r2
ses ; Set Negative Flag
```

Words	1 (2 bytes)
Cycles	1

112. SET – Set T Flag

112.1. Description

Sets the T Flag in SREG (Status Register).

Operation:

(i) T ← 1

	Syntax:	Operands:	Program Counter:
(i)	SET	None	PC ← PC + 1

16-bit Opcode:

1001	0100	0110	1000

112.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	1	–	–	–	–	–	–

T 1

T Flag set.

Example:

```
set ; Set T Flag
```

Words	1 (2 bytes)
Cycles	1

113. SEV – Set Overflow Flag

113.1. Description

Sets the Overflow Flag (V) in SREG (Status Register).

Operation:

(i) V ← 1

	Syntax:	Operands:	Program Counter:
(i)	SEV	None	PC ← PC + 1

16-bit Opcode:

1001	0100	0011	1000

113.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	1	–	–	–

V V: 1

Overflow Flag set.

Example:

```
add r2,r19 ; Add r19 to r2
sev ; Set Overflow Flag
```

Words	1 (2 bytes)
Cycles	1

114. SEZ – Set Zero Flag

114.1. Description

Sets the Zero Flag (Z) in SREG (Status Register).

Operation:

(i) Z ← 1

Syntax:	Operands:	Program Counter:
(i) SEZ	None	PC ← PC + 1

16-bit Opcode:

1001	0100	0001	1000

114.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	1	–

Z 1

Zero Flag set.

Example:

```
add r2,r19 ; Add r19 to r2
sez ; Set Zero Flag
```

Words	1 (2 bytes)
Cycles	1

115. SLEEP

115.1. Description

This instruction sets the circuit in sleep mode defined by the MCU Control Register.

Operation:

(i) Refer to the device documentation for detailed description of SLEEP usage.

Syntax:	Operands:	Program Counter:
(i) SLEEP	None	PC ← PC + 1

16-bit Opcode:

1001	0101	1000	1000

115.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
mov r0,r11 ; Copy r11 to r0
ldi r16,(1<<SE) ; Enable sleep mode
out MCUCR, r16
sleep ; Put MCU in sleep mode
```

Words	1 (2 bytes)
Cycles	1

116. SPM – Store Program Memory

116.1. Description

SPM can be used to erase a page in the Program memory, to write a page in the Program memory (that is already erased), and to set Boot Loader Lock bits. In some devices, the Program memory can be written one word at a time, in other devices an entire page can be programmed simultaneously after first filling a temporary page buffer. In all cases, the Program memory must be erased one page at a time. When erasing the Program memory, the RAMPZ and Z-register are used as page address. When writing the Program memory, the RAMPZ and Z-register are used as page or word address, and the R1:R0 register pair is used as data(1). When setting the Boot Loader Lock bits, the R1:R0 register pair is used as data. Refer to the device documentation for detailed description of SPM usage. This instruction can address the entire Program memory.

The SPM instruction is not available in all devices. Refer to the device specific instruction set summary.

Note: 1. R1 determines the instruction high byte, and R0 determines the instruction low byte.

	Operation:		
(i)	(RAMPZ:Z) ← $ffff	Erase Program memory page	
(ii)	(RAMPZ:Z) ← R1:R0	Write Program memory word	
(iii)	(RAMPZ:Z) ← R1:R0	Write temporary page buffer	
(iv)	(RAMPZ:Z) ← TEMP	Write temporary page buffer to Program memory	
(v)	BLBITS ← R1:R0	Set Boot Loader Lock bits	
	Syntax:	Operands:	Program Counter:
(i)-(v)	SPM	Z+	PC ← PC + 1

16-bit Opcode:

1001	0101	1110	1000

116.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
;This example shows SPM write of one page for devices with page write
;- the routine writes one page of data from RAM to Flash
; the first data location in RAM is pointed to by the Y-pointer
; the first data location in Flash is pointed to by the Z-pointer
;- error handling is not included
;- the routine must be placed inside the boot space
; (at least the do_spm sub routine)
;- registers used: r0, r1, temp1, temp2, looplo, loophi, spmcrval
; (temp1, temp2, looplo, loophi, spmcrval must be defined by the user)
; storing and restoring of registers is not included in the routine
; register usage can be optimized at the expense of code size

.equ PAGESIZEB = PAGESIZE*2 ;PAGESIZEB is page size in BYTES, not words
.org SMALLBOOTSTART
write_page:
 ;page erase
 ldi spmcrval, (1<<PGERS) + (1<<SPMEN)
 call do_spm
 ;transfer data from RAM to Flash page buffer
 ldi looplo, low(PAGESIZEB) ;init loop variable
 ldi loophi, high(PAGESIZEB) ;not required for PAGESIZEB<=256
wrloop: ld r0, Y+
 ld r1, Y+
 ldi spmcrval, (1<<SPMEN)
 call do_spm
 adiw ZH:ZL, 2
 sbiw loophi:looplo, 2 ;use subi for PAGESIZEB<=256
 brne wrloop

 ;execute page write
 subi ZL, low(PAGESIZEB);restore pointer
 sbci ZH, high(PAGESIZEB) ;not required for PAGESIZEB<=256
 ldi spmcrval, (1<<PGWRT) + (1<<SPMEN)
 call do_spm

 ;read back and check, optional
 ldi looplo, low(PAGESIZEB) ;init loop variable
 ldi loophi, high(PAGESIZEB) ;not required for PAGESIZEB<=256
 subi YL, low(PAGESIZEB) ;restore pointer
 sbci YH, high(PAGESIZEB)
rdloop: lpm r0, Z+
 ld r1, Y+
 cpse r0, r1
 jmp error
 sbiw loophi:looplo, 2 ;use subi for PAGESIZEB<=256
 brne rdloop

 ;return
 ret

do_spm:
 ;input: spmcrval determines SPM action
 ;disable interrupts if enabled, store status
 in temp2, SREG
 cli
 ;check for previous SPM complete
wait: in temp1, SPMCR
 sbrc temp1, SPMEN
 rjmp wait
 ;SPM timed sequence
 out SPMCR, spmcrval
 spm
 ;restore SREG (to enable interrupts if originally enabled)
 out SREG, temp2
 ret
```

Words	1 (2 bytes)
Cycles	Depends on the operation

117. SPM #2 – Store Program Memory

117.1. Description

SPM can be used to erase a page in the Program memory and to write a page in the Program memory (that is already erased). An entire page can be programmed simultaneously after first filling a temporary page buffer. The Program memory must be erased one page at a time. When erasing the Program memory, the RAMPZ and Z-register are used as page address. When writing the Program memory, the RAMPZ and Z-register are used as page or word address, and the R1:R0 register pair is used as data(1).

Refer to the device documentation for detailed description of SPM usage. This instruction can address the entire Program memory.

Note: 1. R1 determines the instruction high byte, and R0 determines the instruction low byte.

	Operation:	
(i)	(RAMPZ:Z) ← $ffff	Erase Program memory page
(ii)	(RAMPZ:Z) ← R1:R0	Load Page Buffer
(iii)	(RAMPZ:Z) ← BUFFER	Write Page Buffer to Program memory
(iv)	(RAMPZ:Z) ← $fff, Z ← Z + 2	Erase Program memory page, Z post incremented
(v)	BLBITS ← R1:R0, Z ← Z + 2	Load Page Buffer, Z post incremented
(vi)	(RAMPZ:Z) ←BUFFER, Z ← Z + 2	Write Page Buffer to Program memory, Z post incremented

	Syntax:	Operands:	Program Counter:
(i)-(iii)	SPM	None	PC ← PC + 1
(iv)-(vi)	SPM Z+	None	PC ← PC + 1

16-bit Opcode:

(i)-(iii)	1001	0101	1110	1000
(iv)-(vi)	1001	0101	1111	1000

117.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words 1 (2 bytes)

Cycles Depends on the operation

118. ST – Store Indirect From Register to Data Space using Index X

118.1. Description

Stores one byte indirect from a register to data space. For parts with SRAM, the data space consists of the Register File, I/O memory, and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. The EEPROM has a separate address space.

The data location is pointed to by the X (16 bits) Pointer Register in the Register File. Memory access is limited to the current data segment of 64KB. To access another data segment in devices with more than 64KB data space, the RAMPX in register in the I/O area has to be changed.

The X-pointer Register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for accessing arrays, tables, and Stack Pointer usage of the X-pointer Register. Note that only the low byte of the X-pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPX Register in the I/O area is updated in parts with more than 64KB data space or more than 64KB Program memory, and the increment/ decrement is added to the entire 24-bit address on such devices.

Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

The result of these combinations is undefined:

ST X+, r26

ST X+, r27

ST -X, r26

ST -X, r27

Using the X-pointer:

	Operation:		Comment:
(i)	(X) ← Rr		X: Unchanged
(ii)	(X) ← Rr, X ← X+1		X: Post incremented
(iii)	X ← X - 1, (X) ← Rr		X: Pre decremented
	Syntax:	Operands:	Program Counter:
(i)	ST X, Rr	0 ≤ r ≤ 31	PC ← PC + 1
(ii)	ST X+, Rr	0 ≤ r ≤ 31	PC ← PC + 1
(iii)	ST -X, Rr	0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

(i)	1001	001r	rrrr	1100
(ii)	1001	001r	rrrr	1101
(iii)	1001	001r	rrrr	1110

118.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr r27 ; Clear X high byte
ldi r26,$60 ; Set X low byte to $60
st X+,r0 ; Store r0 in data space loc. $60(X post inc)
st X,r1 ; Store r1 in data space loc. $61
ldi r26,$63 ; Set X low byte to $63
st X,r2 ; Store r2 in data space loc. $63
st -X,r3 ; Store r3 in data space loc. $62(X pre dec)
```

Words	1 (2 bytes)
Cycles	2
Cycles XMEGA	(i) 1
	(ii) 1
	(iii) 2
Cycles Reduced Core tinyAVR	(i) 1
	(ii) 1
	(iii) 2

119. ST (STD) – Store Indirect From Register to Data Space using Index Y

119.1. Description

Stores one byte indirect with or without displacement from a register to data space. For parts with SRAM, the data space consists of the Register File, I/O memory, and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. The EEPROM has a separate address space.

The data location is pointed to by the Y (16 bits) Pointer Register in the Register File. Memory access is limited to the current data segment of 64KB. To access another data segment in devices with more than 64KB data space, the RAMPY in register in the I/O area has to be changed.

The Y-pointer Register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for accessing arrays, tables, and Stack Pointer usage of the Y-pointer Register. Note that only the low byte of the Y-pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPY Register in the I/O area is updated in parts with more than 64KB data space or more than 64KB Program memory, and the increment/ decrement/ displacement is added to the entire 24-bit address on such devices.

Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

The result of these combinations is undefined:

ST Y+, r28

ST Y+, r29

ST -Y, r28

ST -Y, r29

Using the Y-pointer:

	Operation:	Comment:
(i)	(Y) ← Rr	Y: Unchanged
(ii)	(Y) ← Rr, Y ← Y+1	Y: Post incremented
(iii)	Y ← Y - 1, (Y) ← Rr	Y: Pre decremented
(iv)	(Y+q) ← Rr	Y: Unchanged, q: Displacement

	Syntax:	Operands:	Program Counter:
(i)	ST Y, Rr	0 ≤ r ≤ 31	PC ← PC + 1
(ii)	ST Y+, Rr	0 ≤ r ≤ 31	PC ← PC + 1
(iii)	ST -Y, Rr	0 ≤ r ≤ 31	PC ← PC + 1
(iv)	STD Y+q, Rr	0 ≤ r ≤ 31, 0 ≤ q ≤ 63	PC ← PC + 1

16-bit Opcode:

(i)	1000	001r	rrrr	1000
(ii)	1001	001r	rrrr	1001
(iii)	1001	001r	rrrr	1010
(iv)	10q0	qq1r	rrrr	1qqq

119.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr r29 ; Clear Y high byte
ldi r28,$60 ; Set Y low byte to $60
st Y+,r0 ; Store r0 in data space loc. $60(Y post inc)
st Y,r1 ; Store r1 in data space loc. $61
ldi r28,$63 ; Set Y low byte to $63
st Y,r2 ; Store r2 in data space loc. $63
st -Y,r3 ; Store r3 in data space loc. $62(Y pre dec)
std Y+2,r4 ; Store r4 in data space loc. $64
```

Words	1 (2 bytes)
Cycles	2
Cycles XMEGA	(i) 1
	(ii) 1
	(iii) 2
	(iv) 2
Cycles Reduced Core tinyAVR	(i) 1
	(ii) 1
	(iii) 2

120. ST (STD) – Store Indirect From Register to Data Space using Index Z

120.1. Description

Stores one byte indirect with or without displacement from a register to data space. For parts with SRAM, the data space consists of the Register File, I/O memory, and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. The EEPROM has a separate address space.

The data location is pointed to by the Z (16 bits) Pointer Register in the Register File. Memory access is limited to the current data segment of 64KB. To access another data segment in devices with more than 64KB data space, the RAMPZ in register in the I/O area has to be changed.

The Z-pointer Register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for Stack Pointer usage of the Z-pointer Register, however because the Z-pointer Register can be used for indirect subroutine calls, indirect jumps and table look-up, it is often more convenient to use the X- or Y-pointer as a dedicated Stack Pointer. Note that only the low byte of the Z-pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPZ Register in the I/O area is updated in parts with more than 64KB data space or more than 64KB Program memory, and the increment/decrement/displacement is added to the entire 24-bit address on such devices.

Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

The result of these combinations is undefined:

ST Z+, r30

ST Z+, r31

ST -Z, r30

ST -Z, r31

Using the Z-pointer:

	Operation:	Comment:
(i)	(Z) ←Rr	Z: Unchanged
(ii)	(Z) ← Rr, Z ← Z+1	Z: Post incremented
(iii)	Z ← Z - 1, (Z) ← Rr	Z: Pre decremented
(iv)	(Z+q) ← Rr	Z: Unchanged, q: Displacement

	Syntax:	Operands:	Program Counter:
(i)	ST Z, Rr	0 ≤ r ≤ 31	PC ← PC + 1
(ii)	ST Z+, Rr	0 ≤ r ≤ 31	PC ← PC + 1

(iii)	ST -Z, Rr	0 ≤ r ≤ 31	PC ← PC + 1
(iv)	STD Z+q, Rr	0 ≤ r ≤ 31, 0 ≤ q ≤ 63	PC ← PC + 1

16-bit Opcode :

(i)	1000	001r	rrrr	0000
(ii)	1001	001r	rrrr	0001
(iii)	1001	001r	rrrr	0010
(iv)	10q0	qq1r	rrrr	0qqq

120.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr r31 ; Clear Z high byte
ldi r30,$60 ; Set Z low byte to $60
st Z+,r0 ; Store r0 in data space loc. $60(Z post inc)
st Z,r1 ; Store r1 in data space loc. $61
ldi r30,$63 ; Set Z low byte to $63
st Z,r2 ; Store r2 in data space loc. $63
st -Z,r3 ; Store r3 in data space loc. $62(Z pre dec)
std Z+2,r4 ; Store r4 in data space loc. $64
```

Words	1 (2 bytes)
Cycles	2
Cycles XMEGA	(i) 1
	(ii) 1
	(iii) 2
	(iv) 2
Cycles Reduced Core tinyAVR	(i) 1
	(ii) 1
	(iii) 2

121. STS – Store Direct to Data Space

121.1. Description

Stores one byte from a Register to the data space. For parts with SRAM, the data space consists of the Register File, I/O memory, and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. The EEPROM has a separate address space.

A 16-bit address must be supplied. Memory access is limited to the current data segment of 64KB. The STS instruction uses the RAMPD Register to access memory above 64KB. To access another data segment in devices with more than 64KB data space, the RAMPD in register in the I/O area has to be changed.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) (k) ← Rr

	Syntax:	Operands:	Program Counter:
(i)	STS k,Rr	0 ≤ r ≤ 31, 0 ≤ k ≤ 65535	PC ← PC + 2

32-bit Opcode:

1001	001d	dddd	0000
kkkk	kkkk	kkkk	kkkk

121.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
lds r2,$FF00 ; Load r2 with the contents of data space location $FF00
add r2,r1 ; add r1 to r2
sts $FF00,r2 ; Write back
```

Words	2 (4 bytes)
Cycles	2

122. STS (16-bit) – Store Direct to Data Space

122.1. Description

Stores one byte from a Register to the data space. For parts with SRAM, the data space consists of the Register File, I/O memory, and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. In some parts the Flash memory has been mapped to the data space and can be written using this command. The EEPROM has a separate address space.

A 7-bit address must be supplied. The address given in the instruction is coded to a data space address as follows:

ADDR[7:0] = ($\overline{\text{INST[8]}}$, INST[8], INST[10], INST[9], INST[3], INST[2], INST[1], INST[0])

Memory access is limited to the address range 0x40...0xbf of the data segment.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) (k) ← Rr

	Syntax:	Operands:	Program Counter:
(i)	STS k,Rr	16 ≤ r ≤ 31, 0 ≤ k ≤ 127	PC ← PC + 1

16-bit Opcode:

1010	1kkk	dddd	kkkk

122.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
lds r16,$00 ; Load r16 with the contents of data space location $00
add r16,r17 ; add r17 to r16
sts $00,r16 ; Write result to the same address it was fetched from
```

Words	1 (2 bytes)
Cycles	1

Note: Registers r0...r15 are remapped to r16...r31

123. SUB – Subtract Without Carry

123.1. Description

Subtracts two registers and places the result in the destination register Rd.

	Operation:		
(i)	Rd ← Rd - Rr		
	Syntax:	Operands:	Program Counter:
(i)	SUB Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0001	10rd	dddd	rrrr

123.2. Status Register and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$

Set if there was a borrow from bit 3; cleared otherwise.

S N ⊕ V, for signed tests.

V $Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$

Set if two's complement overflow resulted from the operation; cleared otherwise.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

C $\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$

Set if the absolute value of the contents of Rr is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
  sub r13,r12 ; Subtract r12 from r13
  brne noteq ; Branch if r12<>r13
  ...
noteq: nop ; Branch destination (do nothing)
```


Words	1 (2 bytes)
Cycles	1

124. SUBI – Subtract Immediate

124.1. Description

Subtracts a register and a constant, and places the result in the destination register Rd. This instruction is working on Register R16 to R31 and is very well suited for operations on the X, Y, and Z-pointers.

Operation:

(i) Rd ← Rd - K

	Syntax:	Operands:	Program Counter:
(i)	SUBI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

0101	KKKK	dddd	KKKK

124.2. Status Register and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H $\overline{Rd3} \bullet K3 + K3 \bullet R3 + R3 \bullet \overline{Rd3}$

Set if there was a borrow from bit 3; cleared otherwise.

S N ⊕ V, for signed tests.

V $Rd7 \bullet \overline{K7} \bullet \overline{R7} + \overline{Rd7} \bullet K7 \bullet R7$

Set if two's complement overflow resulted from the operation; cleared otherwise.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

C $\overline{Rd7} \bullet K7 + K7 \bullet R7 + R7 \bullet \overline{Rd7}$

Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
  subi r22,$11 ; Subtract $11 from r22
  brne noteq ; Branch if r22<>$11
  ...
noteq: nop ; Branch destination (do nothing)
```


Words	1 (2 bytes)
Cycles	1

125. SWAP – Swap Nibbles

125.1. Description

Swaps high and low nibbles in a register.

Operation:

(i) R(7:4) ← Rd(3:0), R(3:0) ← Rd(7:4)

	Syntax:	Operands:	Program Counter:
(i)	SWAP Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0010

125.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

R (Result) equals Rd after the operation.

Example:

```
inc r1 ; Increment r1
swap r1 ; Swap high and low nibble of r1
inc r1 ; Increment high nibble of r1
swap r1 ; Swap back
```

Words	1 (2 bytes)
Cycles	1

126. TST – Test for Zero or Minus

126.1. Description

Tests if a register is zero or negative. Performs a logical AND between a register and itself. The register will remain unchanged.

Operation:

(i) Rd ← Rd • Rd

	Syntax:	Operands:	Program Counter:
(i)	TST Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode: (see AND Rd, Rd)

0010	00dd	dddd	dddd

126.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S N ⊕ V, for signed tests.

V 0

Cleared.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

R (Result) equals Rd.

Example:

```
 tst r0 ; Test r0
 breq zero ; Branch if r0=0
 ...
zero: nop ; Branch destination (do nothing)
```

Words	1 (2 bytes)
Cycles	1

127. WDR – Watchdog Reset

127.1. Description

This instruction resets the Watchdog Timer. This instruction must be executed within a limited time given by the WD prescaler. See the Watchdog Timer hardware specification.

Operation:

(i) WD timer restart.

	Syntax:	Operands:	Program Counter:
(i)	WDR	None	PC ← PC + 1

16-bit Opcode:

1001	0101	1010	1000

127.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
wdr ; Reset watchdog timer
```

Words	1 (2 bytes)
Cycles	1

128. XCH – Exchange

128.1. Description

Exchanges one byte indirect between register and data space.

The data location is pointed to by the Z (16 bits) Pointer Register in the Register File. Memory access is limited to the current data segment of 64KB. To access another data segment in devices with more than 64KB data space, the RAMPZ in register in the I/O area has to be changed.

The Z-pointer Register is left unchanged by the operation. This instruction is especially suited for writing/reading status bits stored in SRAM.

Operation:

(i) (Z) ← Rd, Rd ← (Z)

	Syntax:	Operands:	Program Counter:
(i)	XCH Z,Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	001r	rrrr	0100

128.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	1 (2 bytes)
Cycles	2

The Assembler is not case sensitive.

The operands have the following forms:

Rd:	R0-R31 or R16-R31 (depending on instruction)
Rr:	R0-R31
b:	Constant (0-7), can be a constant expression
s:	Constant (0-7), can be a constant expression
P:	Constant (0-31/63), can be a constant expression
K:	Constant (0-255), can be a constant expression
k:	Constant, value range depending on instruction. Can be a constant expression.
q:	Constant (0-63), can be a constant expression

4.5 Assembler directives

The Assembler supports a number of directives. The directives are not translated directly into opcodes. Instead, they are used to adjust the location of the program in memory, define macros, initialize memory and so on. An overview of the directives is given in the following table.

Summary of directives:

Directive	Description
BYTE	Reserve byte to a variable
CSEG	Code Segment
DB	Define constant byte(s)
DEF	Define a symbolic name on a register
DEVICE	Define which device to assemble for
DSEG	Data Segment
DW	Define constant word(s)
ENDMACRO	End macro
EQU	Set a symbol equal to an expression
ESEG	EEPROM Segment
EXIT	Exit from file
INCLUDE	Read source from another file
LIST	Turn listfile generation on
LISTMAC	Turn macro expansion on
MACRO	Begin macro
NOLIST	Turn listfile generation off
ORG	Set program origin
SET	Set a symbol to an expression

Note: *All directives must be preceded by a period.*

4.5.1 BYTE - Reserve bytes to a variable

The BYTE directive reserves memory resources in the SRAM. In order to be able to refer to the reserved location, the BYTE directive should be preceded by a label. The directive takes one parameter, which is the number of bytes to reserve. The directive can only be used within a Data Segment (see directives CSEG, DSEG and ESEG). Note that a parameter must be given. The allocated bytes are not initialized.

Syntax:

```
LABEL:  .BYTE expression
```

Example:

```
.DSEG
        var1:   .BYTE 1         ; reserve 1 byte to var1
        table:  .BYTE tab_size  ; reserve tab_size bytes
.CSEG
        ldi     r30,low(var1)   ; Load Z register low
        ldi     r31,high(var1)  ; Load Z register high
        ld      r1,Z            ; Load VAR1 into register 1
```

4.5.2 CSEG - Code Segment

The CSEG directive defines the start of a Code Segment. An Assembler file can consist of several Code Segments, which are concatenated into one Code Segment when assembled. The BYTE directive can not be used within a Code Segment. The default segment type is Code. The Code Segments have their own location counter which is a word counter. The ORG directive (see description later in this document) can be used to place code and constants at specific locations in the Program memory. The directive does not take any parameters.

Syntax:

```
.CSEG
```

Example:

```
.DSEG                           ; Start data segment
        vartab: .BYTE 4         ; Reserve 4 bytes in SRAM
.CSEG                           ; Start code segment
        const:  .DW 2           ; Write 0x0002 in prog.mem.
        mov r1,r0               ; Do something
```

4.5.3 DB-Define constant byte(s) in program memory or E^{2}PROM memory

The DB directive reserves memory resources in the program memory or the EEPROM memory. In order to be able to refer to the reserved locations, the DB directive should be preceded by a label.

The DB directive takes a list of expressions, and must contain at least one expression. The DB directive must be placed in a Code Segment or an EEPROM Segment.

The expression list is a sequence of expressions, delimited by commas. Each expression must evaluate to a number between -128 and 255. If the expression evaluates to a negative number, the 8 bits two's complement of the number will be placed in the program memory or EEPROM memory location.

If the DB directive is used in a Code Segment and the expressionlist contains more than one expression, the expressions are packed so that two bytes are placed in each program memory word. If the expressionlist contains an odd number of expressions, the last expression will be placed in a program memory word of its own, even if the next line in the assembly code contains a DB directive.

Syntax:

```
LABEL: .DB expressionlist
```

Example:

```
.CSEG
        consts: .DB 0, 255, 0b01010101, -128, 0xaa
.ESEG
        eeconst:.DB 0xff
```

4.5.4 DEF - Set a symbolic name on a register

The DEF directive allows the registers to be referred to through symbols. A defined symbol can be used in the rest of the program to refer to the register it is assigned to. A register can have several symbolic names attached to it. A symbol can be redefined later in the program.

Syntax:

```
.DEF Symbol=Register
```

Example:

```
.DEF temp=R16
.DEF ior=R0
.CSEG
        ldi     temp,0xf0     ; Load 0xf0 into temp register
        in      ior,0x3f      ; Read SREG into ior register
        eor     temp,ior      ; Exclusive or temp and ior
```


4.5.5 DEVICE - Define which device to assemble for

The DEVICE directive allows the user to tell the Assembler which device the code is to be executed on. If this directive is used, a warning is issued if an instruction not supported by the specified device occurs in the code. If the size of the Code Segment or EEPROM Segment is larger than supported by the specified device, a warning is issued. If the DEVICE directive is not used, it is assumed that all instructions are supported and that there are no restrictions on memory sizes.

Syntax:

```
.DEVICE AT90S1200 | AT90S2313 | AT90S4414 | AT90S8515
```

Example:

```
.DEVICE AT90S1200            ; Use the AT90S1200
.CSEG
        push    r30          ; This statement will generate
                             ; a warning since the
                             ; specified device does not
                             ; have this instruction
```

4.5.6 DSEG - Data Segment

The DSEG directive defines the start of a Data Segment. An Assembler file can consist of several Data Segments, which are concatenated into one Data Segment when assembled. A Data Segment will normally only consist of BYTE directives (and labels). The Data Segments have their own location counter which is a byte counter. The ORG directive (see description later in this document) can be used to place the variables at specific locations in the SRAM. The directive does not take any parameters.

Syntax:

```
.DSEG
```

Example:

```
.DSEG                           ; Start data segment
        var1:.BYTE 1            ; reserve 1 byte to var1
        table:.BYTE tab_size    ; reserve tab_size bytes.
.CSEG
        ldi     r30,low(var1)   ; Load Z register low
        ldi     r31,high(var1)  ; Load Z register high
        ld      r1,Z            ; Load var1 into register 1
```

4.5.7 DW-Define constant word(s) in program memory or E²PROM memory

The DW directive reserves memory resources in the program memory or EEPROM memory. In order to be able to refer to the reserved locations, the DW directive should be preceded by a label.

The DW directive takes a list of expressions, and must contain at least one expression.

The DB directive must be placed in a Code Segment or an EEPROM Segment.

The expression list is a sequence of expressions, delimited by commas. Each expression must evaluate to a number between -32768 and 65535. If the expression evaluates to a negative number, the 16 bits two's complement of the number will be placed in the program memory location.

Syntax:

```
LABEL: .DW expressionlist
```

Example:

```
.CSEG
        varlist:.DW 0,0xffff,0b1001110001010101,-32768,65535
.ESEG
        eevar:  .DW 0xffff
```

4.5.8 ENDMACRO - End macro

The ENDMACRO directive defines the end of a Macro definition. The directive does not take any parameters. See the MACRO directive for more information on defining Macros.

Syntax:

```
.ENDMACRO
```

Example:

```
.MACRO   SUBI16                 ; Start macro definition
         subi    r16,low(@0)    ; Subtract low byte
         sbci    r17,high(@0)   ; Subtract high byte
.ENDMACRO                       ; End macro definition
```

4.5.9 EQU - Set a symbol equal to an expression

The EQU directive assigns a value to a label. This label can then be used in later expressions. A label assigned to a value by the EQU directive is a constant and can not be changed or redefined.

Syntax:

```
.EQU label = expression
```

Example:

```
.EQU io_offset = 0x23
.EQU porta = io_offset + 2
.CSEG                           ; Start code segment
         clr     r2             ; Clear register 2
         out     porta,r2       ; Write to Port A
```


4.5.10 ESEG - EEPROM Segment

The ESEG directive defines the start of an EEPROM Segment. An Assembler file can consist of several EEPROM Segments, which are concatenated into one EEPROM Segment when assembled. The BYTE directive can not be used within an EEPROM Segment. The EEPROM Segments have their own location counter which is a byte counter. The ORG directive (see description later in this document) can be used to place constants at specific locations in the EEPROM memory. The directive does not take any parameters.

Syntax:

```
.ESEG
```

Example:

```
.DSEG                           ; Start data segment
        vartab:  .BYTE 4        ; Reserve 4 bytes in SRAM
.ESEG
        eevar:   .DW 0xff0f     ; Initialize one word in
                                ; EEPROM
.CSEG                           ; Start code segment
        const:   .DW 2          ; Write 0x0002 in prog.mem.
        mov r1,r0               ; Do something
```

4.5.11 EXIT - Exit this file

The EXIT directive tells the Assembler to stop assembling the file. Normally, the Assembler runs until end of file (EOF). If an EXIT directive appears in an included file, the Assembler continues from the line following the INCLUDE directive in the file containing the INCLUDE directive.

Syntax:

```
.EXIT
```

Example:

```
.EXIT                           ; Exit this file
```

4.5.12 INCLUDE - Include another file

The INCLUDE directive tells the Assembler to start reading from a specified file. The Assembler then assembles the specified file until end of file (EOF) or an EXIT directive is encountered. An included file may itself contain INCLUDE directives.

Syntax:

```
.INCLUDE “filename”
```

Example:

```
                                ; iodefs.asm:
.EQU    sreg=0x3f               ; Status register
.EQU    sphigh=0x3e             ; Stack pointer high
.EQU    splow=0x3d              ; Stack pointer low
                                ; incdemo.asm
.INCLUDE “iodefs.asm”           ; Include I/O definitions
        in    r0,sreg           ; Read status register
```

4.5.13 LIST - Turn the listfile generation on

The LIST directive tells the Assembler to turn listfile generation on. The Assembler generates a listfile which is a combination of assembly source code, addresses and opcodes. Listfile generation is turned on by default. The directive can also be used together with the NOLIST directive in order to only generate listfile of selected parts of an assembly source file.

Syntax:

```
.LIST
```

Example:

```
.NOLIST                     ; Disable listfile generation
.INCLUDE “macro.inc”        ; The included files will not
.INCLUDE “const.def”        ; be shown in the listfile
.LIST                       ; Reenable listfile generation
```

4.5.14 LISTMAC - Turn macro expansion on

The LISTMAC directive tells the Assembler that when a macro is called, the expansion of the macro is to be shown on the listfile generated by the Assembler. The default is that only the macro-call with parameters is shown in the listfile.

Syntax:

```
.LISTMAC
```

Example:

```
.MACRO   MACX                ; Define an example macro
         add     r0,@0       ; Do something
         eor     r1,@1       ; Do something
.ENDMACRO                    ; End macro definition
.LISTMAC                     ; Enable macro expansion
         MACX    r2,r1       ; Call macro, show expansion
```

4.5.15 MACRO - Begin macro

The MACRO directive tells the Assembler that this is the start of a Macro. The MACRO directive takes the Macro name as parameter. When the name of the Macro is written later in the program, the Macro definition is expanded at the place it was used. A Macro can take up to 10 parameters. These parameters are referred to as @0-@9 within the Macro definition. When issuing a Macro call, the parameters are given as a comma separated list. The Macro definition is terminated by an ENDMACRO directive.

By default, only the call to the Macro is shown on the listfile generated by the Assembler. In order to include the macro expansion in the listfile, a LISTMAC directive must be used. A macro is marked with a + in the opcode field of the listfile.

Syntax:

```
.MACRO macroname
```

Example:

```
.MACRO     SUBI16              ; Start macro definition
           subi    @1,low(@0)  ; Subtract low byte
           sbci    @2,high(@0) ; Subtract high byte
.ENDMACRO                      ; End macro definition
.CSEG                          ; Start code segment
           SUBI16  0x1234,r16,r17; Sub.0x1234 from r17:r16
```


4.5.16 NOLIST - Turn listfile generation off

The NOLIST directive tells the Assembler to turn listfile generation off. The Assembler normally generates a listfile which is a combination of assembly source code, addresses and opcodes. Listfile generation is turned on by default, but can be disabled by using this directive. The directive can also be used together with the LIST directive in order to only generate listfile of selected parts of an assembly source file.

Syntax:

```
.NOLIST                     ; Enable listfile generation
```

Example:

```
.NOLIST                     ; Disable listfile generation
.INCLUDE “macro.inc”        ; The included files will not
.INCLUDE “const.def”        ; be shown in the listfile
.LIST                       ; Reenable listfile generation
```

4.5.17 ORG - Set program origin

The ORG directive sets the location counter to an absolute value. The value to set is given as a parameter. If an ORG directive is given within a Data Segment, then it is the SRAM location counter which is set, if the directive is given within a Code Segment, then it is the Program memory counter which is set and if the directive is given within an EEPROM Segment, then it is the EEPROM location counter which is set. If the directive is preceded by a label (on the same source code line), the label will be given the value of the parameter. The default values of the Code and EEPROM location counters are zero, whereas the default value of the SRAM location counter is 32 (due to the registers occupying addresses 0-31) when the assembling is started. Note that the EEPROM and SRAM location counters count bytes whereas the Program memory location counter counts words.

Syntax:

```
.ORG expression
```

Example:

```
.DSEG                          ; Start data segment
.ORG 0x67                      ; Set SRAM address to hex 67
       variable:.BYTE 1        ; Reserve a byte at SRAM
                               ; adr.67H
.ESEG                          ; Start EEPROM Segment
.ORG 0x20                      ; Set EEPROM location
                               ; counter
       eevar:   .DW 0xfeff     ; Initialize one word
.CSEG
.ORG 0x10                      ; Set Program Counter to hex
                               ; 10
       mov      r0,r1          ; Do something
```

4.5.18 SET - Set a symbol equal to an expression

The SET directive assigns a value to a label. This label can then be used in later expressions. A label assigned to a value by the SET directive can be changed later in the program.

Syntax:

```
.SET label = expression
```

Example:

```
.SET io_offset = 0x23
.SET porta = io_offset + 2
.CSEG                          ; Start code segment
       clr     r2              ; Clear register 2
       out     porta,r2        ; Write to Port A
```

4.6 Expressions

The Assembler incorporates expressions. Expressions can consist of operands, operators and functions. All expressions are internally 32 bits.

4.6.1 Operands

The following operands can be used:

- User defined labels which are given the value of the location counter at the place they appear.
- User defined variables defined by the SET directive
- User defined constants defined by the EQU directive
- Integer constants: constants can be given in several formats, including
  a) Decimal (default): 10, 255
  b) Hexadecimal (two notations): 0x0a, $0a, 0xff, $ff
  c) Binary: 0b00001010, 0b11111111
- PC - the current value of the Program memory location counter

4.6.2 Functions

The following functions are defined:

- LOW(expression) returns the low byte of an expression
- HIGH(expression) returns the second byte of an expression
- BYTE2(expression) is the same function as HIGH
- BYTE3(expression) returns the third byte of an expression
- BYTE4(expression) returns the fourth byte of an expression
- LWRD(expression) returns bits 0-15 of an expression
- HWRD(expression) returns bits 16-31 of an expression
- PAGE(expression) returns bits 16-21 of an expression
- EXP2(expression) returns 2^expression
- LOG2(expression) returns the integer part of log2(expression)

4.6.3 Operators

The Assembler supports a number of operators which are described here. The higher the precedence, the higher the priority. Expressions may be enclosed in parentheses, and such expressions are always evaluated before combined with anything outside the parentheses.

4.6.3.1 Logical Not

Symbol: !

Description: Unary operator which returns 1 if the expression was zero, and returns 0 if the expression was nonzero

Precedence: 14

Example:
```
ldi   r16,!0xf0      ; Load r16 with 0x00
```

4.6.3.2 Bitwise Not

Symbol: ~

Description: Unary operator which returns the input expression with all bits inverted

Precedence: 14

Example:
```
ldi   r16,~0xf0      ; Load r16 with 0x0f
```

4.6.3.3 Unary Minus

Symbol: -

Description: Unary operator which returns the arithmetic negation of an expression

Precedence: 14

Example:
```
ldi   r16,-2         ; Load -2(0xfe) in r16
```

4.6.3.4 Multiplication

Symbol: *

Description: Binary operator which returns the product of two expressions

Precedence: 13

Example:
```
ldi   r30,label*2    ; Load r30 with label*2
```

4.6.3.5 Division

Symbol: /

Description: Binary operator which returns the integer quotient of the left expression divided by the right expression

Precedence: 13

Example:
```
ldi   r30,label/2    ; Load r30 with label/2
```

4.6.3.6 Addition

Symbol: +

Description: Binary operator which returns the sum of two expressions

Precedence: 12

Example:
```
ldi   r30,c1+c2      ; Load r30 with c1+c2
```

4.6.3.7 Subtraction

Symbol: -

Description: Binary operator which returns the left expression minus the right expression

Precedence: 12

Example:
```
ldi   r17,c1-c2      ;Load r17 with c1-c2
```

4.6.3.8 Shift left

Symbol: <<

Description: Binary operator which returns the left expression shifted left a number of times given by the right expression

Precedence: 11

Example:
```
ldi   r17,1<<bitmask ;Load r17 with 1 shifted
                     ;left bitmask times
```

4.6.3.9 Shift right

Symbol: >>

Description: Binary operator which returns the left expression shifted right a number of times given by the right expression.

Precedence: 11

Example:
```
ldi   r17,c1>>c2     ;Load r17 with c1 shifted
                     ;right c2 times
```

4.6.3.10 Less than

Symbol: <

Description: Binary operator which returns 1 if the signed expression to the left is Less than the signed expression to the right, 0 otherwise

Precedence: 10

Example:
```
ori   r18,bitmask*(c1<c2)+1      ;Or r18 with
                                 ;an expression
```

4.6.3.11 Less or Equal

Symbol: <=

Description: Binary operator which returns 1 if the signed expression to the left is Less than or Equal to the signed expression to the right, 0 otherwise

Precedence: 10

Example:
```
ori   r18,bitmask*(c1<=c2)+1     ;Or r18 with
                                 ;an expression
```

4.6.3.12 Greater than

Symbol: >

Description: Binary operator which returns 1 if the signed expression to the left is Greater than the signed expression to the right, 0 otherwise

Precedence: 10

Example:
```
ori   r18,bitmask*(c1>c2)+1      ;Or r18 with
                                 ;an expression
```

4.6.3.13 Greater or Equal

Symbol: >=

Description: Binary operator which returns 1 if the signed expression to the left is Greater than or Equal to the signed expression to the right, 0 otherwise

Precedence: 10

Example:
```
ori   r18,bitmask*(c1>=c2)+1     ;Or r18 with
                                 ;an expression
```

4.6.3.14 Equal

Symbol: ==

Description: Binary operator which returns 1 if the signed expression to the left is Equal to the signed expression to the right, 0 otherwise

Precedence: 9

Example:
```
andi  r19,bitmask*(c1==c2)+1     ;And r19 with
                                 ;an expression
```


4.6.3.15 Not Equal

Symbol: !=

Description: Binary operator which returns 1 if the signed expression to the left is Not Equal to the signed expression to the right, 0 otherwise

Precedence: 9

Example:
```
.SET flag=(c1!=c2)       ;Set flag to 1 or 0
```

4.6.3.16 Bitwise And

Symbol: &

Description: Binary operator which returns the bitwise And between two expressions

Precedence: 8

Example:
```
ldi   r18,High(c1&c2)   ;Load r18 with an expression
```

4.6.3.17 Bitwise Xor

Symbol: ^

Description: Binary operator which returns the bitwise Exclusive Or between two expressions

Precedence: 7

Example:
```
ldi   r18,Low(c1^c2)    ;Load r18 with an expression
```

4.6.3.18 Bitwise Or

Symbol: |

Description: Binary operator which returns the bitwise Or between two expressions

Precedence: 6

Example:
```
ldi   r18,Low(c1|c2)    ;Load r18 with an expression
```

4.6.3.19 Logical And

Symbol: &&

Description: Binary operator which returns 1 if the expressions are both nonzero, 0 otherwise

Precedence: 5

Example:
```
ldi   r18,Low(c1&&c2)   ;Load r18 with an expression
```

4.6.3.20 Logical Or

Symbol: ||

Description: Binary operator which returns 1 if one or both of the expressions are nonzero, 0 otherwise

Precedence: 4

Example:
```
ldi   r18,Low(c1||c2)   ;Load r18 with an expression
```

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
Analog Comparator A	**Registers**			
	ACA_AC0CTRL	896	0x380	Analog Comparator 0 Control
	ACA_AC1CTRL	897	0x381	Analog Comparator 1 Control
	ACA_AC0MUXCTRL	898	0x382	Analog Comparator 0 MUX Control
	ACA_AC1MUXCTRL	899	0x383	Analog Comparator 1 MUX Control
	ACA_CTRLA	900	0x384	Control Register A
	ACA_CTRLB	901	0x385	Control Register B
	ACA_WINCTRL	902	0x386	Window Mode Control
	ACA_STATUS	903	0x387	Status
	Interrupt Vectors			
	ACA_AC0_vect	136	0x88	AC0 Interrupt
	ACA_AC1_vect	138	0x8A	AC1 Interrupt
	ACA_ACW_vect	140	0x8C	ACW Window Mode Interrupt
Analog Comparator B	**Registers**			
	ACB_AC0CTRL	912	0x390	Analog Comparator 0 Control
	ACB_AC1CTRL	913	0x391	Analog Comparator 1 Control
	ACB_AC0MUXCTRL	914	0x392	Analog Comparator 0 MUX Control
	ACB_AC1MUXCTRL	915	0x393	Analog Comparator 1 MUX Control
	ACB_CTRLA	916	0x394	Control Register A
	ACB_CTRLB	917	0x395	Control Register B
	ACB_WINCTRL	918	0x396	Window Mode Control
	ACB_STATUS	919	0x397	Status
	Interrupt Vectors			
	ACB_AC0_vect	72	0x48	AC0 Interrupt
	ACB_AC1_vect	74	0x4A	AC1 Interrupt
	ACB_ACW_vect	76	0x4C	ACW Window Mode Interrupt
	Registers			
	ADCA_CTRLA	512	0x200	Control Register A
	ADCA_CTRLB	513	0x201	Control Register B
	ADCA_REFCTRL	514	0x202	Reference Control
	ADCA_EVCTRL	515	0x203	Event Control
	ADCA_PRESCALER	516	0x204	Clock Prescaler
	ADCA_INTFLAGS	518	0x206	Interrupt Flags
	ADCA_TEMP	519	0x207	Temporary Register
	ADCA_CAL	524	0x20C	Calibration Value
	ADCA_CH0RES	528	0x210	Channel 0 Result
	ADCA_CH1RES	530	0x212	Channel 1 Result
	ADCA_CH2RES	532	0x214	Channel 2 Result
	ADCA_CH3RES	534	0x216	Channel 3 Result
	ADCA_CMP	536	0x218	Compare Value
	ADCA_CH0_CTRL	544	0x220	Control Register
	ADCA_CH0_MUXCTRL	545	0x221	MUX Control
	ADCA_CH0_INTCTRL	546	0x222	Channel Interrupt Control Register

Thanks to Mason Turner!

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
Analog to Digital Converter A	ADCA_CH0_INTFLAGS	547	0x223	Interrupt Flags
	ADCA_CH0_RES	548	0x224	Channel Result
	ADCA_CH0_SCAN	550	0x226	Input Channel Scan
	ADCA_CH1_CTRL	552	0x228	Control Register
	ADCA_CH1_MUXCTRL	553	0x229	MUX Control
	ADCA_CH1_INTCTRL	554	0x22A	Channel Interrupt Control Register
	ADCA_CH1_INTFLAGS	555	0x22B	Interrupt Flags
	ADCA_CH1_RES	556	0x22C	Channel Result
	ADCA_CH1_SCAN	558	0x22E	Input Channel Scan
	ADCA_CH2_CTRL	560	0x230	Control Register
	ADCA_CH2_MUXCTRL	561	0x231	MUX Control
	ADCA_CH2_INTCTRL	562	0x232	Channel Interrupt Control Register
	ADCA_CH2_INTFLAGS	563	0x233	Interrupt Flags
	ADCA_CH2_RES	564	0x234	Channel Result
	ADCA_CH2_SCAN	566	0x236	Input Channel Scan
	ADCA_CH3_CTRL	568	0x238	Control Register
	ADCA_CH3_MUXCTRL	569	0x239	MUX Control
	ADCA_CH3_INTCTRL	570	0x23A	Channel Interrupt Control Register
	ADCA_CH3_INTFLAGS	571	0x23B	Interrupt Flags
	ADCA_CH3_RES	572	0x23C	Channel Result
	ADCA_CH3_SCAN	574	0x23E	Input Channel Scan
	Interrupt Vectors			
	ADCA_CH0_vect	142	0x8E	Interrupt 0
	ADCA_CH1_vect	144	0x90	Interrupt 1
	ADCA_CH2_vect	146	0x92	Interrupt 2
	ADCA_CH3_vect	148	0x94	Interrupt 3
	Registers			
	ADCB_CTRLA	576	0x240	Control Register A
	ADCB_CTRLB	577	0x241	Control Register B
	ADCB_REFCTRL	578	0x242	Reference Control
	ADCB_EVCTRL	579	0x243	Event Control
	ADCB_PRESCALER	580	0x244	Clock Prescaler
	ADCB_INTFLAGS	582	0x246	Interrupt Flags
	ADCB_TEMP	583	0x247	Temporary Register
	ADCB_CAL	588	0x24C	Calibration Value
	ADCB_CH0RES	592	0x250	Channel 0 Result
	ADCB_CH1RES	594	0x252	Channel 1 Result
	ADCB_CH2RES	596	0x254	Channel 2 Result
	ADCB_CH3RES	598	0x256	Channel 3 Result
	ADCB_CMP	600	0x258	Compare Value
	ADCB_CH0_CTRL	608	0x260	Control Register
	ADCB_CH0_MUXCTRL	609	0x261	MUX Control
	ADCB_CH0_INTCTRL	610	0x262	Channel Interrupt Control Register

Thanks to Mason Turner!

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
Analog to Digital Converter B	ADCB_CH0_INTFLAGS	611	0x263	Interrupt Flags
	ADCB_CH0_RES	612	0x264	Channel Result
	ADCB_CH0_SCAN	614	0x266	Input Channel Scan
	ADCB_CH1_CTRL	616	0x268	Control Register
	ADCB_CH1_MUXCTRL	617	0x269	MUX Control
	ADCB_CH1_INTCTRL	618	0x26A	Channel Interrupt Control Register
	ADCB_CH1_INTFLAGS	619	0x26B	Interrupt Flags
	ADCB_CH1_RES	620	0x26C	Channel Result
	ADCB_CH1_SCAN	622	0x26E	Input Channel Scan
	ADCB_CH2_CTRL	624	0x270	Control Register
	ADCB_CH2_MUXCTRL	625	0x271	MUX Control
	ADCB_CH2_INTCTRL	626	0x272	Channel Interrupt Control Register
	ADCB_CH2_INTFLAGS	627	0x273	Interrupt Flags
	ADCB_CH2_RES	628	0x274	Channel Result
	ADCB_CH2_SCAN	630	0x276	Input Channel Scan
	ADCB_CH3_CTRL	632	0x278	Control Register
	ADCB_CH3_MUXCTRL	633	0x279	MUX Control
	ADCB_CH3_INTCTRL	634	0x27A	Channel Interrupt Control Register
	ADCB_CH3_INTFLAGS	635	0x27B	Interrupt Flags
	ADCB_CH3_RES	636	0x27C	Channel Result
	ADCB_CH3_SCAN	638	0x27E	Input Channel Scan
	Interrupt Vectors			
	ADCB_CH0_vect	78	0x4E	Interrupt 0
	ADCB_CH1_vect	80	0x50	Interrupt 1
	ADCB_CH2_vect	82	0x52	Interrupt 2
	ADCB_CH3_vect	84	0x54	Interrupt 3
Advanced Waveform Extension on Port C	**Registers**			
	AWEXC_CTRL	2176	0x880	Control Register
	AWEXC_FDEMASK	2178	0x882	Fault Detection Event Mask
	AWEXC_FDCTRL	2179	0x883	Fault Detection Control Register
	AWEXC_STATUS	2180	0x884	Status Register
	AWEXC_STATUSSET	2181	0x885	Status Set Register
	AWEXC_DTBOTH	2182	0x886	Dead Time Both Sides
	AWEXC_DTBOTHBUF	2183	0x887	Dead Time Both Sides Buffer
	AWEXC_DTLS	2184	0x888	Dead Time Low Side
	AWEXC_DTHS	2185	0x889	Dead Time High Side
	AWEXC_DTLSBUF	2186	0x88A	Dead Time Low Side Buffer
	AWEXC_DTHSBUF	2187	0x88B	Dead Time High Side Buffer
	AWEXC_OUTOVEN	2188	0x88C	Output Override Enable
n on Port E	**Registers**			
	AWEXE_CTRL	2688	0xA80	Control Register
	AWEXE_FDEMASK	2690	0xA82	Fault Detection Event Mask
	AWEXE_FDCTRL	2691	0xA83	Fault Detection Control Register

Thanks to Mason Turner!

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
Advanced Waveform Extensio	AWEXE_STATUS	2692	0xA84	Status Register
	AWEXE_STATUSSET	2693	0xA85	Status Set Register
	AWEXE_DTBOTH	2694	0xA86	Dead Time Both Sides
	AWEXE_DTBOTHBUF	2695	0xA87	Dead Time Both Sides Buffer
	AWEXE_DTLS	2696	0xA88	Dead Time Low Side
	AWEXE_DTHS	2697	0xA89	Dead Time High Side
	AWEXE_DTLSBUF	2698	0xA8A	Dead Time Low Side Buffer
	AWEXE_DTHSBUF	2699	0xA8B	Dead Time High Side Buffer
	AWEXE_OUTOVEN	2700	0xA8C	Output Override Enable
Cyclic Redundancy Check Generator	CRC_CTRL	208	0xD0	Control Register
	CRC_STATUS	209	0xD1	Status Register
	CRC_DATAIN	211	0xD3	Data Input
	CRC_CHECKSUM0	212	0xD4	Checksum byte 0
	CRC_CHECKSUM1	213	0xD5	Checksum byte 1
	CRC_CHECKSUM2	214	0xD6	Checksum byte 2
	CRC_CHECKSUM3	215	0xD7	Checksum byte 3
Clock System	**Registers**			
	CLK_CTRL	64	0x40	Control Register
	CLK_PSCTRL	65	0x41	Prescaler Control Register
	CLK_LOCK	66	0x42	Lock register
	CLK_RTCCTRL	67	0x43	RTC Control Register
	CLK_USBCTRL	68	0x44	USB Control Register
Digital Frequency Locked Loop (2MHz)	**Registers**			
	DFLLRC2M_CTRL	104	0x68	Control Register
	DFLLRC2M_CALA	106	0x6A	Calibration Register A
	DFLLRC2M_CALB	107	0x6B	Calibration Register B
	DFLLRC2M_COMP0	108	0x6C	Oscillator Compare Register 0
	DFLLRC2M_COMP1	109	0x6D	Oscillator Compare Register 1
	DFLLRC2M_COMP2	110	0x6E	Oscillator Compare Register 2
Digital Frequency Locked Loop (32MHz)	**Registers**			
	DFLLRC32M_CTRL	96	0x60	Control Register
	DFLLRC32M_CALA	98	0x62	Calibration Register A
	DFLLRC32M_CALB	99	0x63	Calibration Register B
	DFLLRC32M_COMP0	100	0x64	Oscillator Compare Register 0
	DFLLRC32M_COMP1	101	0x65	Oscillator Compare Register 1
	DFLLRC32M_COMP2	102	0x66	Oscillator Compare Register 2
CPU Registers	**Registers**			
	CPU_CCP	52	0x34	Configuration Change Protection
	CPU_RAMPD	56	0x38	Ramp D
	CPU_RAMPX	57	0x39	Ramp X
	CPU_RAMPY	58	0x3A	Ramp Y
	CPU_RAMPZ	59	0x3B	Ramp Z
	CPU_EIND	60	0x3C	Extended Indirect Jump

Thanks to Mason Turner!

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
	CPU_SPL	61	0x3D	Stack Pointer Low
	CPU_SPH	62	0x3E	Stack Pointer High
	CPU_SREG	63	0x3F	Status Register
Digital to Analog Converter A	**Registers**			
	DACA_CTRLA	768	0x300	Control Register A
	DACA_CTRLB	769	0x301	Control Register B
	DACA_CTRLC	770	0x302	Control Register C
	DACA_EVCTRL	771	0x303	Event Input Control
	DACA_TIMCTRL	772	0x304	Timing Control
	DACA_STATUS	773	0x305	Status
	DACA_CH0GAINCAL	776	0x308	Gain Calibration
	DACA_CH0OFFSETCAL	777	0x309	Offset Calibration
	DACA_CH1GAINCAL	778	0x30A	Gain Calibration
	DACA_CH1OFFSETCAL	779	0x30B	Offset Calibration
	DACA_CH0DATA	792	0x318	Channel 0 Data
	DACA_CH1DATA	794	0x31A	Channel 1 Data
Digital to Analog Converter B	**Registers**			
	DACB_CTRLA	800	0x320	Control Register A
	DACB_CTRLB	801	0x321	Control Register B
	DACB_CTRLC	802	0x322	Control Register C
	DACB_EVCTRL	803	0x323	Event Input Control
	DACB_TIMCTRL	804	0x324	Timing Control
	DACB_STATUS	805	0x325	Status
	DACB_CH0GAINCAL	808	0x328	Gain Calibration
	DACB_CH0OFFSETCAL	809	0x329	Offset Calibration
	DACB_CH1GAINCAL	810	0x32A	Gain Calibration
	DACB_CH1OFFSETCAL	811	0x32B	Offset Calibration
	DACB_CH0DATA	824	0x338	Channel 0 Data
	DACB_CH1DATA	826	0x33A	Channel 1 Data
	Registers			
	DMA_CTRL	256	0x100	Control
	DMA_INTFLAGS	259	0x103	Transfer Interrupt Status
	DMA_STATUS	260	0x104	Status
	DMA_TEMP	262	0x106	Temporary Register For 1624-bit Access
	DMA_CH0_CTRLA	272	0x110	Channel Control
	DMA_CH0_CTRLB	273	0x111	Channel Control
	DMA_CH0_ADDRCTRL	274	0x112	Address Control
	DMA_CH0_TRIGSRC	275	0x113	Channel Trigger Source
	DMA_CH0_TRFCNT	276	0x114	Channel Block Transfer Count
	DMA_CH0_REPCNT	278	0x116	Channel Repeat Count
	DMA_CH0_SRCADDR0	280	0x118	Channel Source Address 0
	DMA_CH0_SRCADDR1	281	0x119	Channel Source Address 1
	DMA_CH0_SRCADDR2	282	0x11A	Channel Source Address 2

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
DMA Controller	DMA_CH0_DESTADDR0	284	0x11C	Channel Destination Address 0
	DMA_CH0_DESTADDR1	285	0x11D	Channel Destination Address 1
	DMA_CH0_DESTADDR2	286	0x11E	Channel Destination Address 2
	DMA_CH1_CTRLA	288	0x120	Channel Control
	DMA_CH1_CTRLB	289	0x121	Channel Control
	DMA_CH1_ADDRCTRL	290	0x122	Address Control
	DMA_CH1_TRIGSRC	291	0x123	Channel Trigger Source
	DMA_CH1_TRFCNT	292	0x124	Channel Block Transfer Count
	DMA_CH1_REPCNT	294	0x126	Channel Repeat Count
	DMA_CH1_SRCADDR0	296	0x128	Channel Source Address 0
	DMA_CH1_SRCADDR1	297	0x129	Channel Source Address 1
	DMA_CH1_SRCADDR2	298	0x12A	Channel Source Address 2
	DMA_CH1_DESTADDR0	300	0x12C	Channel Destination Address 0
	DMA_CH1_DESTADDR1	301	0x12D	Channel Destination Address 1
	DMA_CH1_DESTADDR2	302	0x12E	Channel Destination Address 2
	DMA_CH2_CTRLA	304	0x130	Channel Control
	DMA_CH2_CTRLB	305	0x131	Channel Control
	DMA_CH2_ADDRCTRL	306	0x132	Address Control
	DMA_CH2_TRIGSRC	307	0x133	Channel Trigger Source
	DMA_CH2_TRFCNT	308	0x134	Channel Block Transfer Count
	DMA_CH2_REPCNT	310	0x136	Channel Repeat Count
	DMA_CH2_SRCADDR0	312	0x138	Channel Source Address 0
	DMA_CH2_SRCADDR1	313	0x139	Channel Source Address 1
	DMA_CH2_SRCADDR2	314	0x13A	Channel Source Address 2
	DMA_CH2_DESTADDR0	316	0x13C	Channel Destination Address 0
	DMA_CH2_DESTADDR1	317	0x13D	Channel Destination Address 1
	DMA_CH2_DESTADDR2	318	0x13E	Channel Destination Address 2
	DMA_CH3_CTRLA	320	0x140	Channel Control
	DMA_CH3_CTRLB	321	0x141	Channel Control
	DMA_CH3_ADDRCTRL	322	0x142	Address Control
	DMA_CH3_TRIGSRC	323	0x143	Channel Trigger Source
	DMA_CH3_TRFCNT	324	0x144	Channel Block Transfer Count
	DMA_CH3_REPCNT	326	0x146	Channel Repeat Count
	DMA_CH3_SRCADDR0	328	0x148	Channel Source Address 0
	DMA_CH3_SRCADDR1	329	0x149	Channel Source Address 1
	DMA_CH3_SRCADDR2	330	0x14A	Channel Source Address 2
	DMA_CH3_DESTADDR0	332	0x14C	Channel Destination Address 0
	DMA_CH3_DESTADDR1	333	0x14D	Channel Destination Address 1
	DMA_CH3_DESTADDR2	334	0x14E	Channel Destination Address 2
	Interrupt Vectors			
	DMA_CH0_vect	12	0xC	Channel 0 Interrupt
	DMA_CH1_vect	14	0xE	Channel 1 Interrupt
	DMA_CH2_vect	16	0x10	Channel 2 Interrupt

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
	DMA_CH3_vect	18	0x12	Channel 3 Interrupt
Event System	**Registers**			
	EVSYS_CH0MUX	384	0x180	Event Channel 0 Multiplexer
	EVSYS_CH1MUX	385	0x181	Event Channel 1 Multiplexer
	EVSYS_CH2MUX	386	0x182	Event Channel 2 Multiplexer
	EVSYS_CH3MUX	387	0x183	Event Channel 3 Multiplexer
	EVSYS_CH4MUX	388	0x184	Event Channel 4 Multiplexer
	EVSYS_CH5MUX	389	0x185	Event Channel 5 Multiplexer
	EVSYS_CH6MUX	390	0x186	Event Channel 6 Multiplexer
	EVSYS_CH7MUX	391	0x187	Event Channel 7 Multiplexer
	EVSYS_CH0CTRL	392	0x188	Channel 0 Control Register
	EVSYS_CH1CTRL	393	0x189	Channel 1 Control Register
	EVSYS_CH2CTRL	394	0x18A	Channel 2 Control Register
	EVSYS_CH3CTRL	395	0x18B	Channel 3 Control Register
	EVSYS_CH4CTRL	396	0x18C	Channel 4 Control Register
	EVSYS_CH5CTRL	397	0x18D	Channel 5 Control Register
	EVSYS_CH6CTRL	398	0x18E	Channel 6 Control Register
	EVSYS_CH7CTRL	399	0x18F	Channel 7 Control Register
	EVSYS_STROBE	400	0x190	Event Strobe
	EVSYS_DATA	401	0x191	Event Data
External Bus Interface (EBI)	**Registers**			
	EBI_CTRL	1088	0x440	Control
	EBI_SDRAMCTRLA	1089	0x441	SDRAM Control Register A
	EBI_REFRESH	1092	0x444	SDRAM Refresh Period
	EBI_INITDLY	1094	0x446	SDRAM Initialization Delay
	EBI_SDRAMCTRLB	1096	0x448	SDRAM Control Register B
	EBI_SDRAMCTRLC	1097	0x449	SDRAM Control Register C
	EBI_CS0_CTRLA	1104	0x450	Chip Select Control Register A
	EBI_CS0_CTRLB	1105	0x451	Chip Select Control Register B
	EBI_CS0_BASEADDR	1106	0x452	Chip Select Base Address
	EBI_CS1_CTRLA	1108	0x454	Chip Select Control Register A
	EBI_CS1_CTRLB	1109	0x455	Chip Select Control Register B
	EBI_CS1_BASEADDR	1110	0x456	Chip Select Base Address
	EBI_CS2_CTRLA	1112	0x458	Chip Select Control Register A
	EBI_CS2_CTRLB	1113	0x459	Chip Select Control Register B
	EBI_CS2_BASEADDR	1114	0x45A	Chip Select Base Address
	EBI_CS3_CTRLA	1116	0x45C	Chip Select Control Register A
	EBI_CS3_CTRLB	1117	0x45D	Chip Select Control Register B
	EBI_CS3_BASEADDR	1118	0x45E	Chip Select Base Address
gh Resolution Extension	**Registers**			
	HIRESC_CTRLA	2192	0x890	Control Register
	HIRESD_CTRLA	2448	0x990	Control Register
	HIRESE_CTRLA	2704	0xA90	Control Register

Thanks to Mason Turner!

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
Hi	HIRESF_CTRLA	2960	0xB90	Control Register
IR Com. Module	**Registers**			
	IRCOM_CTRL	2296	0x8F8	Control Register
	IRCOM_TXPLCTRL	2297	0x8F9	IrDA Transmitter Pulse Length Control Register
	IRCOM_RXPLCTRL	2298	0x8FA	IrDA Receiver Pulse Length Control Register
MCU Control	**Registers**			
	MCU_DEVID0	144	0x90	Device ID byte 0
	MCU_DEVID1	145	0x91	Device ID byte 1
	MCU_DEVID2	146	0x92	Device ID byte 2
	MCU_REVID	147	0x93	Revision ID
	MCU_JTAGUID	148	0x94	JTAG User ID
	MCU_MCUCR	150	0x96	MCU Control
	MCU_ANAINIT	151	0x97	Analog Startup Delay
	MCU_EVSYSLOCK	152	0x98	Event System Lock
	MCU_AWEXLOCK	153	0x99	AWEX Lock
Non-Volatile Memory	**Registers**			
	NVM_ADDR0	448	0x1C0	Address Register 0
	NVM_ADDR1	449	0x1C1	Address Register 1
	NVM_ADDR2	450	0x1C2	Address Register 2
	NVM_DATA0	452	0x1C4	Data Register 0
	NVM_DATA1	453	0x1C5	Data Register 1
	NVM_DATA2	454	0x1C6	Data Register 2
	NVM_CMD	458	0x1CA	Command
	NVM_CTRLA	459	0x1CB	Control Register A
	NVM_CTRLB	460	0x1CC	Control Register B
	NVM_INTCTRL	461	0x1CD	Interrupt Control
	NVM_STATUS	463	0x1CF	Status
	NVM_LOCKBITS	464	0x1D0	Lock Bits
	Interrupt Vectors			
	NVM_EE_vect	64	0x40	EE Interrupt
	NVM_SPM_vect	66	0x42	SPM Interrupt
Oscillator Control	**Registers**			
	OSC_CTRL	80	0x50	Control Register
	OSC_STATUS	81	0x51	Status Register
	OSC_XOSCCTRL	82	0x52	External Oscillator Control Register
	OSC_XOSCFAIL	83	0x53	Oscillator Failure Detection Register
	OSC_RC32KCAL	84	0x54	32.768 kHz Internal Oscillator Calibration Register
	OSC_PLLCTRL	85	0x55	PLL Control Register
	OSC_DFLLCTRL	86	0x56	DFLL Control Register
	Interrupt Vectors			
	OSC_OSCF_vect	2	0x2	Oscillator Failure Interrupt (NMI)
n	**Registers**			

Thanks to Mason Turner!

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
Port Configuratio	PORTCFG_MPCMASK	176	0xB0	Multi-pin Configuration Mask
	PORTCFG_VPCTRLA	178	0xB2	Virtual Port Control Register A
	PORTCFG_VPCTRLB	179	0xB3	Virtual Port Control Register B
	PORTCFG_CLKEVOUT	180	0xB4	Clock and Event Out Register
	PORTCFG_EVOUTSEL	182	0xB6	Event Output Select
PORT A	**Registers**			
	PORTA_DIR	1536	0x600	IO Port Data Direction
	PORTA_DIRSET	1537	0x601	IO Port Data Direction Set
	PORTA_DIRCLR	1538	0x602	IO Port Data Direction Clear
	PORTA_DIRTGL	1539	0x603	IO Port Data Direction Toggle
	PORTA_OUT	1540	0x604	IO Port Output
	PORTA_OUTSET	1541	0x605	IO Port Output Set
	PORTA_OUTCLR	1542	0x606	IO Port Output Clear
	PORTA_OUTTGL	1543	0x607	IO Port Output Toggle
	PORTA_IN	1544	0x608	IO port Input
	PORTA_INTCTRL	1545	0x609	Interrupt Control Register
	PORTA_INT0MASK	1546	0x60A	Port Interrupt 0 Mask
	PORTA_INT1MASK	1547	0x60B	Port Interrupt 1 Mask
	PORTA_INTFLAGS	1548	0x60C	Interrupt Flag Register
	PORTA_REMAP	1550	0x60E	IO Port Pin Remap Register
	PORTA_PIN0CTRL	1552	0x610	Pin 0 Control Register
	PORTA_PIN1CTRL	1553	0x611	Pin 1 Control Register
	PORTA_PIN2CTRL	1554	0x612	Pin 2 Control Register
	PORTA_PIN3CTRL	1555	0x613	Pin 3 Control Register
	PORTA_PIN4CTRL	1556	0x614	Pin 4 Control Register
	PORTA_PIN5CTRL	1557	0x615	Pin 5 Control Register
	PORTA_PIN6CTRL	1558	0x616	Pin 6 Control Register
	PORTA_PIN7CTRL	1559	0x617	Pin 7 Control Register
	Interrupt Vectors			
	PORTA_INT0_vect	132	0x84	External Interrupt 0
	PORTA_INT1_vect	134	0x86	External Interrupt 1
	Registers			
	PORTB_DIR	1568	0x620	IO Port Data Direction
	PORTB_DIRSET	1569	0x621	IO Port Data Direction Set
	PORTB_DIRCLR	1570	0x622	IO Port Data Direction Clear
	PORTB_DIRTGL	1571	0x623	IO Port Data Direction Toggle
	PORTB_OUT	1572	0x624	IO Port Output
	PORTB_OUTSET	1573	0x625	IO Port Output Set
	PORTB_OUTCLR	1574	0x626	IO Port Output Clear
	PORTB_OUTTGL	1575	0x627	IO Port Output Toggle
	PORTB_IN	1576	0x628	IO port Input
	PORTB_INTCTRL	1577	0x629	Interrupt Control Register
	PORTB_INT0MASK	1578	0x62A	Port Interrupt 0 Mask

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
PORT B	PORTB_INT1MASK	1579	0x62B	Port Interrupt 1 Mask
	PORTB_INTFLAGS	1580	0x62C	Interrupt Flag Register
	PORTB_REMAP	1582	0x62E	IO Port Pin Remap Register
	PORTB_PIN0CTRL	1584	0x630	Pin 0 Control Register
	PORTB_PIN1CTRL	1585	0x631	Pin 1 Control Register
	PORTB_PIN2CTRL	1586	0x632	Pin 2 Control Register
	PORTB_PIN3CTRL	1587	0x633	Pin 3 Control Register
	PORTB_PIN4CTRL	1588	0x634	Pin 4 Control Register
	PORTB_PIN5CTRL	1589	0x635	Pin 5 Control Register
	PORTB_PIN6CTRL	1590	0x636	Pin 6 Control Register
	PORTB_PIN7CTRL	1591	0x637	Pin 7 Control Register
	Interrupt Vectors			
	PORTB_INT0_vect	68	0x44	External Interrupt 0
	PORTB_INT1_vect	70	0x46	External Interrupt 1
PORT C	**Registers**			
	PORTC_DIR	1600	0x640	IO Port Data Direction
	PORTC_DIRSET	1601	0x641	IO Port Data Direction Set
	PORTC_DIRCLR	1602	0x642	IO Port Data Direction Clear
	PORTC_DIRTGL	1603	0x643	IO Port Data Direction Toggle
	PORTC_OUT	1604	0x644	IO Port Output
	PORTC_OUTSET	1605	0x645	IO Port Output Set
	PORTC_OUTCLR	1606	0x646	IO Port Output Clear
	PORTC_OUTTGL	1607	0x647	IO Port Output Toggle
	PORTC_IN	1608	0x648	IO port Input
	PORTC_INTCTRL	1609	0x649	Interrupt Control Register
	PORTC_INT0MASK	1610	0x64A	Port Interrupt 0 Mask
	PORTC_INT1MASK	1611	0x64B	Port Interrupt 1 Mask
	PORTC_INTFLAGS	1612	0x64C	Interrupt Flag Register
	PORTC_REMAP	1614	0x64E	IO Port Pin Remap Register
	PORTC_PIN0CTRL	1616	0x650	Pin 0 Control Register
	PORTC_PIN1CTRL	1617	0x651	Pin 1 Control Register
	PORTC_PIN2CTRL	1618	0x652	Pin 2 Control Register
	PORTC_PIN3CTRL	1619	0x653	Pin 3 Control Register
	PORTC_PIN4CTRL	1620	0x654	Pin 4 Control Register
	PORTC_PIN5CTRL	1621	0x655	Pin 5 Control Register
	PORTC_PIN6CTRL	1622	0x656	Pin 6 Control Register
	PORTC_PIN7CTRL	1623	0x657	Pin 7 Control Register
	Interrupt Vectors			
	PORTC_INT0_vect	4	0x4	External Interrupt 0
	PORTC_INT1_vect	6	0x6	External Interrupt 1
	Registers			
	PORTD_DIR	1632	0x660	IO Port Data Direction
	PORTD_DIRSET	1633	0x661	IO Port Data Direction Set

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
PORT D	PORTD_DIRCLR	1634	0x662	IO Port Data Direction Clear
	PORTD_DIRTGL	1635	0x663	IO Port Data Direction Toggle
	PORTD_OUT	1636	0x664	IO Port Output
	PORTD_OUTSET	1637	0x665	IO Port Output Set
	PORTD_OUTCLR	1638	0x666	IO Port Output Clear
	PORTD_OUTTGL	1639	0x667	IO Port Output Toggle
	PORTD_IN	1640	0x668	IO port Input
	PORTD_INTCTRL	1641	0x669	Interrupt Control Register
	PORTD_INT0MASK	1642	0x66A	Port Interrupt 0 Mask
	PORTD_INT1MASK	1643	0x66B	Port Interrupt 1 Mask
	PORTD_INTFLAGS	1644	0x66C	Interrupt Flag Register
	PORTD_REMAP	1646	0x66E	IO Port Pin Remap Register
	PORTD_PIN0CTRL	1648	0x670	Pin 0 Control Register
	PORTD_PIN1CTRL	1649	0x671	Pin 1 Control Register
	PORTD_PIN2CTRL	1650	0x672	Pin 2 Control Register
	PORTD_PIN3CTRL	1651	0x673	Pin 3 Control Register
	PORTD_PIN4CTRL	1652	0x674	Pin 4 Control Register
	PORTD_PIN5CTRL	1653	0x675	Pin 5 Control Register
	PORTD_PIN6CTRL	1654	0x676	Pin 6 Control Register
	PORTD_PIN7CTRL	1655	0x677	Pin 7 Control Register
	Interrupt Vectors			
	PORTD_INT0_vect	128	0x80	External Interrupt 0
	PORTD_INT1_vect	130	0x82	External Interrupt 1
PORT E	**Registers**			
	PORTE_DIR	1664	0x680	IO Port Data Direction
	PORTE_DIRSET	1665	0x681	IO Port Data Direction Set
	PORTE_DIRCLR	1666	0x682	IO Port Data Direction Clear
	PORTE_DIRTGL	1667	0x683	IO Port Data Direction Toggle
	PORTE_OUT	1668	0x684	IO Port Output
	PORTE_OUTSET	1669	0x685	IO Port Output Set
	PORTE_OUTCLR	1670	0x686	IO Port Output Clear
	PORTE_OUTTGL	1671	0x687	IO Port Output Toggle
	PORTE_IN	1672	0x688	IO port Input
	PORTE_INTCTRL	1673	0x689	Interrupt Control Register
	PORTE_INT0MASK	1674	0x68A	Port Interrupt 0 Mask
	PORTE_INT1MASK	1675	0x68B	Port Interrupt 1 Mask
	PORTE_INTFLAGS	1676	0x68C	Interrupt Flag Register
	PORTE_REMAP	1678	0x68E	IO Port Pin Remap Register
	PORTE_PIN0CTRL	1680	0x690	Pin 0 Control Register
	PORTE_PIN1CTRL	1681	0x691	Pin 1 Control Register
	PORTE_PIN2CTRL	1682	0x692	Pin 2 Control Register
	PORTE_PIN3CTRL	1683	0x693	Pin 3 Control Register
	PORTE_PIN4CTRL	1684	0x694	Pin 4 Control Register

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
	PORTE_PIN5CTRL	1685	0x695	Pin 5 Control Register
	PORTE_PIN6CTRL	1686	0x696	Pin 6 Control Register
	PORTE_PIN7CTRL	1687	0x697	Pin 7 Control Register
	Interrupt Vectors			
	PORTE_INT0_vect	86	0x56	External Interrupt 0
	PORTE_INT1_vect	88	0x58	External Interrupt 1
PORT F	**Registers**			
	PORTF_DIR	1696	0x6A0	IO Port Data Direction
	PORTF_DIRSET	1697	0x6A1	IO Port Data Direction Set
	PORTF_DIRCLR	1698	0x6A2	IO Port Data Direction Clear
	PORTF_DIRTGL	1699	0x6A3	IO Port Data Direction Toggle
	PORTF_OUT	1700	0x6A4	IO Port Output
	PORTF_OUTSET	1701	0x6A5	IO Port Output Set
	PORTF_OUTCLR	1702	0x6A6	IO Port Output Clear
	PORTF_OUTTGL	1703	0x6A7	IO Port Output Toggle
	PORTF_IN	1704	0x6A8	IO port Input
	PORTF_INTCTRL	1705	0x6A9	Interrupt Control Register
	PORTF_INT0MASK	1706	0x6AA	Port Interrupt 0 Mask
	PORTF_INT1MASK	1707	0x6AB	Port Interrupt 1 Mask
	PORTF_INTFLAGS	1708	0x6AC	Interrupt Flag Register
	PORTF_REMAP	1710	0x6AE	IO Port Pin Remap Register
	PORTF_PIN0CTRL	1712	0x6B0	Pin 0 Control Register
	PORTF_PIN1CTRL	1713	0x6B1	Pin 1 Control Register
	PORTF_PIN2CTRL	1714	0x6B2	Pin 2 Control Register
	PORTF_PIN3CTRL	1715	0x6B3	Pin 3 Control Register
	PORTF_PIN4CTRL	1716	0x6B4	Pin 4 Control Register
	PORTF_PIN5CTRL	1717	0x6B5	Pin 5 Control Register
	PORTF_PIN6CTRL	1718	0x6B6	Pin 6 Control Register
	PORTF_PIN7CTRL	1719	0x6B7	Pin 7 Control Register
	Interrupt Vectors			
	PORTF_INT0_vect	208	0xD0	External Interrupt 0
	PORTF_INT1_vect	210	0xD2	External Interrupt 1
	Registers			
	PORTH_DIR	1760	0x6E0	IO Port Data Direction
	PORTH_DIRSET	1761	0x6E1	IO Port Data Direction Set
	PORTH_DIRCLR	1762	0x6E2	IO Port Data Direction Clear
	PORTH_DIRTGL	1763	0x6E3	IO Port Data Direction Toggle
	PORTH_OUT	1764	0x6E4	IO Port Output
	PORTH_OUTSET	1765	0x6E5	IO Port Output Set
	PORTH_OUTCLR	1766	0x6E6	IO Port Output Clear
	PORTH_OUTTGL	1767	0x6E7	IO Port Output Toggle
	PORTH_IN	1768	0x6E8	IO port Input
	PORTH_INTCTRL	1769	0x6E9	Interrupt Control Register

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
PORT H	PORTH_INT0MASK	1770	0x6EA	Port Interrupt 0 Mask
	PORTH_INT1MASK	1771	0x6EB	Port Interrupt 1 Mask
	PORTH_INTFLAGS	1772	0x6EC	Interrupt Flag Register
	PORTH_REMAP	1774	0x6EE	IO Port Pin Remap Register
	PORTH_PIN0CTRL	1776	0x6F0	Pin 0 Control Register
	PORTH_PIN1CTRL	1777	0x6F1	Pin 1 Control Register
	PORTH_PIN2CTRL	1778	0x6F2	Pin 2 Control Register
	PORTH_PIN3CTRL	1779	0x6F3	Pin 3 Control Register
	PORTH_PIN4CTRL	1780	0x6F4	Pin 4 Control Register
	PORTH_PIN5CTRL	1781	0x6F5	Pin 5 Control Register
	PORTH_PIN6CTRL	1782	0x6F6	Pin 6 Control Register
	PORTH_PIN7CTRL	1783	0x6F7	Pin 7 Control Register
	Interrupt Vectors			
	PORTH_INT0_vect	192	0xC0	External Interrupt 0
	PORTH_INT1_vect	194	0xC2	External Interrupt 1
PORT J	**Registers**			
	PORTJ_DIR	1792	0x700	IO Port Data Direction
	PORTJ_DIRSET	1793	0x701	IO Port Data Direction Set
	PORTJ_DIRCLR	1794	0x702	IO Port Data Direction Clear
	PORTJ_DIRTGL	1795	0x703	IO Port Data Direction Toggle
	PORTJ_OUT	1796	0x704	IO Port Output
	PORTJ_OUTSET	1797	0x705	IO Port Output Set
	PORTJ_OUTCLR	1798	0x706	IO Port Output Clear
	PORTJ_OUTTGL	1799	0x707	IO Port Output Toggle
	PORTJ_IN	1800	0x708	IO port Input
	PORTJ_INTCTRL	1801	0x709	Interrupt Control Register
	PORTJ_INT0MASK	1802	0x70A	Port Interrupt 0 Mask
	PORTJ_INT1MASK	1803	0x70B	Port Interrupt 1 Mask
	PORTJ_INTFLAGS	1804	0x70C	Interrupt Flag Register
	PORTJ_REMAP	1806	0x70E	IO Port Pin Remap Register
	PORTJ_PIN0CTRL	1808	0x710	Pin 0 Control Register
	PORTJ_PIN1CTRL	1809	0x711	Pin 1 Control Register
	PORTJ_PIN2CTRL	1810	0x712	Pin 2 Control Register
	PORTJ_PIN3CTRL	1811	0x713	Pin 3 Control Register
	PORTJ_PIN4CTRL	1812	0x714	Pin 4 Control Register
	PORTJ_PIN5CTRL	1813	0x715	Pin 5 Control Register
	PORTJ_PIN6CTRL	1814	0x716	Pin 6 Control Register
	PORTJ_PIN7CTRL	1815	0x717	Pin 7 Control Register
	Interrupt Vectors			
	PORTJ_INT0_vect	196	0xC4	External Interrupt 0
	PORTJ_INT1_vect	198	0xC6	External Interrupt 1
	Registers			
	PORTK_DIR	1824	0x720	IO Port Data Direction

Thanks to Mason Turner!

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
PORT K	PORTK_DIRSET	1825	0x721	IO Port Data Direction Set
	PORTK_DIRCLR	1826	0x722	IO Port Data Direction Clear
	PORTK_DIRTGL	1827	0x723	IO Port Data Direction Toggle
	PORTK_OUT	1828	0x724	IO Port Output
	PORTK_OUTSET	1829	0x725	IO Port Output Set
	PORTK_OUTCLR	1830	0x726	IO Port Output Clear
	PORTK_OUTTGL	1831	0x727	IO Port Output Toggle
	PORTK_IN	1832	0x728	IO port Input
	PORTK_INTCTRL	1833	0x729	Interrupt Control Register
	PORTK_INT0MASK	1834	0x72A	Port Interrupt 0 Mask
	PORTK_INT1MASK	1835	0x72B	Port Interrupt 1 Mask
	PORTK_INTFLAGS	1836	0x72C	Interrupt Flag Register
	PORTK_REMAP	1838	0x72E	IO Port Pin Remap Register
	PORTK_PIN0CTRL	1840	0x730	Pin 0 Control Register
	PORTK_PIN1CTRL	1841	0x731	Pin 1 Control Register
	PORTK_PIN2CTRL	1842	0x732	Pin 2 Control Register
	PORTK_PIN3CTRL	1843	0x733	Pin 3 Control Register
	PORTK_PIN4CTRL	1844	0x734	Pin 4 Control Register
	PORTK_PIN5CTRL	1845	0x735	Pin 5 Control Register
	PORTK_PIN6CTRL	1846	0x736	Pin 6 Control Register
	PORTK_PIN7CTRL	1847	0x737	Pin 7 Control Register
	Interrupt Vectors			
	PORTK_INT0_vect	200	0xC8	External Interrupt 0
	PORTK_INT1_vect	202	0xCA	External Interrupt 1
PORT Q	**Registers**			
	PORTQ_DIR	1984	0x7C0	IO Port Data Direction
	PORTQ_DIRSET	1985	0x7C1	IO Port Data Direction Set
	PORTQ_DIRCLR	1986	0x7C2	IO Port Data Direction Clear
	PORTQ_DIRTGL	1987	0x7C3	IO Port Data Direction Toggle
	PORTQ_OUT	1988	0x7C4	IO Port Output
	PORTQ_OUTSET	1989	0x7C5	IO Port Output Set
	PORTQ_OUTCLR	1990	0x7C6	IO Port Output Clear
	PORTQ_OUTTGL	1991	0x7C7	IO Port Output Toggle
	PORTQ_IN	1992	0x7C8	IO port Input
	PORTQ_INTCTRL	1993	0x7C9	Interrupt Control Register
	PORTQ_INT0MASK	1994	0x7CA	Port Interrupt 0 Mask
	PORTQ_INT1MASK	1995	0x7CB	Port Interrupt 1 Mask
	PORTQ_INTFLAGS	1996	0x7CC	Interrupt Flag Register
	PORTQ_REMAP	1998	0x7CE	IO Port Pin Remap Register
	PORTQ_PIN0CTRL	2000	0x7D0	Pin 0 Control Register
	PORTQ_PIN1CTRL	2001	0x7D1	Pin 1 Control Register
	PORTQ_PIN2CTRL	2002	0x7D2	Pin 2 Control Register
	PORTQ_PIN3CTRL	2003	0x7D3	Pin 3 Control Register

Thanks to Mason Turner!

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
	PORTQ_PIN4CTRL	2004	0x7D4	Pin 4 Control Register
	PORTQ_PIN5CTRL	2005	0x7D5	Pin 5 Control Register
	PORTQ_PIN6CTRL	2006	0x7D6	Pin 6 Control Register
	PORTQ_PIN7CTRL	2007	0x7D7	Pin 7 Control Register
	Interrupt Vectors			
	PORTQ_INT0_vect	188	0xBC	External Interrupt 0
	PORTQ_INT1_vect	190	0xBE	External Interrupt 1
PORT R	**Registers**			
	PORTR_DIR	2016	0x7E0	IO Port Data Direction
	PORTR_DIRSET	2017	0x7E1	IO Port Data Direction Set
	PORTR_DIRCLR	2018	0x7E2	IO Port Data Direction Clear
	PORTR_DIRTGL	2019	0x7E3	IO Port Data Direction Toggle
	PORTR_OUT	2020	0x7E4	IO Port Output
	PORTR_OUTSET	2021	0x7E5	IO Port Output Set
	PORTR_OUTCLR	2022	0x7E6	IO Port Output Clear
	PORTR_OUTTGL	2023	0x7E7	IO Port Output Toggle
	PORTR_IN	2024	0x7E8	IO port Input
	PORTR_INTCTRL	2025	0x7E9	Interrupt Control Register
	PORTR_INT0MASK	2026	0x7EA	Port Interrupt 0 Mask
	PORTR_INT1MASK	2027	0x7EB	Port Interrupt 1 Mask
	PORTR_INTFLAGS	2028	0x7EC	Interrupt Flag Register
	PORTR_REMAP	2030	0x7EE	IO Port Pin Remap Register
	PORTR_PIN0CTRL	2032	0x7F0	Pin 0 Control Register
	PORTR_PIN1CTRL	2033	0x7F1	Pin 1 Control Register
	PORTR_PIN2CTRL	2034	0x7F2	Pin 2 Control Register
	PORTR_PIN3CTRL	2035	0x7F3	Pin 3 Control Register
	PORTR_PIN4CTRL	2036	0x7F4	Pin 4 Control Register
	PORTR_PIN5CTRL	2037	0x7F5	Pin 5 Control Register
	PORTR_PIN6CTRL	2038	0x7F6	Pin 6 Control Register
	PORTR_PIN7CTRL	2039	0x7F7	Pin 7 Control Register
	Interrupt Vectors			
	PORTR_INT0_vect	8	0x8	External Interrupt 0
	PORTR_INT1_vect	10	0xA	External Interrupt 1
PMIC	**Registers**			
	PMIC_STATUS	160	0xA0	Status Register
	PMIC_INTPRI	161	0xA1	Interrupt Priority
	PMIC_CTRL	162	0xA2	Control Register
Power Reduction	**Registers**			
	PR_PRGEN	112	0x70	General Power Reduction
	PR_PRPA	113	0x71	Power Reduction Port A
	PR_PRPB	114	0x72	Power Reduction Port B
	PR_PRPC	115	0x73	Power Reduction Port C
	PR_PRPD	116	0x74	Power Reduction Port D

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
P	PR_PRPE	117	0x75	Power Reduction Port E
	PR_PRPF	118	0x76	Power Reduction Port F
Reset Cont.	**Registers**			
	RST_STATUS	120	0x78	Status Register
	RST_CTRL	121	0x79	Control Register
Real Time Clock	**Registers**			
	RTC_CTRL	1024	0x400	Control Register
	RTC_STATUS	1025	0x401	Status Register
	RTC_INTCTRL	1026	0x402	Interrupt Control Register
	RTC_INTFLAGS	1027	0x403	Interrupt Flags
	RTC_TEMP	1028	0x404	Temporary register
	RTC_CNT	1032	0x408	Count Register
	RTC_PER	1034	0x40A	Period Register
	RTC_COMP	1036	0x40C	Compare Register
	Interrupt Vectors			
	RTC_COMP_vect	22	0x16	Compare Interrupt
	RTC_OVF_vect	20	0x14	Overflow Interrupt
Serial Peripheral Interface C	**Registers**			
	SPIC_CTRL	2240	0x8C0	Control Register
	SPIC_INTCTRL	2241	0x8C1	Interrupt Control Register
	SPIC_STATUS	2242	0x8C2	Status Register
	SPIC_DATA	2243	0x8C3	Data Register
	Interrupt Vectors			
	SPIC_INT_vect	48	0x30	SPI Interrupt
Serial Peripheral Interface D	**Registers**			
	SPID_CTRL	2496	0x9C0	Control Register
	SPID_INTCTRL	2497	0x9C1	Interrupt Control Register
	SPID_STATUS	2498	0x9C2	Status Register
	SPID_DATA	2499	0x9C3	Data Register
	Interrupt Vectors			
	SPID_INT_vect	174	0xAE	SPI Interrupt
Serial Peripheral Interface E	**Registers**			
	SPIE_CTRL	2752	0xAC0	Control Register
	SPIE_INTCTRL	2753	0xAC1	Interrupt Control Register
	SPIE_STATUS	2754	0xAC2	Status Register
	SPIE_DATA	2755	0xAC3	Data Register
	Interrupt Vectors			
	SPIE_INT_vect	114	0x72	SPI Interrupt
rial Peripheral Interface F	**Registers**			
	SPIF_CTRL	3008		Control Register
	SPIF_INTCTRL	3009		Interrupt Control Register
	SPIF_STATUS	3010		Status Register
	SPIF_DATA	3011		Data Register

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
Se	**Interrupt Vectors**			
	SPIF_INT_vect	236	0xEC	SPI Interrupt
Slp Cntr.	**Registers**			
	SLEEP_CTRL	72		Control Register
Timer/Counter 0 on Port C	**Registers**			
	TCC0_CTRLA	2048	0x800	Control Register A
	TCC0_CTRLB	2049	0x801	Control Register B
	TCC0_CTRLC	2050	0x802	Control register C
	TCC0_CTRLD	2051	0x803	Control Register D
	TCC0_CTRLE	2052	0x804	Control Register E
	TCC0_INTCTRLA	2054	0x806	Interrupt Control Register A
	TCC0_INTCTRLB	2055	0x807	Interrupt Control Register B
	TCC0_CTRLFCLR	2056	0x808	Control Register F Clear
	TCC0_CTRLFSET	2057	0x809	Control Register F Set
	TCC0_CTRLGCLR	2058	0x80A	Control Register G Clear
	TCC0_CTRLGSET	2059	0x80B	Control Register G Set
	TCC0_INTFLAGS	2060	0x80C	Interrupt Flag Register
	TCC0_TEMP	2063	0x80F	Temporary Register For 16-bit Access
	TCC0_CNT	2080	0x820	Count
	TCC0_PER	2086	0x826	Period
	TCC0_CCA	2088	0x828	Compare or Capture A
	TCC0_CCB	2090	0x82A	Compare or Capture B
	TCC0_CCC	2092	0x82C	Compare or Capture C
	TCC0_CCD	2094	0x82E	Compare or Capture D
	TCC0_PERBUF	2102	0x836	Period Buffer
	TCC0_CCABUF	2104	0x838	Compare Or Capture A Buffer
	TCC0_CCBBUF	2106	0x83A	Compare Or Capture B Buffer
	TCC0_CCCBUF	2108	0x83C	Compare Or Capture C Buffer
	TCC0_CCDBUF	2110	0x83E	Compare Or Capture D Buffer
	Interrupt Vectors			
	TCC0_CCA_vect	32	0x20	Compare or Capture A Interrupt
	TCC0_CCB_vect	34	0x22	Compare or Capture B Interrupt
	TCC0_CCC_vect	36	0x24	Compare or Capture C Interrupt
	TCC0_CCD_vect	38	0x26	Compare or Capture D Interrupt
	TCC0_ERR_vect	30	0x1E	Error Interrupt
	TCC0_OVF_vect	28	0x1C	Overflow Interrupt
	Registers			
	TCC1_CTRLA	2112	0x840	Control Register A
	TCC1_CTRLB	2113	0x841	Control Register B
	TCC1_CTRLC	2114	0x842	Control register C
	TCC1_CTRLD	2115	0x843	Control Register D
	TCC1_CTRLE	2116	0x844	Control Register E
	TCC1_INTCTRLA	2118	0x846	Interrupt Control Register A

Thanks to Mason Turner!

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
Timer/Counter 1 on Port C	TCC1_INTCTRLB	2119	0x847	Interrupt Control Register B
	TCC1_CTRLFCLR	2120	0x848	Control Register F Clear
	TCC1_CTRLFSET	2121	0x849	Control Register F Set
	TCC1_CTRLGCLR	2122	0x84A	Control Register G Clear
	TCC1_CTRLGSET	2123	0x84B	Control Register G Set
	TCC1_INTFLAGS	2124	0x84C	Interrupt Flag Register
	TCC1_TEMP	2127	0x84F	Temporary Register For 16-bit Access
	TCC1_CNT	2144	0x860	Count
	TCC1_PER	2150	0x866	Period
	TCC1_CCA	2152	0x868	Compare or Capture A
	TCC1_CCB	2154	0x86A	Compare or Capture B
	TCC1_PERBUF	2166	0x876	Period Buffer
	TCC1_CCABUF	2168	0x878	Compare Or Capture A Buffer
	TCC1_CCBBUF	2170	0x87A	Compare Or Capture B Buffer
	Interrupt Vectors			
	TCC1_CCA_vect	44	0x2C	Compare or Capture A Interrupt
	TCC1_CCB_vect	46	0x2E	Compare or Capture B Interrupt
	TCC1_ERR_vect	42	0x2A	Error Interrupt
	TCC1_OVF_vect	40	0x28	Overflow Interrupt
Timer/Counter 2 on Port C	**Registers**			
	TCC2_CTRLA	2048	0x800	Control Register A
	TCC2_CTRLB	2049	0x801	Control Register B
	TCC2_CTRLC	2050	0x802	Control register C
	TCC2_CTRLE	2052	0x804	Control Register E
	TCC2_INTCTRLA	2054	0x806	Interrupt Control Register A
	TCC2_INTCTRLB	2055	0x807	Interrupt Control Register B
	TCC2_CTRLF	2057	0x809	Control Register F
	TCC2_INTFLAGS	2060	0x80C	Interrupt Flag Register
	TCC2_LCNT	2080	0x820	Low Byte Count
	TCC2_HCNT	2081	0x821	High Byte Count
	TCC2_LPER	2086	0x826	Low Byte Period
	TCC2_HPER	2087	0x827	High Byte Period
	TCC2_LCMPA	2088	0x828	Low Byte Compare A
	TCC2_HCMPA	2089	0x829	High Byte Compare A
	TCC2_LCMPB	2090	0x82A	Low Byte Compare B
	TCC2_HCMPB	2091	0x82B	High Byte Compare B
	TCC2_LCMPC	2092	0x82C	Low Byte Compare C
	TCC2_HCMPC	2093	0x82D	High Byte Compare C
	TCC2_LCMPD	2094	0x82E	Low Byte Compare D
	TCC2_HCMPD	2095	0x82F	High Byte Compare D
	Interrupt Vectors			
	TCC2_HUNF_vect	30	0x1E	High Byte Underflow Interrupt
	TCC2_LCMPA_vect	32	0x20	Low Byte Compare A Interrupt

Thanks to Mason Turner!

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
	TCC2_LCMPB_vect	34	0x22	Low Byte Compare B Interrupt
	TCC2_LCMPC_vect	36	0x24	Low Byte Compare C Interrupt
	TCC2_LCMPD_vect	38	0x26	Low Byte Compare D Interrupt
	TCC2_LUNF_vect	28	0x1C	Low Byte Underflow Interrupt
Timer/Counter 0 on Port D	**Registers**			
	TCD0_CTRLA	2304	0x900	Control Register A
	TCD0_CTRLB	2305	0x901	Control Register B
	TCD0_CTRLC	2306	0x902	Control register C
	TCD0_CTRLD	2307	0x903	Control Register D
	TCD0_CTRLE	2308	0x904	Control Register E
	TCD0_INTCTRLA	2310	0x906	Interrupt Control Register A
	TCD0_INTCTRLB	2311	0x907	Interrupt Control Register B
	TCD0_CTRLFCLR	2312	0x908	Control Register F Clear
	TCD0_CTRLFSET	2313	0x909	Control Register F Set
	TCD0_CTRLGCLR	2314	0x90A	Control Register G Clear
	TCD0_CTRLGSET	2315	0x90B	Control Register G Set
	TCD0_INTFLAGS	2316	0x90C	Interrupt Flag Register
	TCD0_TEMP	2319	0x90F	Temporary Register For 16-bit Access
	TCD0_CNT	2336	0x920	Count
	TCD0_PER	2342	0x926	Period
	TCD0_CCA	2344	0x928	Compare or Capture A
	TCD0_CCB	2346	0x92A	Compare or Capture B
	TCD0_CCC	2348	0x92C	Compare or Capture C
	TCD0_CCD	2350	0x92E	Compare or Capture D
	TCD0_PERBUF	2358	0x936	Period Buffer
	TCD0_CCABUF	2360	0x938	Compare Or Capture A Buffer
	TCD0_CCBBUF	2362	0x93A	Compare Or Capture B Buffer
	TCD0_CCCBUF	2364	0x93C	Compare Or Capture C Buffer
	TCD0_CCDBUF	2366	0x93E	Compare Or Capture D Buffer
	Interrupt Vectors			
	TCD0_CCA_vect	158	0x9E	Compare or Capture A Interrupt
	TCD0_CCB_vect	160	0xA0	Compare or Capture B Interrupt
	TCD0_CCC_vect	162	0xA2	Compare or Capture C Interrupt
	TCD0_CCD_vect	164	0xA4	Compare or Capture D Interrupt
	TCD0_ERR_vect	156	0x9C	Error Interrupt
	TCD0_OVF_vect	154	0x9A	Overflow Interrupt
	Registers			
	TCD1_CTRLA	2368	0x940	Control Register A
	TCD1_CTRLB	2369	0x941	Control Register B
	TCD1_CTRLC	2370	0x942	Control register C
	TCD1_CTRLD	2371	0x943	Control Register D
	TCD1_CTRLE	2372	0x944	Control Register E
	TCD1_INTCTRLA	2374	0x946	Interrupt Control Register A

Thanks to Mason Turner!

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
Timer/Counter 1 on Port D	TCD1_INTCTRLB	2375	0x947	Interrupt Control Register B
	TCD1_CTRLFCLR	2376	0x948	Control Register F Clear
	TCD1_CTRLFSET	2377	0x949	Control Register F Set
	TCD1_CTRLGCLR	2378	0x94A	Control Register G Clear
	TCD1_CTRLGSET	2379	0x94B	Control Register G Set
	TCD1_INTFLAGS	2380	0x94C	Interrupt Flag Register
	TCD1_TEMP	2383	0x94F	Temporary Register For 16-bit Access
	TCD1_CNT	2400	0x960	Count
	TCD1_PER	2406	0x966	Period
	TCD1_CCA	2408	0x968	Compare or Capture A
	TCD1_CCB	2410	0x96A	Compare or Capture B
	TCD1_PERBUF	2422	0x976	Period Buffer
	TCD1_CCABUF	2424	0x978	Compare Or Capture A Buffer
	TCD1_CCBBUF	2426	0x97A	Compare Or Capture B Buffer
	Interrupt Vectors			
	TCD1_CCA_vect	170	0xAA	Compare or Capture A Interrupt
	TCD1_CCB_vect	172	0xAC	Compare or Capture B Interrupt
	TCD1_ERR_vect	168	0xA8	Error Interrupt
	TCD1_OVF_vect	166	0xA6	Overflow Interrupt
Timer/Counter 2 on Port D	**Registers**			
	TCD2_CTRLA	2304	0x900	Control Register A
	TCD2_CTRLB	2305	0x901	Control Register B
	TCD2_CTRLC	2306	0x902	Control register C
	TCD2_CTRLE	2308	0x904	Control Register E
	TCD2_INTCTRLA	2310	0x906	Interrupt Control Register A
	TCD2_INTCTRLB	2311	0x907	Interrupt Control Register B
	TCD2_CTRLF	2313	0x909	Control Register F
	TCD2_INTFLAGS	2316	0x90C	Interrupt Flag Register
	TCD2_LCNT	2336	0x920	Low Byte Count
	TCD2_HCNT	2337	0x921	High Byte Count
	TCD2_LPER	2342	0x926	Low Byte Period
	TCD2_HPER	2343	0x927	High Byte Period
	TCD2_LCMPA	2344	0x928	Low Byte Compare A
	TCD2_HCMPA	2345	0x929	High Byte Compare A
	TCD2_LCMPB	2346	0x92A	Low Byte Compare B
	TCD2_HCMPB	2347	0x92B	High Byte Compare B
	TCD2_LCMPC	2348	0x92C	Low Byte Compare C
	TCD2_HCMPC	2349	0x92D	High Byte Compare C
	TCD2_LCMPD	2350	0x92E	Low Byte Compare D
	TCD2_HCMPD	2351	0x92F	High Byte Compare D
	Interrupt Vectors			
	TCD2_HUNF_vect	156	0x9C	High Byte Underflow Interrupt
	TCD2_LCMPA_vect	158	0x9E	Low Byte Compare A Interrupt

Thanks to Mason Turner!

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
	TCD2_LCMPB_vect	160	0xA0	Low Byte Compare B Interrupt
	TCD2_LCMPC_vect	162	0xA2	Low Byte Compare C Interrupt
	TCD2_LCMPD_vect	164	0xA4	Low Byte Compare D Interrupt
	TCD2_LUNF_vect	154	0x9A	Low Byte Underflow Interrupt
Timer/Counter 0 on Port E	**Registers**			
	TCE0_CTRLA	2560	0xA00	Control Register A
	TCE0_CTRLB	2561	0xA01	Control Register B
	TCE0_CTRLC	2562	0xA02	Control register C
	TCE0_CTRLD	2563	0xA03	Control Register D
	TCE0_CTRLE	2564	0xA04	Control Register E
	TCE0_INTCTRLA	2566	0xA06	Interrupt Control Register A
	TCE0_INTCTRLB	2567	0xA07	Interrupt Control Register B
	TCE0_CTRLFCLR	2568	0xA08	Control Register F Clear
	TCE0_CTRLFSET	2569	0xA09	Control Register F Set
	TCE0_CTRLGCLR	2570	0xA0A	Control Register G Clear
	TCE0_CTRLGSET	2571	0xA0B	Control Register G Set
	TCE0_INTFLAGS	2572	0xA0C	Interrupt Flag Register
	TCE0_TEMP	2575	0xA0F	Temporary Register For 16-bit Access
	TCE0_CNT	2592	0xA20	Count
	TCE0_PER	2598	0xA26	Period
	TCE0_CCA	2600	0xA28	Compare or Capture A
	TCE0_CCB	2602	0xA2A	Compare or Capture B
	TCE0_CCC	2604	0xA2C	Compare or Capture C
	TCE0_CCD	2606	0xA2E	Compare or Capture D
	TCE0_PERBUF	2614	0xA36	Period Buffer
	TCE0_CCABUF	2616	0xA38	Compare Or Capture A Buffer
	TCE0_CCBBUF	2618	0xA3A	Compare Or Capture B Buffer
	TCE0_CCCBUF	2620	0xA3C	Compare Or Capture C Buffer
	TCE0_CCDBUF	2622	0xA3E	Compare Or Capture D Buffer
	Interrupt Vectors			
	TCE0_CCA_vect	98	0x62	Compare or Capture A Interrupt
	TCE0_CCB_vect	100	0x64	Compare or Capture B Interrupt
	TCE0_CCC_vect	102	0x66	Compare or Capture C Interrupt
	TCE0_CCD_vect	104	0x68	Compare or Capture D Interrupt
	TCE0_ERR_vect	96	0x60	Error Interrupt
	TCE0_OVF_vect	94	0x5E	Overflow Interrupt
	Registers			
	TCE1_CTRLA	2624	0xA40	Control Register A
	TCE1_CTRLB	2625	0xA41	Control Register B
	TCE1_CTRLC	2626	0xA42	Control register C
	TCE1_CTRLD	2627	0xA43	Control Register D
	TCE1_CTRLE	2628	0xA44	Control Register E
	TCE1_INTCTRLA	2630	0xA46	Interrupt Control Register A

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
Timer/Counter 1 on Port E	TCE1_INTCTRLB	2631	0xA47	Interrupt Control Register B
	TCE1_CTRLFCLR	2632	0xA48	Control Register F Clear
	TCE1_CTRLFSET	2633	0xA49	Control Register F Set
	TCE1_CTRLGCLR	2634	0xA4A	Control Register G Clear
	TCE1_CTRLGSET	2635	0xA4B	Control Register G Set
	TCE1_INTFLAGS	2636	0xA4C	Interrupt Flag Register
	TCE1_TEMP	2639	0xA4F	Temporary Register For 16-bit Access
	TCE1_CNT	2656	0xA60	Count
	TCE1_PER	2662	0xA66	Period
	TCE1_CCA	2664	0xA68	Compare or Capture A
	TCE1_CCB	2666	0xA6A	Compare or Capture B
	TCE1_PERBUF	2678	0xA76	Period Buffer
	TCE1_CCABUF	2680	0xA78	Compare Or Capture A Buffer
	TCE1_CCBBUF	2682	0xA7A	Compare Or Capture B Buffer
	Interrupt Vectors			
	TCE1_CCA_vect	110	0x6E	Compare or Capture A Interrupt
	TCE1_CCB_vect	112	0x70	Compare or Capture B Interrupt
	TCE1_ERR_vect	108	0x6C	Error Interrupt
	TCE1_OVF_vect	106	0x6A	Overflow Interrupt
Timer/Counter 2 on Port E	**Registers**			
	TCE2_CTRLA	2560	0xA00	Control Register A
	TCE2_CTRLB	2561	0xA01	Control Register B
	TCE2_CTRLC	2562	0xA02	Control register C
	TCE2_CTRLE	2564	0xA04	Control Register E
	TCE2_INTCTRLA	2566	0xA06	Interrupt Control Register A
	TCE2_INTCTRLB	2567	0xA07	Interrupt Control Register B
	TCE2_CTRLF	2569	0xA09	Control Register F
	TCE2_INTFLAGS	2572	0xA0C	Interrupt Flag Register
	TCE2_LCNT	2592	0xA20	Low Byte Count
	TCE2_HCNT	2593	0xA21	High Byte Count
	TCE2_LPER	2598	0xA26	Low Byte Period
	TCE2_HPER	2599	0xA27	High Byte Period
	TCE2_LCMPA	2600	0xA28	Low Byte Compare A
	TCE2_HCMPA	2601	0xA29	High Byte Compare A
	TCE2_LCMPB	2602	0xA2A	Low Byte Compare B
	TCE2_HCMPB	2603	0xA2B	High Byte Compare B
	TCE2_LCMPC	2604	0xA2C	Low Byte Compare C
	TCE2_HCMPC	2605	0xA2D	High Byte Compare C
	TCE2_LCMPD	2606	0xA2E	Low Byte Compare D
	TCE2_HCMPD	2607	0xA2F	High Byte Compare D
	Interrupt Vectors			
	TCE2_HUNF_vect	96	0x60	High Byte Underflow Interrupt
	TCE2_LCMPA_vect	98	0x62	Low Byte Compare A Interrupt

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
	TCE2_LCMPB_vect	100	0x64	Low Byte Compare B Interrupt
	TCE2_LCMPC_vect	102	0x66	Low Byte Compare C Interrupt
	TCE2_LCMPD_vect	104	0x68	Low Byte Compare D Interrupt
	TCE2_LUNF_vect	94	0x5E	Low Byte Underflow Interrupt
Timer/Counter 0 on Port F	**Registers**			
	TCF0_CTRLA	2816	0xB00	Control Register A
	TCF0_CTRLB	2817	0xB01	Control Register B
	TCF0_CTRLC	2818	0xB02	Control register C
	TCF0_CTRLD	2819	0xB03	Control Register D
	TCF0_CTRLE	2820	0xB04	Control Register E
	TCF0_INTCTRLA	2822	0xB06	Interrupt Control Register A
	TCF0_INTCTRLB	2823	0xB07	Interrupt Control Register B
	TCF0_CTRLFCLR	2824	0xB08	Control Register F Clear
	TCF0_CTRLFSET	2825	0xB09	Control Register F Set
	TCF0_CTRLGCLR	2826	0xB0A	Control Register G Clear
	TCF0_CTRLGSET	2827	0xB0B	Control Register G Set
	TCF0_INTFLAGS	2828	0xB0C	Interrupt Flag Register
	TCF0_TEMP	2831	0xB0F	Temporary Register For 16-bit Access
	TCF0_CNT	2848	0xB20	Count
	TCF0_PER	2854	0xB26	Period
	TCF0_CCA	2856	0xB28	Compare or Capture A
	TCF0_CCB	2858	0xB2A	Compare or Capture B
	TCF0_CCC	2860	0xB2C	Compare or Capture C
	TCF0_CCD	2862	0xB2E	Compare or Capture D
	TCF0_PERBUF	2870	0xB36	Period Buffer
	TCF0_CCABUF	2872	0xB38	Compare Or Capture A Buffer
	TCF0_CCBBUF	2874	0xB3A	Compare Or Capture B Buffer
	TCF0_CCCBUF	2876	0xB3C	Compare Or Capture C Buffer
	TCF0_CCDBUF	2878	0xB3E	Compare Or Capture D Buffer
	Interrupt Vectors			
	TCF0_CCA_vect	220	0xDC	Compare or Capture A Interrupt
	TCF0_CCB_vect	222	0xDE	Compare or Capture B Interrupt
	TCF0_CCC_vect	224	0xE0	Compare or Capture C Interrupt
	TCF0_CCD_vect	226	0xE2	Compare or Capture D Interrupt
	TCF0_ERR_vect	218	0xDA	Error Interrupt
	TCF0_OVF_vect	216	0xD8	Overflow Interrupt
	Registers			
	TCF1_CTRLA	2880	0xB40	Control Register A
	TCF1_CTRLB	2881	0xB41	Control Register B
	TCF1_CTRLC	2882	0xB42	Control register C
	TCF1_CTRLD	2883	0xB43	Control Register D
	TCF1_CTRLE	2884	0xB44	Control Register E
	TCF1_INTCTRLA	2886	0xB46	Interrupt Control Register A

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
Timer/Counter 1 on Port F	TCF1_INTCTRLB	2887	0xB47	Interrupt Control Register B
	TCF1_CTRLFCLR	2888	0xB48	Control Register F Clear
	TCF1_CTRLFSET	2889	0xB49	Control Register F Set
	TCF1_CTRLGCLR	2890	0xB4A	Control Register G Clear
	TCF1_CTRLGSET	2891	0xB4B	Control Register G Set
	TCF1_INTFLAGS	2892	0xB4C	Interrupt Flag Register
	TCF1_TEMP	2895	0xB4F	Temporary Register For 16-bit Access
	TCF1_CNT	2912	0xB60	Count
	TCF1_PER	2918	0xB66	Period
	TCF1_CCA	2920	0xB68	Compare or Capture A
	TCF1_CCB	2922	0xB6A	Compare or Capture B
	TCF1_PERBUF	2934	0xB76	Period Buffer
	TCF1_CCABUF	2936	0xB78	Compare Or Capture A Buffer
	TCF1_CCBBUF	2938	0xB7A	Compare Or Capture B Buffer
	Interrupt Vectors			
	TCF1_CCA_vect	232	0xE8	Compare or Capture A Interrupt
	TCF1_CCB_vect	234	0xEA	Compare or Capture B Interrupt
	TCF1_ERR_vect	230	0xE6	Error Interrupt
	TCF1_OVF_vect	228	0xE4	Overflow Interrupt
Timer/Counter 2 on Port F	**Registers**			
	TCF2_CTRLA	2816	0xB00	Control Register A
	TCF2_CTRLB	2817	0xB01	Control Register B
	TCF2_CTRLC	2818	0xB02	Control register C
	TCF2_CTRLE	2820	0xB04	Control Register E
	TCF2_INTCTRLA	2822	0xB06	Interrupt Control Register A
	TCF2_INTCTRLB	2823	0xB07	Interrupt Control Register B
	TCF2_CTRLF	2825	0xB09	Control Register F
	TCF2_INTFLAGS	2828	0xB0C	Interrupt Flag Register
	TCF2_LCNT	2848	0xB20	Low Byte Count
	TCF2_HCNT	2849	0xB21	High Byte Count
	TCF2_LPER	2854	0xB26	Low Byte Period
	TCF2_HPER	2855	0xB27	High Byte Period
	TCF2_LCMPA	2856	0xB28	Low Byte Compare A
	TCF2_HCMPA	2857	0xB29	High Byte Compare A
	TCF2_LCMPB	2858	0xB2A	Low Byte Compare B
	TCF2_HCMPB	2859	0xB2B	High Byte Compare B
	TCF2_LCMPC	2860	0xB2C	Low Byte Compare C
	TCF2_HCMPC	2861	0xB2D	High Byte Compare C
	TCF2_LCMPD	2862	0xB2E	Low Byte Compare D
	TCF2_HCMPD	2863	0xB2F	High Byte Compare D
	Interrupt Vectors			
	TCF2_HUNF_vect	218	0xDA	High Byte Underflow Interrupt
	TCF2_LCMPA_vect	220	0xDC	Low Byte Compare A Interrupt

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
	TCF2_LCMPB_vect	222	0xDE	Low Byte Compare B Interrupt
	TCF2_LCMPC_vect	224	0xE0	Low Byte Compare C Interrupt
	TCF2_LCMPD_vect	226	0xE2	Low Byte Compare D Interrupt
	TCF2_LUNF_vect	216	0xD8	Low Byte Underflow Interrupt
Two Wire Interfaceon Port C	**Registers**			
	TWIC_CTRL	1152	0x480	TWI Common Control Register
	TWIC_MASTER_CTRLA	1153	0x481	Control Register A
	TWIC_MASTER_CTRLB	1154	0x482	Control Register B
	TWIC_MASTER_CTRLC	1155	0x483	Control Register C
	TWIC_MASTER_STATUS	1156	0x484	Status Register
	TWIC_MASTER_BAUD	1157	0x485	Baurd Rate Control Register
	TWIC_MASTER_ADDR	1158	0x486	Address Register
	TWIC_MASTER_DATA	1159	0x487	Data Register
	TWIC_SLAVE_CTRLA	1160	0x488	Control Register A
	TWIC_SLAVE_CTRLB	1161	0x489	Control Register B
	TWIC_SLAVE_STATUS	1162	0x48A	Status Register
	TWIC_SLAVE_ADDR	1163	0x48B	Address Register
	TWIC_SLAVE_DATA	1164	0x48C	Data Register
	TWIC_SLAVE_ADDRMASK	1165	0x48D	Address Mask Register
	Interrupt Vectors			
	TWIC_TWIM_vect	26	0x1A	TWI Master Interrupt
	TWIC_TWIS_vect	24	0x18	TWI Slave Interrupt
Two Wire Interfaceon Port D	**Registers**			
	TWID_CTRL	1168	0x490	TWI Common Control Register
	TWID_MASTER_CTRLA	1169	0x491	Control Register A
	TWID_MASTER_CTRLB	1170	0x492	Control Register B
	TWID_MASTER_CTRLC	1171	0x493	Control Register C
	TWID_MASTER_STATUS	1172	0x494	Status Register
	TWID_MASTER_BAUD	1173	0x495	Baurd Rate Control Register
	TWID_MASTER_ADDR	1174	0x496	Address Register
	TWID_MASTER_DATA	1175	0x497	Data Register
	TWID_SLAVE_CTRLA	1176	0x498	Control Register A
	TWID_SLAVE_CTRLB	1177	0x499	Control Register B
	TWID_SLAVE_STATUS	1178	0x49A	Status Register
	TWID_SLAVE_ADDR	1179	0x49B	Address Register
	TWID_SLAVE_DATA	1180	0x49C	Data Register
	TWID_SLAVE_ADDRMASK	1181	0x49D	Address Mask Register
	Interrupt Vectors			
	TWID_TWIM_vect	152	0x98	TWI Master Interrupt
	TWID_TWIS_vect	150	0x96	TWI Slave Interrupt
	Registers			
	TWIE_CTRL	1184	0x4A0	TWI Common Control Register
	TWIE_MASTER_CTRLA	1185	0x4A1	Control Register A

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
Two Wire Interfaceon Port E	TWIE_MASTER_CTRLB	1186	0x4A2	Control Register B
	TWIE_MASTER_CTRLC	1187	0x4A3	Control Register C
	TWIE_MASTER_STATUS	1188	0x4A4	Status Register
	TWIE_MASTER_BAUD	1189	0x4A5	Baurd Rate Control Register
	TWIE_MASTER_ADDR	1190	0x4A6	Address Register
	TWIE_MASTER_DATA	1191	0x4A7	Data Register
	TWIE_SLAVE_CTRLA	1192	0x4A8	Control Register A
	TWIE_SLAVE_CTRLB	1193	0x4A9	Control Register B
	TWIE_SLAVE_STATUS	1194	0x4AA	Status Register
	TWIE_SLAVE_ADDR	1195	0x4AB	Address Register
	TWIE_SLAVE_DATA	1196	0x4AC	Data Register
	TWIE_SLAVE_ADDRMASK	1197	0x4AD	Address Mask Register
	Interrupt Vectors			
	TWIE_TWIM_vect	92	0x5C	TWI Master Interrupt
	TWIE_TWIS_vect	90	0x5A	TWI Slave Interrupt
Two Wire Interfaceon Port F	**Registers**			
	TWIF_CTRL	1200	0x4B0	TWI Common Control Register
	TWIF_MASTER_CTRLA	1201	0x4B1	Control Register A
	TWIF_MASTER_CTRLB	1202	0x4B2	Control Register B
	TWIF_MASTER_CTRLC	1203	0x4B3	Control Register C
	TWIF_MASTER_STATUS	1204	0x4B4	Status Register
	TWIF_MASTER_BAUD	1205	0x4B5	Baurd Rate Control Register
	TWIF_MASTER_ADDR	1206	0x4B6	Address Register
	TWIF_MASTER_DATA	1207	0x4B7	Data Register
	TWIF_SLAVE_CTRLA	1208	0x4B8	Control Register A
	TWIF_SLAVE_CTRLB	1209	0x4B9	Control Register B
	TWIF_SLAVE_STATUS	1210	0x4BA	Status Register
	TWIF_SLAVE_ADDR	1211	0x4BB	Address Register
	TWIF_SLAVE_DATA	1212	0x4BC	Data Register
	TWIF_SLAVE_ADDRMASK	1213	0x4BD	Address Mask Register
	Interrupt Vectors			
	TWIF_TWIM_vect	214	0xD6	TWI Master Interrupt
	TWIF_TWIS_vect	212	0xD4	TWI Slave Interrupt
USART0 on Port C	**Registers**			
	USARTC0_DATA	2208	0x8A0	Data Register
	USARTC0_STATUS	2209	0x8A1	Status Register
	USARTC0_CTRLA	2211	0x8A3	Control Register A
	USARTC0_CTRLB	2212	0x8A4	Control Register B
	USARTC0_CTRLC	2213	0x8A5	Control Register C
	USARTC0_BAUDCTRLA	2214	0x8A6	Baud Rate Control Register A
	USARTC0_BAUDCTRLB	2215	0x8A7	Baud Rate Control Register B
	Interrupt Vectors			
	USARTC0_DRE_vect	52	0x34	Data Register Empty Interrupt

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
	USARTC0_RXC_vect	50	0x32	Reception Complete Interrupt
	USARTC0_TXC_vect	54	0x36	Transmission Complete Interrupt
USART1 on Port C	**Registers**			
	USARTC1_DATA	2224	0x8B0	Data Register
	USARTC1_STATUS	2225	0x8B1	Status Register
	USARTC1_CTRLA	2227	0x8B3	Control Register A
	USARTC1_CTRLB	2228	0x8B4	Control Register B
	USARTC1_CTRLC	2229	0x8B5	Control Register C
	USARTC1_BAUDCTRLA	2230	0x8B6	Baud Rate Control Register A
	USARTC1_BAUDCTRLB	2231	0x8B7	Baud Rate Control Register B
	Interrupt Vectors			
	USARTC1_DRE_vect	58	0x3A	Data Register Empty Interrupt
	USARTC1_RXC_vect	56	0x38	Reception Complete Interrupt
	USARTC1_TXC_vect	60	0x3C	Transmission Complete Interrupt
USART0 on Port D	**Registers**			
	USARTD0_DATA	2464	0x9A0	Data Register
	USARTD0_STATUS	2465	0x9A1	Status Register
	USARTD0_CTRLA	2467	0x9A3	Control Register A
	USARTD0_CTRLB	2468	0x9A4	Control Register B
	USARTD0_CTRLC	2469	0x9A5	Control Register C
	USARTD0_BAUDCTRLA	2470	0x9A6	Baud Rate Control Register A
	USARTD0_BAUDCTRLB	2471	0x9A7	Baud Rate Control Register B
	Interrupt Vectors			
	USARTD0_DRE_vect	178	0xB2	Data Register Empty Interrupt
	USARTD0_RXC_vect	176	0xB0	Reception Complete Interrupt
	USARTD0_TXC_vect	180	0xB4	Transmission Complete Interrupt
USART1 on Port D	**Registers**			
	USARTD1_DATA	2480	0x9B0	Data Register
	USARTD1_STATUS	2481	0x9B1	Status Register
	USARTD1_CTRLA	2483	0x9B3	Control Register A
	USARTD1_CTRLB	2484	0x9B4	Control Register B
	USARTD1_CTRLC	2485	0x9B5	Control Register C
	USARTD1_BAUDCTRLA	2486	0x9B6	Baud Rate Control Register A
	USARTD1_BAUDCTRLB	2487	0x9B7	Baud Rate Control Register B
	Interrupt Vectors			
	USARTD1_DRE_vect	184	0xB8	Data Register Empty Interrupt
	USARTD1_RXC_vect	182	0xB6	Reception Complete Interrupt
	USARTD1_TXC_vect	186	0xBA	Transmission Complete Interrupt
E	**Registers**			
	USARTE0_DATA	2720	0xAA0	Data Register
	USARTE0_STATUS	2721	0xAA1	Status Register
	USARTE0_CTRLA	2723	0xAA3	Control Register A

Thanks to Mason Turner!

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
USART0 on Port	USARTE0_CTRLB	2724	0xAA4	Control Register B
	USARTE0_CTRLC	2725	0xAA5	Control Register C
	USARTE0_BAUDCTRLA	2726	0xAA6	Baud Rate Control Register A
	USARTE0_BAUDCTRLB	2727	0xAA7	Baud Rate Control Register B
	Interrupt Vectors			
	USARTE0_DRE_vect	118	0x76	Data Register Empty Interrupt
	USARTE0_RXC_vect	116	0x74	Reception Complete Interrupt
	USARTE0_TXC_vect	120	0x78	Transmission Complete Interrupt
USART1 on Port E	**Registers**			
	USARTE1_DATA	2736	0xAB0	Data Register
	USARTE1_STATUS	2737	0xAB1	Status Register
	USARTE1_CTRLA	2739	0xAB3	Control Register A
	USARTE1_CTRLB	2740	0xAB4	Control Register B
	USARTE1_CTRLC	2741	0xAB5	Control Register C
	USARTE1_BAUDCTRLA	2742	0xAB6	Baud Rate Control Register A
	USARTE1_BAUDCTRLB	2743	0xAB7	Baud Rate Control Register B
	Interrupt Vectors			
	USARTE1_DRE_vect	124	0x7C	Data Register Empty Interrupt
	USARTE1_RXC_vect	122	0x7A	Reception Complete Interrupt
	USARTE1_TXC_vect	126	0x7E	Transmission Complete Interrupt
USART0 on Port F	**Registers**			
	USARTF0_DATA	2976	0xBA0	Data Register
	USARTF0_STATUS	2977	0xBA1	Status Register
	USARTF0_CTRLA	2979	0xBA3	Control Register A
	USARTF0_CTRLB	2980	0xBA4	Control Register B
	USARTF0_CTRLC	2981	0xBA5	Control Register C
	USARTF0_BAUDCTRLA	2982	0xBA6	Baud Rate Control Register A
	USARTF0_BAUDCTRLB	2983	0xBA7	Baud Rate Control Register B
	Interrupt Vectors			
	USARTF0_DRE_vect	240	0xF0	Data Register Empty Interrupt
	USARTF0_RXC_vect	238	0xEE	Reception Complete Interrupt
	USARTF0_TXC_vect	242	0xF2	Transmission Complete Interrupt
USART1 on Port F	**Registers**			
	USARTF1_DATA	2992	0xBB0	Data Register
	USARTF1_STATUS	2993	0xBB1	Status Register
	USARTF1_CTRLA	2995	0xBB3	Control Register A
	USARTF1_CTRLB	2996	0xBB4	Control Register B
	USARTF1_CTRLC	2997	0xBB5	Control Register C
	USARTF1_BAUDCTRLA	2998	0xBB6	Baud Rate Control Register A
	USARTF1_BAUDCTRLB	2999	0xBB7	Baud Rate Control Register B
	Interrupt Vectors			
	USARTF1_DRE_vect	246	0xF6	Data Register Empty Interrupt
	USARTF1_RXC_vect	244	0xF4	Reception Complete Interrupt

Thanks to Mason Turner!

Device	Name	$Addr_{10}$	$Addr_{16}$	Description
	USARTF1_TXC_vect	248	0xF8	Transmission Complete Interrupt
Universal Serial Bus (USB)	**Registers**			
	USB_CTRLA	1216	0x4C0	Control Register A
	USB_CTRLB	1217	0x4C1	Control Register B
	USB_STATUS	1218	0x4C2	Status Register
	USB_ADDR	1219	0x4C3	Address Register
	USB_FIFOWP	1220	0x4C4	FIFO Write Pointer Register
	USB_FIFORP	1221	0x4C5	FIFO Read Pointer Register
	USB_EPPTR	1222	0x4C6	Endpoint Configuration Table Pointer
	USB_INTCTRLA	1224	0x4C8	Interrupt Control Register A
	USB_INTCTRLB	1225	0x4C9	Interrupt Control Register B
	USB_INTFLAGSACLR	1226	0x4CA	Clear Interrupt Flag Register A
	USB_INTFLAGSASET	1227	0x4CB	Set Interrupt Flag Register A
	USB_INTFLAGSBCLR	1228	0x4CC	Clear Interrupt Flag Register B
	USB_INTFLAGSBSET	1229	0x4CD	Set Interrupt Flag Register B
	USB_CAL0	1274	0x4FA	Calibration Byte 0
	USB_CAL1	1275	0x4FB	Calibration Byte 1
	Interrupt Vectors			
	USB_BUSEVENT_vect	250	0xFA	SOF
	USB_TRNCOMPL_vect	252	0xFC	Transaction complete interrupt
Watchdog Timer	**Registers**			
	WDT_CTRL	128	0x80	Control
	WDT_WINCTRL	129	0x81	Windowed Mode Control
	WDT_STATUS	130	0x82	Status

Thanks to Mason Turner!