ATtiny828

8-bit AVR Microcontroller with 8K Bytes In-System Programmable Flash

DATASHEET

Features

- High Performance, Low Power Atmel® AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 123 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
- Non-volatile Program and Data Memories
  - 8K Bytes of In-System Programmable Flash Program Memory
    - Endurance: 10,000 Write/Erase Cycles
  - 256 Bytes of In-System Programmable EEPROM
    - Endurance: 100,000 Write/Erase Cycles
  - 512 Bytes Internal SRAM
  - Optional Boot Code Section with Independent Lock Bits
  - Data Retention: 20 Years at 85°C / 100 Years at 25°C
- Peripheral Features
  - One 8-bit and one 16-bit Timer/Counter with Two PWM Channels, Each
  - Programmable Ultra Low Power Watchdog Timer
  - On-chip Analog Comparator
  - 10-bit Analog to Digital Converter
    - 28 External and 4 Internal, Single-ended Input Channels
  - Full Duplex USART with Start Frame Detection
  - Master/Slave SPI Serial Interface
  - Slave I^2C Serial Interface
- Special Microcontroller Features
  - Low Power Idle, ADC Noise Reduction, and Power-down Modes
  - Enhanced Power-on Reset Circuit
  - Programmable Brown-out Detection Circuit with Supply Voltage Sampling
  - External and Internal Interrupt Sources
    - Pin Change Interrupt on 28 Pins
  - Calibrated 8MHz Oscillator with Temperature Calibration Option
  - Calibrated 32kHz Ultra Low Power Oscillator
  - High-Current Drive Capability on 8 I/O Pins
- I/O and Packages
  - 32-lead TQFP, and 32-pad QFN/MLF: 28 Programmable I/O Lines
- Speed Grade
  - 0 – 2 MHz @ 1.7 – 1.8V
  - 0 – 4 MHz @ 1.8 – 5.5V
  - 0 – 10 MHz @ 2.7 – 5.5V
  - 0 – 20 MHz @ 4.5 – 5.5V

- Low Power Consumption
  - Active Mode: 0.2 mA at 1.8V and 1MHz
  - Idle Mode: 30 µA at 1.8V and 1MHz
  - Power-Down Mode (WDT Enabled): 1 µA at 1.8V
  - Power-Down Mode (WDT Disabled): 100 nA at 1.8V

1. Pin Configurations

Figure 1. ATtiny828 Pinout in MLF32.

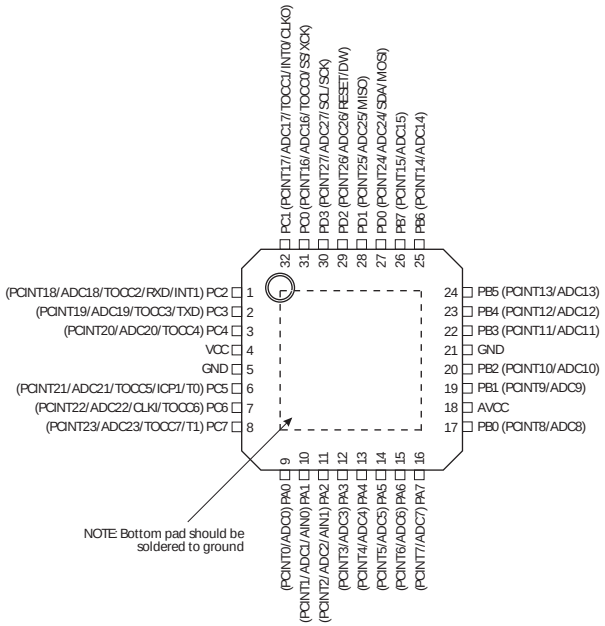
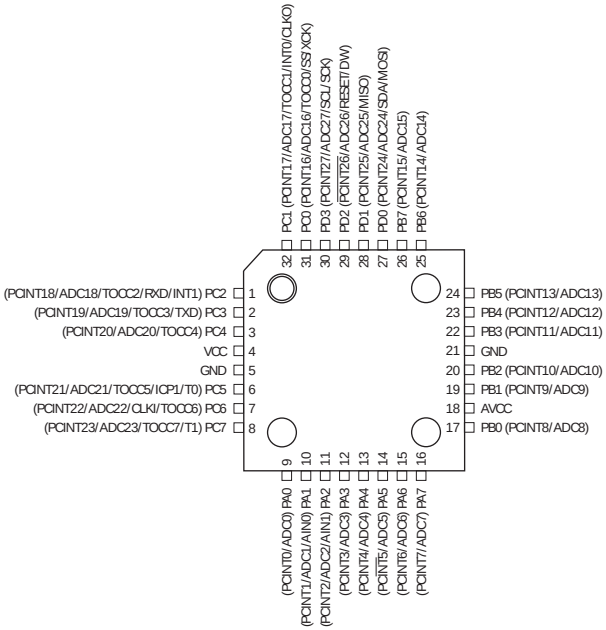


Figure 2. ATtiny828 Pinout in TQFP32.



1.1 Pin Description

1.1.1 VCC

Supply voltage.

1.1.2 AVCC

AV_{CC} is the supply voltage pin for the A/D converter and a selection of I/O pins. This pin should be externally connected to V_{CC} even if the ADC is not used. If the ADC is used, it is recommended this pin is connected to V_{CC} through a low-pass filter, as described in “Noise Canceling Techniques” on page 145.

All pins of Port A and Port B are powered by AV_{CC}. All other I/O pins take their supply voltage from V_{CC}.

1.1.3 GND

Ground.

1.1.4 $\overline{\text{RESET}}$

Reset input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running and provided the reset pin has not been disabled. The minimum pulse length is given in Table 107 on page 250. Shorter pulses are not guaranteed to generate a reset.

The reset pin can also be used as a (weak) I/O pin.

1.1.5 Port A (PA7:PA0)

This is an 8-bit, bi-directional I/O port with internal pull-up resistors (selected for each bit). Output buffers have high sink and standard source capability. See Table 107 on page 250 for port drive strength.

As inputs, port pins that are externally pulled low will source current provided that pull-up resistors are activated. Port pins are tri-stated when a reset condition becomes active, even if the clock is not running.

This port has alternative pin functions for pin change interrupts, the analog comparator, and ADC. See “Alternative Port Functions” on page 63.

1.1.6 Port B (PB7:PB0)

This is an 8-bit, bi-directional I/O port with internal pull-up resistors (selected for each bit). Output buffers have high sink and standard source capability. See Table 103 on page 247 for port drive strength.

As inputs, port pins that are externally pulled low will source current provided that pull-up resistors are activated. Port pins are tri-stated when a reset condition becomes active, even if the clock is not running.

This port has alternative pin functions for pin change interrupts, and ADC. See “Alternative Port Functions” on page 63.

1.1.7 Port C (PC7:PC0)

This is an 8-bit, bi-directional I/O port with internal pull-up resistors (selected for each bit). Output buffers have high sink and standard source capability. Optionally, extra high sink capability can be enabled. See Table 103 on page 247 for port drive strength.

As inputs, port pins that are externally pulled low will source current provided that pull-up resistors are activated. Port pins are tri-stated when a reset condition becomes active, even if the clock is not running.

This port has alternative pin functions for pin change interrupts, ADC, timer/counter, external interrupts, and serial interfaces. See “Alternative Port Functions” on page 63.

1.1.8 Port D (PD3:PD0)

This is a 4-bit, bi-directional I/O port with internal pull-up resistors (selected for each bit). Output buffers of PD0 and PD3 have symmetrical drive characteristics, with both sink and source capability. Output buffer PD1 has high sink and

standard source capability, while PD2 only has weak drive characteristics due to its use as a reset pin. See Table 103 on page 247 for port drive strength.

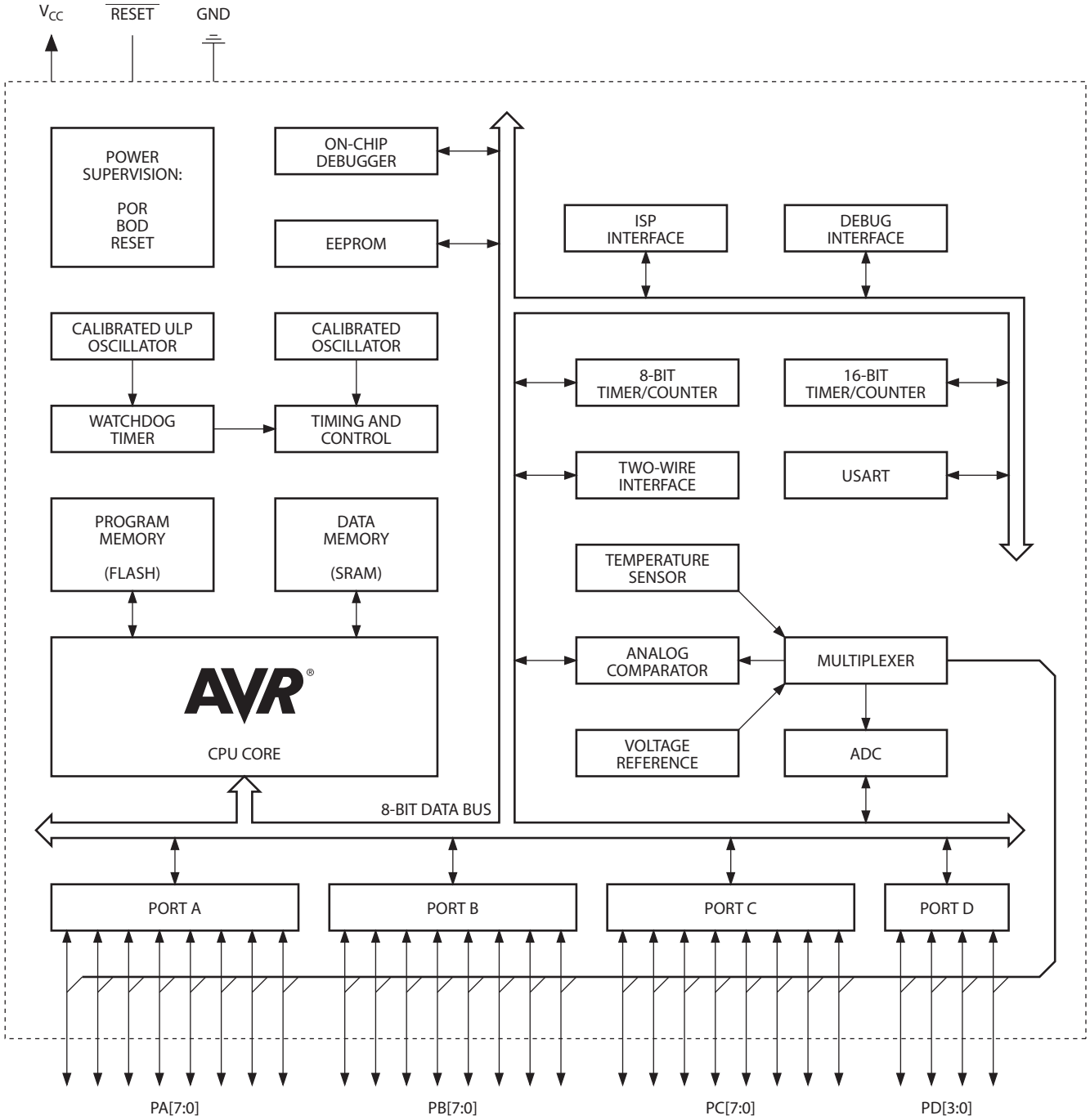
As inputs, port pins that are externally pulled low will source current provided that pull-up resistors are activated. Port pins are tri-stated when a reset condition becomes active, even if the clock is not running.

This port has alternative pin functions for pin change interrupts, ADC, serial interfaces, and debugWire. See “Alternative Port Functions” on page 63.

2. Overview

ATtiny828 is a low-power CMOS 8-bit microcontrollers based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATtiny828 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

Figure 3. Block Diagram



The AVR core combines a rich instruction set with 32 general purpose working registers. All 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in a single instruction, executed in one clock cycle. The resulting architecture is compact and code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

ATtiny828 provides the following features:

- 8K bytes of in-system programmable Flash
- 512 bytes of SRAM data memory
- 256 bytes of EEPROM data memory
- 28 general purpose I/O lines
- 32 general purpose working registers
- An 8-bit timer/counter with two PWM channels
- A16-bit timer/counter with two PWM channels
- Internal and external interrupts
- A 10-bit ADC with 4 internal and 28 external chanels
- An ultra-low power, programmable watchdog timer with internal oscillator
- A programmable USART with start frame detection
- A slave, I^2C compliant Two-Wire Interface (TWI)
- A master/slave Serial Peripheral Interface (SPI)
- A calibrated 8MHz oscillator
- A calibrated 32kHz, ultra low power oscillator
- Three software selectable power saving modes.

The device includes the following modes for saving power:

- Idle mode: stops the CPU while allowing the timer/counter, ADC, analog comparator, SPI, TWI, and interrupt system to continue functioning
- ADC Noise Reduction mode: minimizes switching noise during ADC conversions by stopping the CPU and all I/O modules except the ADC
- Power-down mode: registers keep their contents and all chip functions are disabled until the next interrupt or hardware reset

The device is manufactured using Atmel's high density non-volatile memory technology. The Flash program memory can be re-programmed in-system through a serial interface, by a conventional non-volatile memory programmer or by an on-chip boot code, running on the AVR core. The boot program can use any interface to download the application program to the Flash memory. Software in the boot section of the Flash executes while the application section of the Flash is updated, providing true read-while-write operation.

The ATtiny828 AVR is supported by a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators and evaluation kits.

3. General Information

3.1 Resources

A comprehensive set of drivers, application notes, data sheets and descriptions on development tools are available for download at http://www.atmel.com/avr.

3.2 Code Examples

This documentation contains simple code examples that briefly show how to use various parts of the device. These code examples assume that the part specific header file is included before compilation. Be aware that not all C compiler vendors include bit definitions in the header files and interrupt handling in C is compiler dependent. Please confirm with the C compiler documentation for more details.

3.3 Data Retention

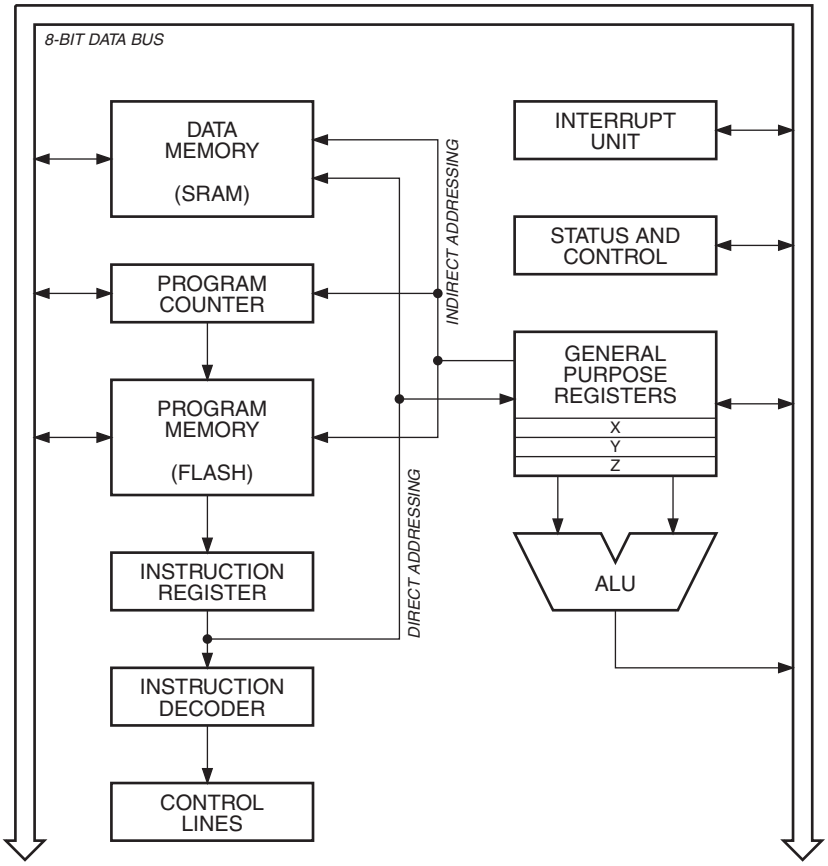
Reliability Qualification results show that the projected data retention failure rate is much less than 1 PPM over 20 years at 85°C or 100 years at 25°C.

4. CPU Core

This section discusses the AVR core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts.

4.1 Architectural Overview

Figure 4. Block Diagram of the AVR Architecture



In order to maximize performance and parallelism, the AVR uses a Harvard architecture – with separate memories and buses for program and data. Instructions in the Program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the Program memory. This concept enables instructions to be executed in every clock cycle. The Program memory is In-System Reprogrammable Flash memory.

The fast-access Register File contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two operands are output from the Register File, the operation is executed, and the result is stored back in the Register File – in one clock cycle.

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

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

Program flow is provided by conditional and unconditional jump and call instructions, capable of directly addressing the whole address space. Most AVR instructions have a single 16-bit word format but 32-bit wide instructions also exist. The actual instruction set varies, as some devices only implement a part of the instruction set.

Program Flash memory is divided in two sections; the boot program section and the application program section. Both sections have dedicated lock bits for write and read/write protection. The SPM instruction, which is used to write the application memory section, must reside in the boot program section.

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

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

A flexible interrupt module has its control registers in the I/O space with an additional Global Interrupt Enable bit in the Status Register. All interrupts have a separate Interrupt Vector in the Interrupt Vector table. The interrupts have priority in accordance with their Interrupt Vector position. The lower the Interrupt Vector address, the higher the priority.

The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers, SPI, and other I/O functions. The I/O memory can be accessed directly, or as the Data Space locations following those of the Register File, 0x20 - 0x5F. In addition, the ATtiny828 has Extended I/O Space from 0x60 - 0xFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

4.2 ALU – Arithmetic Logic Unit

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format. See external document “AVR Instruction Set” and “Instruction Set Summary” on page 301 section for more information.

4.3 Status Register

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the Status Register is updated after all ALU operations. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code. See external document “AVR Instruction Set” and “Instruction Set Summary” on page 301 section for more information.

The Status Register is neither automatically stored when entering an interrupt routine, nor restored when returning from an interrupt. This must be handled by software.

4.4 General Purpose Register File

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

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

Figure 5 below shows the structure of the 32 general purpose working registers in the CPU.

Figure 5. General Purpose Working Registers

7 … 0		
R0	0x00	
R1	0x01	
R2	0x02	
R3	0x03	
…	...	
R12	0x0C	
R13	0x0D	
R14	0x0E	
R15	0x0F	
R16	0x10	
R17	0x11	
…	...	
R26	0x1A	X-register Low Byte
R27	0x1B	X-register High Byte
R28	0x1C	Y-register Low Byte
R29	0x1D	Y-register High Byte
R30	0x1E	Z-register Low Byte
R31	0x1F	Z-register High Byte

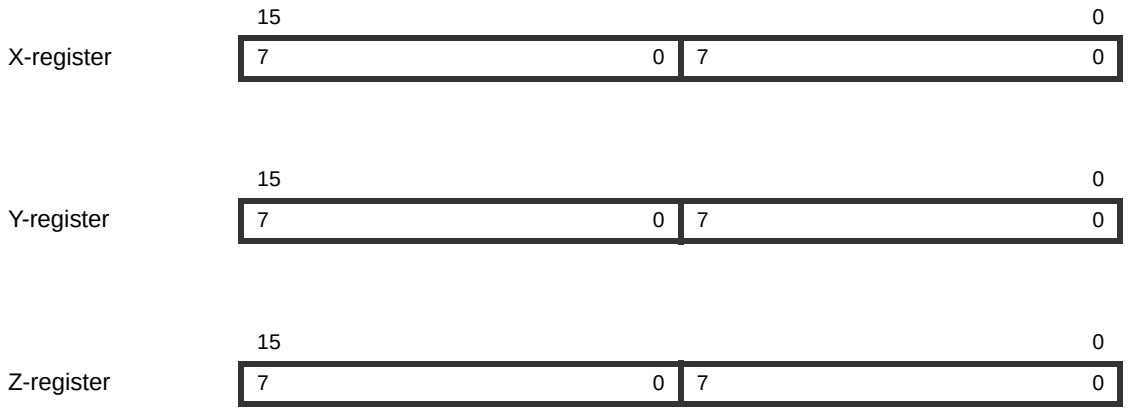
Most of the instructions operating on the Register File are single cycle instructions with direct access to all registers.

As shown in Figure 5, each register is also assigned a Data memory address, mapping them directly into the first 32 locations of the user Data Space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y- and Z-pointer registers can be set to index any register in the file.

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

The registers R26..R31 have added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the data space. The three indirect address registers X, Y, and Z are defined as described in Figure 6 below.

Figure 6. The X-, Y-, and Z-registers



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

4.5 Stack Pointer

The stack is mainly used for storing temporary data, local variables and return addresses after interrupts and subroutine calls. The Stack Pointer registers (SPH and SPL) always point to the top of the stack. Note that the stack grows from higher memory locations to lower memory locations. This means that the PUSH instructions decreases and the POP instruction increases the stack pointer value.

The stack pointer points to the area of data memory where subroutine and interrupt stacks are located. This stack space must be defined by the program before any subroutine calls are executed or interrupts are enabled.

The pointer is decremented by one when data is put on the stack with the PUSH instruction, and incremented by one when data is fetched with the POP instruction. It is decremented by two when the return address is put on the stack by a subroutine call or a jump to an interrupt service routine, and incremented by two when data is fetched by a return from subroutine (the RET instruction) or a return from interrupt service routine (the RETI instruction).

The AVR stack pointer is typically implemented as two 8-bit registers in the I/O register file. The width of the stack pointer and the number of bits implemented is device dependent. In some AVR devices all data memory can be addressed using SPL, only. In this case, the SPH register is not implemented.

The stack pointer must be set to point above the I/O register areas, the minimum value being the lowest address of SRAM. See Table 3 on page 17.

4.6 Instruction Execution Timing

This section describes the general access timing concepts for instruction execution. The AVR CPU is driven by the CPU clock clk_{CPU}, directly generated from the selected clock source for the chip. No internal clock division is used.

Figure 7 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 to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

Figure 7. The Parallel Instruction Fetches and Instruction Executions

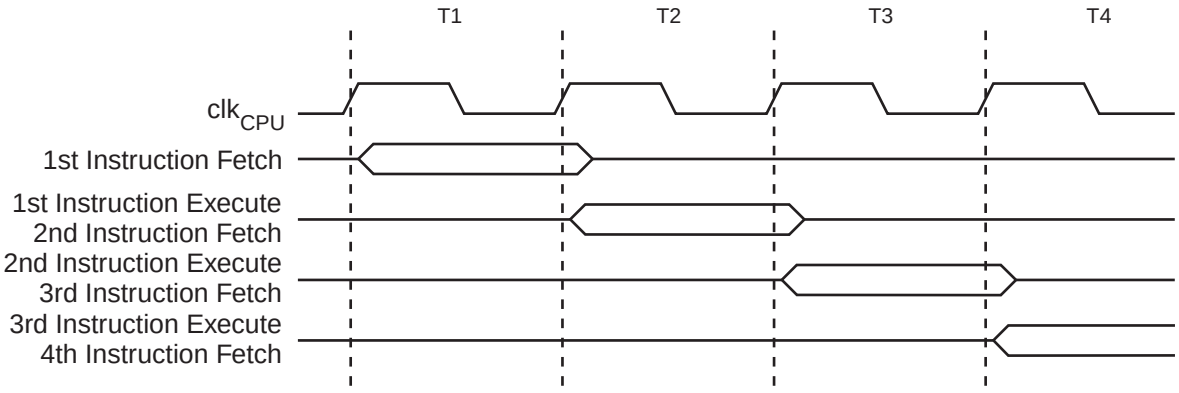
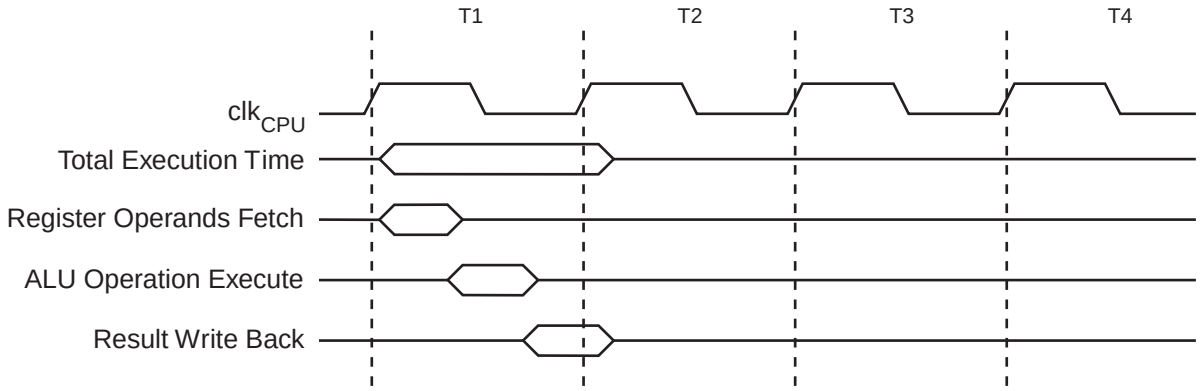


Figure 8 shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

Figure 8. Single Cycle ALU Operation



4.7 Reset and Interrupt Handling

The AVR provides several different interrupt sources. These interrupts and the separate Reset Vector each have a separate Program Vector in the Program memory space. All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt.

Depending on the value of the program counter, interrupts may be automatically disabled when Boot Lock Bits (BLB02 or BLB12) are programmed. This feature improves software security. See section “Lock Bits” on page 225 for details.

The lowest addresses in the Program memory space are by default defined as the Reset and Interrupt Vectors. The complete list of vectors is shown in “Interrupts” on page 48. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 – the External Interrupt Request 0.

The interrupt vector table can be moved to the start of Flash boot section by setting the IVSEL bit. For more information, see “MCUCR – MCU Control Register” on page 53 and “Interrupts” on page 48. The reset vector can also be moved to the start of Flash boot section by programming the BOOTRST fuse. See “Entering the Boot Loader Program” on page 216.

When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed.

There are basically two types of interrupts. The first type is triggered by an event that sets the Interrupt Flag. For these interrupts, the Program Counter is vectored to the actual Interrupt Vector in order to execute the interrupt handling routine, and hardware clears the corresponding Interrupt Flag. Interrupt Flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, 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 the Global Interrupt Enable bit is cleared, the corresponding Interrupt Flag(s) will be set and remembered until the Global Interrupt Enable bit is set, and will then be executed by order of priority.

The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have Interrupt Flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered.

When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.

Note that the Status Register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction. The following example shows how this can be used to avoid interrupts during the timed EEPROM write sequence.

Assembly Code Example

```
in    r16,  SREG              ; store SREG value
cli                           ; disable interrupts during timed sequence
sbi   EECR, EEMPE             ; start EEPROM write
sbi   EECR, EEPE
out   SREG, r16               ; restore SREG value (I-bit)
```

C Code Example

```
char  cSREG;

cSREG = SREG;           /* store SREG value */
_CLI();                 /* disable interrupts during timed sequence */
EECR |= (1<<EEMPE);     /* start EEPROM write */
EECR |= (1<<EEPE);
SREG = cSREG;           /* restore SREG value (I-bit) */
```

Note: See “Code Examples” on page 7.

When using the SEI instruction to enable interrupts, the instruction following SEI will be executed before any pending interrupts, as shown in the following example.

Assembly Code Example

```
sei               ; set Global Interrupt Enable
sleep             ; enter sleep, waiting for interrupt
                  ; note: will enter sleep before any pending interrupt(s)
```

C Code Example

```
_SEI();           /* set Global Interrupt Enable */
_SLEEP();         /* enter sleep, waiting for interrupt */
                  /* note: will enter sleep before any pending interrupt */
```

Note: See “Code Examples” on page 7.

4.7.1 Interrupt Response Time

The interrupt execution response for all the enabled AVR interrupts is four clock cycles minimum. After four clock cycles the Program Vector address for the actual interrupt handling routine is executed. During this four clock cycle period, the Program Counter is pushed onto the Stack. The vector is normally a jump to the interrupt routine, and this jump takes three clock cycles. If an interrupt occurs during execution of a multi-cycle instruction, this instruction is completed before the interrupt is served. If an interrupt occurs when the MCU is in sleep mode, the interrupt execution response time is increased by four clock cycles. This increase comes in addition to the start-up time from the selected sleep mode.

A return from an interrupt handling routine takes four clock cycles. During these four clock cycles, the Program Counter (two bytes) is popped back from the Stack, the Stack Pointer is incremented by two, and the I-bit in SREG is set.

4.8 Register Description

4.8.1 CCP – Configuration Change Protection Register

Bit	7	6	5	4	3	2	1	0	
0x36 (0x56)	CCP[7:0]								CCP
Read/Write	W	W	W	W	W	W	W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – CCP[7:0]: Configuration Change Protection**

In order to change the contents of a protected I/O register the CCP register must first be written with the correct signature. After CCP is written the protected I/O registers may be written to during the next four CPU instruction cycles. All interrupts are ignored during these cycles. After these cycles interrupts are automatically handled again by the CPU, and any pending interrupts will be executed according to their priority.

When the protected I/O register signature is written, CCP0 will read as one as long as the protected feature is enabled, while CCP[7:1] will always read as zero.

Table 1 shows the signatures that are recognised.

Table 1. Signatures Recognised by the Configuration Change Protection Register

Signature	Registers	Description
0xD8	CLKPR, MCUCR, WDTCSR(1)	Protected I/O register

Notes: 1. Only WDE and WDP[3:0] bits are protected in WDTCSR.

4.8.2 SPH and SPL — Stack Pointer Registers

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

- **Bits 9:0 – SP[9:0]: Stack Pointer**

The Stack Pointer register points to the top of the stack, which is implemented growing from higher memory locations to lower memory locations. Hence, a stack PUSH command decreases the Stack Pointer.

The stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled.

4.8.3 SREG – Status Register

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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

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

- **Bit 6 – T: Bit Copy Storage**

The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) 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 T by the BST instruction, and a bit in T 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. See the “Instruction Set Description” for detailed information.

- **Bit 4 – S: Sign Bit, S = N ⊕ V**

The S-bit is always an exclusive or between the Negative Flag N and the Two’s Complement Overflow Flag V. See the “Instruction Set Description” for detailed information.

- **Bit 3 – V: Two’s Complement Overflow Flag**

The Two’s Complement Overflow Flag V supports two’s complement arithmetics. See the “Instruction Set Description” for detailed information.

- **Bit 2 – N: Negative Flag**

The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

- **Bit 1 – Z: Zero Flag**

The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

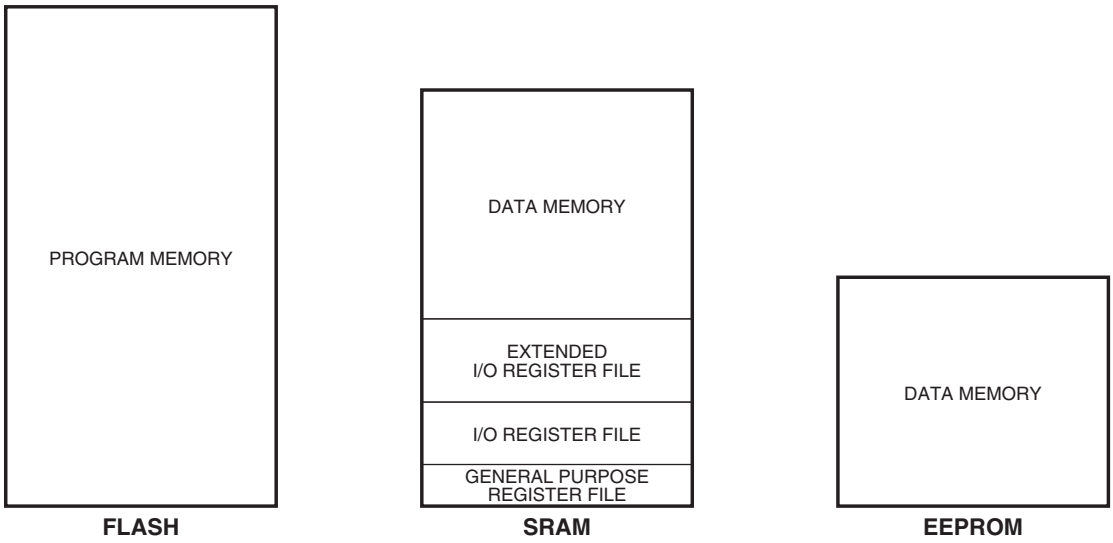
- **Bit 0 – C: Carry Flag**

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

5. Memories

The AVR architecture makes a distinction between program memory and data memory, locating each memory type in a separate address space. Executable code is located in non-volatile program memory (Flash), whereas data can be placed in either volatile (SRAM) or non-volatile memory (EEPROM). See Figure 9, below.

Figure 9. Memory Overview.



All memory spaces are linear and regular.

5.1 Program Memory (Flash)

ATtiny828 contains 8K byte of on-chip, in-system reprogrammable Flash memory for program storage. Flash memories are non-volatile, i.e. they retain stored information even when not powered.

Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 4096 x 16 bits. The Program Counter (PC) is 12 bits wide, thus capable of addressing all 4096 locations of program memory, as illustrated in Table 1, below.

Table 2. Size of Program Memory (Flash)

Device	Flash Size		Address Range
ATtiny828	8KB	4096 words	0x0000 – 0x0FFF

For reasons of software security, the Flash program memory has been divided into two sections; the boot loader section and the application program section. For more information, see “Self-Programming the Flash” on page 218, and “Application and Boot Loader Flash Sections” on page 214.

Constant tables can be allocated within the entire address space of program memory. See instructions LPM (Load Program Memory), and SPM (Store Program Memory) in “Instruction Set Summary” on page 301. Flash program memory can also be programmed from an external device, as described in “External Programming” on page 232.

Timing diagrams for instruction fetch and execution are presented in “Instruction Execution Timing” on page 11.

The Flash memory has a minimum endurance of 10,000 write/erase cycles.

5.2 Data Memory (SRAM) and Register Files

Table 3 shows how the data memory and register files of ATtiny828 are organized. These memory areas are volatile, i.e. they do not retain information when power is removed.

Table 3. Layout of Data Memory and Register Area

Device	Memory Area	Size	Long Address (1)	Short Address (2)
ATtiny828	General purpose register file	32B	0x0000 – 0x001F	n/a
	I/O register file	64B	0x0020 – 0x005F	0x00 – 0x3F
	Extended I/O register file	160B	0x0060 – 0x00FF	n/a
	Data SRAM	512B	0x0100 – 0x02FF	n/a

Note:
1. Also known as data address. This mode of addressing covers the entire data memory and register area. The address is contained in a 16-bit area of two-word instructions.
2. Also known as direct I/O address. This mode of addressing covers part of the register area, only. It is used by instructions where the address is embedded in the instruction word.

The 768 memory locations include the general purpose register file, I/O register file, extended I/O register file, and the internal data memory.

For compatibility with future devices, reserved bits should be written to zero, if accessed. Reserved I/O memory addresses should never be written.

5.2.1 General Purpose Register File

The first 32 locations are reserved for the general purpose register file. These registers are described in detail in “General Purpose Register File” on page 9.

5.2.2 I/O Register File

Following the general purpose register file, the next 64 locations are reserved for I/O registers. Registers in this area are used mainly for communicating with I/O and peripheral units of the device. Data can be transferred between I/O space and the general purpose register file using instructions such as IN, OUT, LD, ST, and derivatives.

All I/O registers in this area can be accessed with the instructions IN and OUT. These I/O specific instructions address the first location in the I/O register area as 0x00 and the last as 0x3F.

The low 32 registers (address range 0x00...0x1F) are accessible by some bit-specific instructions. In these registers, bits are easily set and cleared using SBI and CBI, while bit-conditional branches are readily constructed using instructions SBIC, SBIS, SBRC, and SBRS.

Registers in this area may also be accessed with instructions LD/LDD/LDI/LDS and ST/STD/STS. These instructions treat the entire volatile memory as one data space and, therefore, address I/O registers starting at 0x20.

See “Instruction Set Summary” on page 301.

ATtiny828 also contains three general purpose I/O registers that can be used for storing any information. See GPIOR0, GPIOR1 and GPIOR2 in “Register Summary” on page 297. These general purpose I/O registers are particularly useful for storing global variables and status flags, since they are accessible to bit-specific instructions such as SBI, CBI, SBIC, SBIS, SBRC, and SBRS.

5.2.3 Extended I/O Register File

Following the standard I/O register file, the next 160 locations are reserved for extended I/O registers. ATtiny828 is a complex microcontroller with more peripheral units than can be addressed with the IN and OUT instructions. Registers in the extended I/O area must be accessed using instructions LD/LDD/LDI/LDS and ST/STD/STS. See “Instruction Set Summary” on page 301.

See “Register Summary” on page 297 for a list of I/O registers.

5.2.4 Data Memory (SRAM)

Following the general purpose register file and the I/O register files, the remaining 512 locations are reserved for the internal data SRAM.

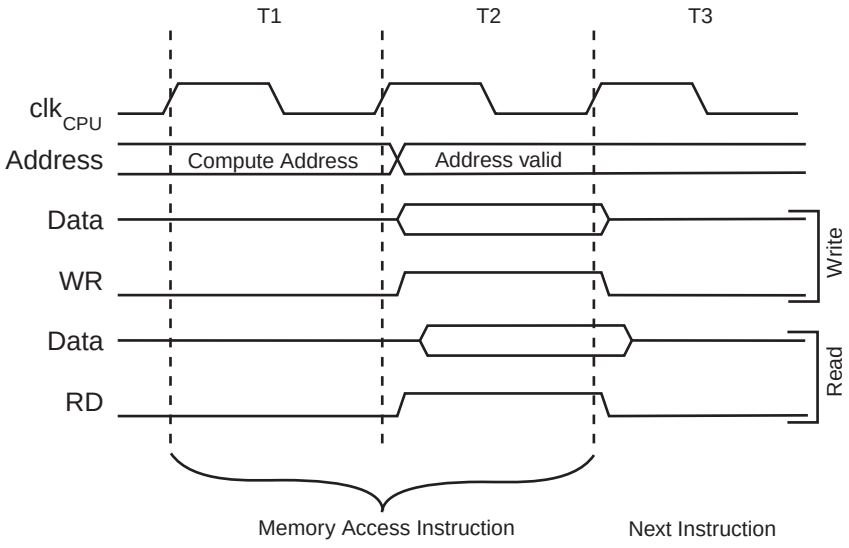
There are five addressing modes available:

- Direct. This mode of addressing reaches the entire data space.
- Indirect.
- Indirect with Displacement. This mode of addressing reaches 63 address locations from the base address given by the Y- or Z-register.
- Indirect with Pre-decrement. In this mode the address register is automatically decremented before access. Address pointer registers (X, Y, and Z) are located in the general purpose register file, in registers R26 to R31. See “General Purpose Register File” on page 9.
- Indirect with Post-increment. In this mode the address register is automatically incremented after access. Address pointer registers (X, Y, and Z) are located in the general purpose register file, in registers R26 to R31. See “General Purpose Register File” on page 9.

All addressing modes can be used on the entire volatile memory, including the general purpose register file, the I/O register files and the data memory.

Internal SRAM is accessed in two clk_{CPU} cycles, as illustrated in Figure 10, below.

Figure 10. On-chip Data SRAM Access Cycles



5.3 Data Memory (EEPROM)

ATtiny828 contains 256 bytes of non-volatile data memory. This EEPROM is organized as a separate data space, in which single bytes can be read and written. All access registers are located in the I/O space.

The EEPROM memory layout is summarised in Table 4, below.

Table 4. Size of Non-Volatile Data Memory (EEPROM)

Device	EEPROM Size	Address Range
ATtiny828	256B	0x00 – 0xFF

The internal 8MHz oscillator is used to time EEPROM operations. The frequency of the oscillator must be within the requirements described in “OSCCAL0 – Oscillator Calibration Register” on page 32.

When powered by heavily filtered supplies, the supply voltage, V_{CC}, is likely to rise or fall slowly on power-up and power-down. Slow rise and fall times may put the device in a state where it is running at supply voltages lower than specified. To avoid problems in situations like this, see “Preventing EEPROM Corruption” on page 20.

The EEPROM has a minimum endurance of 100,000 write/erase cycles.

5.3.1 Programming Methods

There are two methods for EEPROM programming:

- Atomic byte programming. This is the simple mode of programming, where target locations are erased and written in a single operation. In this mode of operation the target is guaranteed to always be erased before writing but programmin times are longer.
- Split byte programming. It is possible to split the erase and write cycle in two different operations. This is useful when short access times are required, for example when supply voltage is falling. In order to take advantage of this method target locations must be erased before writing to them. This can be done at times when the system allows time-critical operations, typically at start-up and initialisation.

The programming method is selected using the EEPROM Programming Mode bits (EEPM1 and EEPM0) in EEPROM Control Register (EECR). See Table 5 on page 24. Write and erase times are given in the same table.

Since EEPROM programming takes some time the application must wait for one operation to complete before starting the next. This can be done by either polling the EEPROM Program Enable bit (EEPE) in EEPROM Control Register (EECR), or via the EEPROM Ready Interrupt. The EEPROM interrupt is controlled by the EEPROM Ready Interrupt Enable (EERIE) bit in EECR.

5.3.2 Read

To read an EEPROM memory location follow the procedure below:

- Poll the EEPROM Program Enable bit (EEPE) in EEPROM Control Register (EECR) to make sure no other EEPROM operations are in process. If set, wait to clear.
- Write target address to EEPROM Address Registers (EEARH/EEARL).
- Start the read operation by setting the EEPROM Read Enable bit (EERE) in the EEPROM Control Register (EECR). During the read operation, the CPU is halted for four clock cycles before executing the next instruction.
- Read data from the EEPROM Data Register (EEDR).

5.3.3 Erase

In order to prevent unintentional EEPROM writes, a specific procedure must be followed to erase memory locations. To erase an EEPROM memory location follow the procedure below:

- Poll the EEPROM Program Enable bit (EEPE) in EEPROM Control Register (EECR) to make sure no other EEPROM operations are in process. If set, wait to clear.
- Poll the SPMEN bit in Store Program Memory Control and Status Register (SPMCSR) to make sure no self-programming opertaions are in process. If set, wait to clear. This step is relevant only if the application contains a boot loader that programs the Flash memory. If not, this step can be omitted.
- Set mode of programming to erase by writing EEPROM Programming Mode bits (EEPM0 and EEPM1) in EEPROM Control Register (EECR).
- Write target address to EEPROM Address Registers (EEARH/EEARL).
- Enable erase by setting EEPROM Master Program Enable (EEMPE) in EEPROM Control Register (EECR). Within four clock cycles, start the erase operation by setting the EEPROM Program Enable bit (EEPE) in the EEPROM Control Register (EECR). During the erase operation, the CPU is halted for two clock cycles before executing the next instruction.

The EEPE bit remains set until the erase operation has completed. While the device is busy programming, it is not possible to perform any other EEPROM operations.

5.3.4 Write

In order to prevent unintentional EEPROM writes, a specific procedure must be followed to write to memory locations.

Before writing data to EEPROM the target location must be erased. This can be done either in the same operation or as part of a split operation. Writing to an unerased EEPROM location will result in corrupted data.

To write an EEPROM memory location follow the procedure below:

- Poll the EEPROM Program Enable bit (EEPE) in EEPROM Control Register (EECR) to make sure no other EEPROM operations are in process. If set, wait to clear.
- Poll the SPMEN bit in Store Program Memory Control and Status Register (SPMCSR) to make sure no self-programming opertaions are in process. If set, wait to clear. This step is relevant only if the application contains a boot loader that programs the Flash memory. If not, this step can be omitted.
- Set mode of programming by writing EEPROM Programming Mode bits (EEPM0 and EEPM1) in EEPROM Control Register (EECR). Alternatively, data can be written in one operation or the write procedure can be split up in erase, only, and write, only.
- Write target address to EEPROM Address Registers (EEARH/EEARL).
- Write target data to EEPROM Data Register (EEDR).
- Enable write by setting EEPROM Master Program Enable (EEMPE) in EEPROM Control Register (EECR). Within four clock cycles, start the write operation by setting the EEPROM Program Enable bit (EEPE) in the EEPROM Control Register (EECR). During the write operation, the CPU is halted for two clock cycles before executing the next instruction.

The EEPE bit remains set until the write operation has completed. While the device is busy with programming, it is not possible to do any other EEPROM operations.

5.3.5 Preventing EEPROM Corruption

During periods of low V_{CC}, the EEPROM data can be corrupted because the supply voltage is too low for the CPU and the EEPROM to operate properly. These issues are the same as for board level systems using EEPROM, and the same design solutions should be applied.

At low supply voltages data in EEPROM can be corrupted in two ways:

- The supply voltage is too low to maintain proper operation of an otherwise legitimate EEPROM program sequence.
- The supply voltage is too low for the CPU and instructions may be executed incorrectly.

EEPROM data corruption is avoided by keeping the device in reset during periods of insufficient power supply voltage. This is easily done by enabling the internal Brown-Out Detector (BOD). If BOD detection levels are not sufficient for the design, an external reset circuit for low V_{CC} can be used.

Provided that supply voltage is sufficient, an EEPROM write operation will be completed even when a reset occurs.

5.3.6 Program Examples

The following code examples show one assembly and one C function for erase, write, or atomic write of the EEPROM. The examples assume that interrupts are controlled (e.g., by disabling interrupts globally) so that no interrupts occur during execution of these functions.

The examples also assume that a boot loader is not used. If a boot loader is present, the EEPROM write function must be expanded to wait for any ongoing SPM operations to finish.

Assembly Code Example

```
EEPROM_write:
      sbic  EECR, EEPE
      rjmp  EEPROM_write; Wait for completion of previous write

      ldi   r16,  (0<<EEPM1)|(0<<EEPM0)
      out   EECR, r16    ; Set Programming mode

      out   EEARH,r18
      out   EEARL,r17    ; Set up address (r18:r17) in address registers

      out   EEDR, r19    ; Write data (r19) to data register

      sbi   EECR, EEMPE ; Write logical one to EEMPE

      sbi   EECR, EEPE   ; Start eeprom write by setting EEPE
      ret
```

Note: See “Code Examples” on page 7.

C Code Example

```
void EEPROM_write(unsigned int ucAddress, unsigned char ucData)
{
      /* Wait for completion of previous write */
      while(EECR & (1<<EEPE))
            ;
      /* Set Programming mode */
      EECR = (0<<EEPM1)|(0<<EEPM0)

      /* Set up address and data registers */
      EEAR = ucAddress;
      EEDR = ucData;

      /* Write logical one to EEMPE */
      EECR |= (1<<EEMPE);

      /* Start eeprom write by setting EEPE */
      EECR |= (1<<EEPE);
}
```

Note: See “Code Examples” on page 7.

The next code examples show assembly and C functions for reading the EEPROM. The examples assume that interrupts are controlled so that no interrupts will occur during execution of these functions.

Assembly Code Example

```
EEPROM_read:

      sbic    EECR,EEPE
      rjmp    EEPROM_read ; Wait for completion of previous write

      out     EEARH,r18
      out     EEARL,r17   ; Set up address (r18:r17) in address registers

      sbi     EECR,EERE   ; Start eeprom read by writing EERE

      in      r16, EEDR   ; Read data from data register
      ret
```

Note: See “Code Examples” on page 7.

C Code Example

```
unsigned char EEPROM_read(unsigned int ucAddress)
{
      /* Wait for completion of previous write */
      while(EECR & (1<<EEPE))
            ;
      /* Set up address register */
      EEAR = ucAddress;

      /* Start eeprom read by writing EERE */
      EECR |= (1<<EERE);

      /* Return data from data register */
      return EEDR;
}
```

Note: See “Code Examples” on page 7.

5.4 Register Description

5.4.1 EEARL – EEPROM Address Register Low

Bit	7	6	5	4	3	2	1	0	
0x21 (0x41)	**EEAR7**	**EEAR6**	**EEAR5**	**EEAR4**	**EEAR3**	**EEAR2**	**EEAR1**	**EEAR0**	**EEARL**
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	

- **Bits 7:0 – EEAR[7:0]: EEPROM Address**

The EEPROM address register is required by the read and write operations to indicate the memory location that is being accessed.

EEPROM data bytes are addressed linearly over the entire memory range (0...[256-1]). The initial value of these bits is undefined and a legitimate value must therefore be written to the register before EEPROM is accessed.

Devices with 256 bytes of EEPROM, or less, do not require a high address registers (EEARH). In such devices the high address register is therefore left out but, for compatibility issues, the remaining register is still referred to as the low byte of the EEPROM address register (EEARL).

Devices that to do not fill an entire address byte, i.e. devices with an EEPROM size not equal to 256, implement read-only bits in the unused locations. Unused bits are located in the most significant end of the address register and they always read zero.

5.4.2 EEDR – EEPROM Data Register

Bit	7	6	5	4	3	2	1	0	
0x20 (0x40)	EEDR7	EEDR6	EEDR5	EEDR4	EEDR3	EEDR2	EEDR1	EEDR0	EEDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – EEDR[7:0]: EEPROM Data**

For EEPROM write operations, EEDR contains the data to be written to the EEPROM address given in the EEAR Register. For EEPROM read operations, EEDR contains the data read out from the EEPROM address given by EEAR.

5.4.3 EECR – EEPROM Control Register

Bit	7	6	5	4	3	2	1	0	
0x1F (0x3F)	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	EECR
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	X	X	0	0	X	0	

- **Bits 7, 6 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bits 5, 4 – EEPM1 and EEPM0: EEPROM Programming Mode Bits**

EEPROM programming mode bits define the action that will be triggered when EEPE is written. Data can be programmed in a single atomic operation, where the previous value is automatically erased before the new value is programmed, or Erase and Write can be split in two different operations. The programming times for the different modes are shown in Table 5.

Table 5. EEPROM Programming Mode Bits and Programming Times

EEPM1	EEPM0	Programming Time	Operation
0	0	3.4 ms	Atomic (erase and write in one operation)
0	1	1.8 ms	Erase, only
1	0	1.8 ms	Write, only
1	1	–	Reserved

When EEPE is set any write to EEPMn will be ignored.

During reset, the EEPMn bits will be reset to 0b00 unless the EEPROM is busy programming.

- **Bit 3 – EERIE: EEPROM Ready Interrupt Enable**

Writing this bit to one enables the EEPROM Ready Interrupt. Provided the I-bit in SREG is set, the EEPROM Ready Interrupt is triggered when non-volatile memory is ready for programming.

Writing this bit to zero disables the EEPROM Ready Interrupt.

- **Bit 2 – EEMPE: EEPROM Master Program Enable**

The EEMPE bit determines whether writing EEPE to one will have effect or not.

When EEMPE is set and EEPE written within four clock cycles the EEPROM at the selected address will be programmed. Hardware clears the EEMPE bit to zero after four clock cycles.

If EEMPE is zero the EEPE bit will have no effect.

- **Bit 1 – EEPE: EEPROM Program Enable**

This is the programming enable signal of the EEPROM. The EEMPE bit must be set before EEPE is written, or EEPROM will not be programmed.

When EEPE is written, the EEPROM will be programmed according to the EEPMn bit settings. When EEPE has been set, the CPU is halted for two cycles before the next instruction is executed. After the write access time has elapsed, the EEPE bit is cleared by hardware.

Note that an EEPROM write operation blocks all software programming of Flash, fuse bits, and lock bits.

- **Bit 0 – EERE: EEPROM Read Enable**

This is the read strobe of the EEPROM. When the target address has been set up in the EEAR, the EERE bit must be written to one to trigger the EEPROM read operation.

EEPROM read access takes one instruction, and the requested data is available immediately. When the EEPROM is read, the CPU is halted for four cycles before the next instruction is executed.

The user should poll the EEPE bit before starting the read operation. If a write operation is in progress, it not possible to read the EEPROM, or to change the address register (EEAR).

5.4.4 GPIOR2 – General Purpose I/O Register 2

Bit	7	6	5	4	3	2	1	0	
0x2B (0x4B)	**MSB**							**LSB**	**GPIOR2**
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	

This register may be used freely for storing any kind of data.

5.4.5 GPIOR1 – General Purpose I/O Register 1

Bit	7	6	5	4	3	2	1	0	
0x2A (0x4A)	**MSB**							**LSB**	**GPIOR1**
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	

This register may be used freely for storing any kind of data.

5.4.6 GPIOR0 – General Purpose I/O Register 0

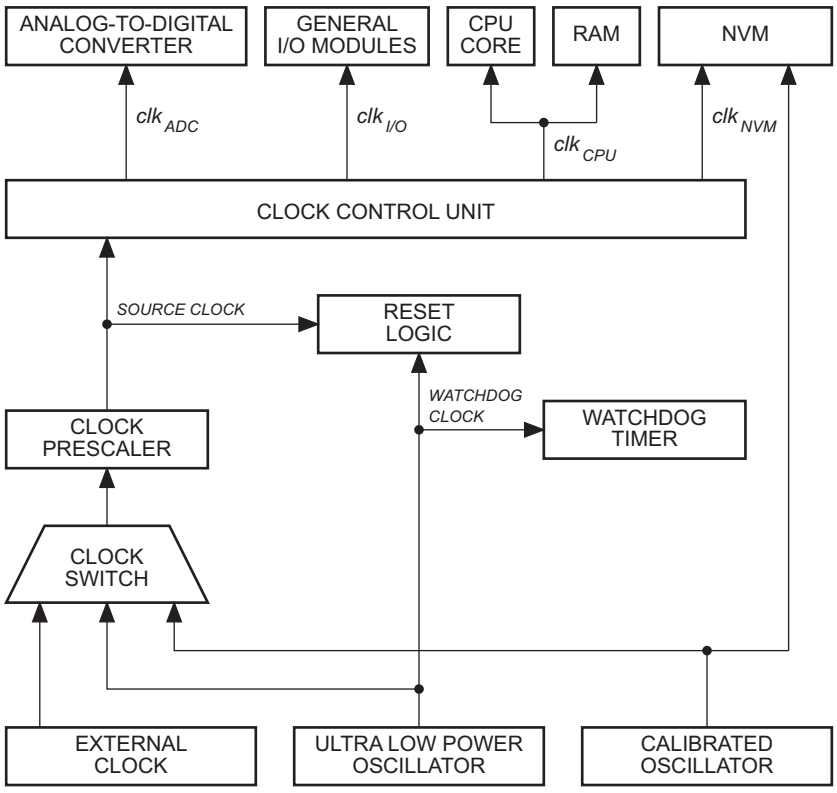
Bit	7	6	5	4	3	2	1	0	
0x1E (0x3E)	**MSB**							**LSB**	**GPIOR0**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

This register may be used freely for storing any kind of data.

6. Clock System

Figure 11 presents the principal clock systems and their distribution in ATtiny828. All of the clocks need not be active at a given time. In order to reduce power consumption, the clocks to modules not being used can be halted by using different sleep modes and power reduction register bits, as described in “Power Management and Sleep Modes” on page 34. The clock systems is detailed below.

Figure 11. Clock Distribution



6.1 Clock Subsystems

The clock subsystems are detailed in the sections below.

6.1.1 CPU Clock – clk_{CPU}

The CPU clock is routed to parts of the system concerned with operation of the AVR Core. Examples of such modules are the General Purpose Register File, the System Registers and the SRAM data memory. Halting the CPU clock inhibits the core from performing general operations and calculations.

6.1.2 I/O Clock – $clk_{I/O}$

The I/O clock is used by the majority of the I/O modules, like Timer/Counter. The I/O clock is also used by the External Interrupt module, but note that some external interrupts are detected by asynchronous logic, allowing such interrupts to be detected even if the I/O clock is halted.

6.1.3 NVM clock - clk_{NVM}

The NVM clock controls operation of the Non-Volatile Memory Controller. The NVM clock is usually active simultaneously with the CPU clock.

6.1.4 ADC Clock – clk_{ADC}

The ADC is provided with a dedicated clock domain. This allows halting the CPU and I/O clocks in order to reduce noise generated by digital circuitry. This gives more accurate ADC conversion results.

6.2 Clock Sources

The device can use any of the following sources for the system clock:

- External Clock (see page 28)
- Calibrated Internal 8MHz Oscillator (see page 29)
- Internal 32kHz Ultra Low Power (ULP) Oscillator (see page 29)

The clock source is selected using CKSEL fuses, as shown in Table 6 below.

Table 6. CKSEL Fuse Bits and Device Clocking Options

CKSEL[1:0] (1)	Frequency	Device Clocking Option
0X	Any	External Clock (see page 28)
10	8MHz	Calibrated Internal 8MHz Oscillator (see page 29) (2)
11	32kHz	Internal 32kHz Ultra Low Power (ULP) Oscillator (see page 29)

Note: 1. For all fuses “1” means unprogrammed and “0” means programmed.
2. This is the default setting. The device is shipped with this fuse combination.

.

CKSEL fuse bits can be read by firmware (see “Reading Lock, Fuse and Signature Data from Software” on page 229), but firmware can not write to fuse bits.

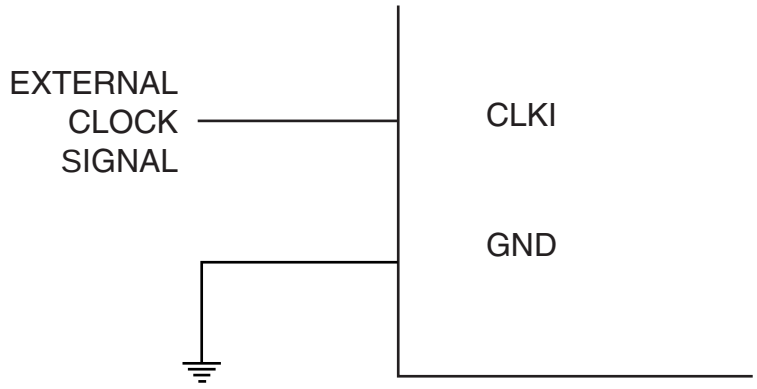
When the device wakes up from power-down the selected clock source is used to time the start-up, ensuring stable oscillator operation before instruction execution starts. When the CPU starts from reset, the internal 32kHz oscillator is used for generating an additional delay, allowing supply voltage to reach a stable level before normal device operation is started.

System clock alternatives are discussed in the following sections.

6.2.1 External Clock

To drive the device from an external clock source, CLKI should be connected as shown in Figure 12, below.

Figure 12. External Clock Drive Configuration



Start-up time for this clock source is determined by the SUT fuse bit, as shown in Table 7 on page 30.

To ensure stable operation of the MCU it is required to avoid sudden changes in the external clock frequency . A variation in frequency of more than 2% from one clock cycle to the next can lead to unpredictable behavior. It is required to ensure that the MCU is kept in Reset during such changes in the clock frequency.

Stable operation for large step changes in system clock frequency is guaranteed when using the system clock prescaler. See “System Clock Prescaler” on page 30.

6.2.2 Calibrated Internal 8MHz Oscillator

The internal 8MHz oscillator operates with no external components and, by default, provides a clock source with an approximate frequency of 8MHz. Though voltage and temperature dependent, this clock can be very accurately calibrated by the user. See Table 104 on page 249 and “Internal Oscillator Speed” on page 293 for more details.

During reset, hardware loads the pre-programmed calibration value into the OSCCAL0 register and thereby automatically calibrates the oscillator. The accuracy of this calibration is referred to as “Factory Calibration” in Table 104 on page 249. For more information on automatic loading of pre-programmed calibration value, see section “Calibration Bytes” on page 229.

It is possible to reach higher accuracies than factory defaults, especially when the application allows temperature and voltage ranges to be narrowed. The firmware can reprogram the calibration data in OSCCAL0 either at start-up or during run-time. The continuous, run-time calibration method allows firmware to monitor voltage and temperature and compensate for any detected variations. See “OSCCAL0 – Oscillator Calibration Register” on page 32, “Temperature Measurement” on page 148, and Table 52 on page 150. The accuracy of this calibration is referred to as “User Calibration” in Table 104 on page 249.

The oscillator temperature calibration registers, OSCTCAL0A and OSCTCAL0B, can be used for one-time temperature calibration of oscillator frequency. See “OSCTCAL0A – Oscillator Temperature Calibration Register A” on page 33 and “OSCTCAL0B – Oscillator Temperature Calibration Register B” on page 33.

When this oscillator is used as the chip clock, it will still be used for the Watchdog Timer and for the Reset Time-out.

Start-up time for this clock source is determined by the SUT fuse bit, as shown in Table 7 on page 30.

6.2.3 Internal 32kHz Ultra Low Power (ULP) Oscillator

The internal 32kHz oscillator is a low power oscillator that operates with no external components. It provides a clock source with an approximate frequency of 32kHz. The frequency depends on supply voltage, temperature and batch variations. See Table 105 on page 250 for accuracy details.

During reset, hardware loads the pre-programmed calibration value into the OSCCAL1 register and thereby automatically calibrates the oscillator. The accuracy of this calibration is referred to as “Factory Calibration” in Table 105 on page 250. For more information on automatic loading of pre-programmed calibration value, see section “Calibration Bytes” on page 229.

Start-up time for this clock source is determined by the SUT fuse bit, as shown in Table 7 on page 30.

6.2.4 Default Clock Settings

The device is shipped with following fuse settings:

- Calibrated Internal 8MHz Oscillator (see CKSEL fuse bits in Table 6 on page 28)
- Longest possible start-up time (see SUT fuse bits in Table 7 on page 30)
- System clock prescaler set to 8 (see CKDIV8 fuse bit on page 32)

The default setting gives a 1MHz system clock and ensures all users can make their desired clock source setting using an in-system or high-voltage programmer.

6.3 System Clock Prescaler

The ATtiny828 system clock can be divided by setting the “CLKPR – Clock Prescale Register” on page 31. This feature can be used to decrease power consumption when the requirement for processing power is low. This can be used with all clock source options, and it will affect the clock frequency of the CPU and all synchronous peripherals. $clk_{I/O}$, clk_{ADC}, clk_{CPU}, and clk_{FLASH} are divided by a factor as shown in Table 8 on page 31.

6.3.1 Switching Prescaler Setting

When switching between prescaler settings, the System Clock Prescaler ensures that no glitch occurs in the clock system and that no intermediate frequency is higher than neither the clock frequency corresponding to the previous setting, nor the clock frequency corresponding to the new setting.

The ripple counter that implements the prescaler runs at the frequency of the undivided clock, which may be faster than the CPU's clock frequency. Hence, it is not possible to determine the state of the prescaler - even if it were readable, and the exact time it takes to switch from one clock division to another cannot be exactly predicted.

From the time the CLKPS values are written, it takes between T1 + T2 and T1 + 2*T2 before the new clock frequency is active. In this interval, 2 active clock edges are produced. Here, T1 is the previous clock period, and T2 is the period corresponding to the new prescaler setting.

6.4 Clock Output Buffer

The device can output the system clock on the CLKO pin. To enable the output, the CKOUT fuse has to be programmed.

This mode is suitable when the chip clock is used to drive other circuits on the system. Note that the clock will not be output during reset and that the normal operation of the I/O pin will be overridden when the fuse is programmed. Any clock source, including the internal oscillators, can be selected when the clock is output on CLKO. If the System Clock Prescaler is used, it is the divided system clock that is output.

6.5 Start-Up Time

The CKSEL and SUT fuse bits define the start-up time of the device, as shown in Table 7 on page 30, below.

Table 7. CKSEL and SUT Fuse Bits vs. Device Start-up Time

CKSEL	SUT	Clock	From Power-Down (1)(2)	From Reset (3)
0X	00	External	6 CK	20 CK
	01			20 CK + 4ms
	1X			20 CK + 64ms
10 (4)	00	Internal 8MHz	6 CK (5)	20 CK (5)
	01			20 CK + 4ms
	1X (4)			20 CK + 64ms
11	00	Internal 32kHz		20 CK (5)
	01			20 CK + 4ms
	1X			20 CK + 64ms

Note:
1. Device start-up time from power-down sleep mode.
2. When BOD has been disabled by software, the wake-up time from sleep mode will be approximately 60µs to ensure the BOD is working correctly before MCU continues executing code.
3. Device start-up time after reset.
4. The device is shipped with this option selected.

5. At least 4ms when reset is disabled.

6.6 Register Description

6.6.1 CLKPR – Clock Prescale Register

Bit	7	6	5	4	3	2	1	0	
(0x61)	–	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0	CLKPR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	See Bit Description				

- **Bits 7:4 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bits 3:0 – CLKPS[3:0]: Clock Prescaler Select Bits 3 - 0**

These bits define the division factor between the selected clock source and the internal system clock. These bits can be written run-time to vary the clock frequency to suit the application requirements. As the divider divides the master clock input to the MCU, the speed of all synchronous peripherals is reduced when a division factor is used. The division factors are given in Table 8.

Interrupts must be disabled when changing prescaler setting to make sure the write procedure is not interrupted.

Table 8. Clock Prescaler Select

CLKPS3	CLKPS2	CLKPS1	CLKPS0	Clock Division Factor
0	0	0	0	1 (1)
0	0	0	1	2
0	0	1	0	4
0	0	1	1	8 (2)
0	1	0	0	16
0	1	0	1	32
0	1	1	0	64
0	1	1	1	128
1	0	0	0	256
1	0	0	1	Reserved
1	0	1	0	Reserved
1	0	1	1	Reserved
1	1	0	0	Reserved
1	1	0	1	Reserved
1	1	1	0	Reserved
1	1	1	1	Reserved

Note: 1. This is the initial value when CKDIV8 fuse has been unprogrammed.
2. This is the initial value when CKDIV8 fuse has been programmed. The device is shipped with the CKDIV8 Fuse programmed.

The initial value of clock prescaler bits is determined by the CKDIV8 fuse (see Table 91 on page 227). When CKDIV8 is unprogrammed, the system clock prescaler is set to one and, when programmed, to eight. Any value can be written to the CLKPS bits regardless of the CKDIV8 fuse bit setting.

When CKDIV8 is programmed the initial value of CLKPS bits give a clock division factor of eight at start up. This is useful when the selected clock source has a higher frequency than allowed under present operating conditions. See "Speed" on page 249.

To avoid unintentional changes to clock frequency, the following sequence must be followed:

1. Write the required signature to the CCP register. See page 14.
2. Within four instruction cycles, write the desired value to CLKPS bits.

6.6.2 OSCCAL0 – Oscillator Calibration Register

Bit	7	6	5	4	3	2	1	0	
(0x66)	**CAL07**	**CAL06**	**CAL05**	**CAL04**	**CAL03**	**CAL02**	**CAL01**	**CAL00**	**OSCCAL0**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	Device Specific Calibration Value								

- **Bits 7:0 – CAL0[7:0]: Oscillator Calibration Value**

The oscillator calibration register is used to trim the internal 8MHz oscillator and to remove process variations from the oscillator frequency. A pre-programmed calibration value is automatically written to this register during chip reset, giving the factory calibrated frequency specified in Table 104 on page 249.

The application software can write this register to change the oscillator frequency. The oscillator can be calibrated to frequencies specified in Table 104 on page 249. Calibration outside that range is not guaranteed.

The lowest oscillator frequency is reached by programming these bits to zero. Increasing the register value increases the oscillator frequency. A typical frequency response curve is shown in "Calibrated Oscillator Frequency vs. OSCCAL0 Value" on page 295.

Note that this oscillator is used to time EEPROM and Flash write accesses, and write times will be affected accordingly. Do not calibrate to more than 8.8MHz if EEPROM or Flash is to be written. Otherwise, the EEPROM or Flash write may fail.

To ensure stable operation of the MCU the calibration value should be changed in small steps. A step change in frequency of more than 2% from one cycle to the next can lead to unpredictable behavior. Also, the difference between two consecutive register values should not exceed 0x20. If these limits are exceeded the MCU must be kept in reset during changes to clock frequency.

6.6.3 OSCTCAL0A – Oscillator Temperature Calibration Register A

Bit	7	6	5	4	3	2	1	0	
(0xF0)	Oscillator Temperature Calibration Data								OSCTCAL0A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	Device Specific Calibration Value								

- **Bits 7:0 – Oscillator Temperature Calibration Value**

The temperature calibration value can be used to trim the calibrated 8MHz oscillator and remove temperature variations from the oscillator frequency.

6.6.4 OSCTCAL0B – Oscillator Temperature Calibration Register B

Bit	7	6	5	4	3	2	1	0	
(0xF1)	Oscillator Temperature Calibration Data								OSCTCAL0B
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	Device Specific Calibration Value								

- **Bits 7:0 – Oscillator Temperature Calibration Value**

The temperature calibration value can be used to trim the calibrated 8MHz oscillator and remove temperature variations from the oscillator frequency.

6.6.5 OSCCAL1 – Oscillator Calibration Register

Bit	7	6	5	4	3	2	1	0	
(0x67)	–	–	–	–	–	–	CAL11	CAL10	OSCCAL1
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	Calibration Value		

- **Bits 7:0 – CAL[11:10]: Oscillator Calibration Value**

The oscillator calibration register is used to trim the internal 32kHz oscillator and to remove process variations from the oscillator frequency. A pre-programmed calibration value is automatically written to this register during chip reset, giving the factory calibrated frequency as specified in Table 105 on page 250.

The application software can write this register to change the oscillator frequency. The oscillator can be calibrated to frequencies as specified in Table 105 on page 250. Calibration outside that range is not guaranteed.

The lowest oscillator frequency is reached by programming these bits to zero. Increasing the register value increases the oscillator frequency. A typical frequency response curve is shown in "ULP Oscillator Frequency vs. OSCCAL1 Value" on page 296.

To ensure stable operation of the MCU the calibration value should be changed in small steps. A step change in frequency of more than 2% from one cycle to the next can lead to unpredictable behavior. Also, the difference between two consecutive register values should not exceed 0x20. If these limits are exceeded the MCU must be kept in reset during changes to clock frequency.

7. Power Management and Sleep Modes

The high performance and industry leading code efficiency makes the AVR microcontrollers an ideal choise for low power applications. In addition, sleep modes enable the application to shut down unused modules in the MCU, thereby saving power. The AVR provides various sleep modes allowing the user to tailor the power consumption to the application's requirements.

7.1 Sleep Modes

Figure 11 on page 27 presents the different clock systems and their distribution in ATtiny828. The figure is helpful in selecting an appropriate sleep mode. Table 9 shows the different sleep modes and the sources that may be used for wake up.

Table 9. Active Clock Domains and Wake-up Sources in Different Sleep Modes

	Oscillators	Active Clock Domains				Wake-up Sources						
Sleep Mode	Main Clock Source Enabled	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	Watchdog Interrupt	INT0 and Pin Change	SPM/EEPROM Ready Interrupt	ADC Interrupt	USART (1)	TWI Slave	Other I/O
Idle	X			X	X	X	X	X	X	X	X	X
ADC Noise Reduction	X				X	X	X(3)	X	X	X	X(2)	
Power-down						X	X(3)			X	X(2)	

Note: 1. Start frame detection, only.
2. Address match interrupt, only.
3. For INT0 level interrupt, only.

To enter a sleep mode, the SE bit in MCUCR must be set and a SLEEP instruction must be executed. The SMn bits in MCUCR select which sleep mode will be activated by the SLEEP instruction. See Table 10 on page 37 for a summary.

If an enabled interrupt occurs while the MCU is in a sleep mode, the MCU wakes up. The MCU is then halted for four cycles in addition to the start-up time, executes the interrupt routine, and resumes execution from the instruction following SLEEP. The contents of the Register File and SRAM are unaltered when the device wakes up from sleep. If a reset occurs during sleep mode, the MCU wakes up and executes from the Reset Vector.

Note that if a level triggered interrupt is used for wake-up the changed level must be held for some time to wake up the MCU (and for the MCU to enter the interrupt service routine). See "External Interrupts" on page 51 for details.

7.1.1 Idle Mode

This sleep mode basically halts clk_{CPU} and clk_{FLASH}, while allowing other clocks to run. In Idle Mode, the CPU is stopped but the following peripherals continue to operate:

- Watchdog and interrupt system
- Analog comparator, and ADC
- USART, TWI, and timer/counters

Idle mode allows the MCU to wake up from external triggered interrupts as well as internal ones, such as Timer Overflow. If wake-up from the analog comparator interrupt is not required, the analog comparator can be powered down by setting

the ACD bit in ACSRA. See “ACSRA – Analog Comparator Control and Status Register” on page 134. This will reduce power consumption in Idle mode.

If the ADC is enabled, a conversion starts automatically when this mode is entered.

7.1.2 ADC Noise Reduction Mode

This sleep mode halts $clk_{I/O}$, clk_{CPU}, and clk_{FLASH}, while allowing other clocks to run. In ADC Noise Reduction mode, the CPU is stopped but the following peripherals continue to operate:

- Watchdog (if enabled), and external interrupts
- ADC
- USART start frame detector, and TWI

This improves the noise environment for the ADC, enabling higher resolution measurements. If the ADC is enabled, a conversion starts automatically when this mode is entered.

The following events can wake up the MCU:

- Watchdog reset, external reset, and brown-out reset
- External level interrupt on INT0, and pin change interrupt
- ADC conversion complete interrupt, and SPM/EEPROM ready interrupt
- USART start frame detection, and TWI slave address match

7.1.3 Power-Down Mode

This sleep mode halts all generated clocks, allowing operation of asynchronous modules, only. In Power-down Mode the oscillator is stopped, while the following peripherals continue to operate:

- Watchdog (if enabled), external interrupts

The following events can wake up the MCU:

- Watchdog reset, external reset, and brown-out reset
- External level interrupt on INT0, and pin change interrupt
- USART start frame detection, and TWI slave address match

7.2 Power Reduction Register

The Power Reduction Register (PRR), see “PRR – Power Reduction Register” on page 37, provides a method to reduce power consumption by stopping the clock to individual peripherals. When the clock for a peripheral is stopped then:

- The current state of the peripheral is frozen.
- The associated registers can not be read or written.
- Resources used by the peripheral will remain occupied.

The peripheral should in most cases be disabled before stopping the clock. Clearing the PRR bit wakes up the peripheral and puts it in the same state as before shutdown.

Peripheral shutdown can be used in Idle mode and Active mode to significantly reduce the overall power consumption. See “Current Consumption of Peripheral Units” on page 266 for examples. In all other sleep modes, the clock is already stopped.

7.3 Minimizing Power Consumption

There are several issues to consider when trying to minimize the power consumption in an AVR controlled system. In general, sleep modes should be used as much as possible, and the sleep mode should be selected so that as few as

possible of the device's functions are operating. All functions not needed should be disabled. In particular, the following modules may need special consideration when trying to achieve the lowest possible power consumption.

7.3.1 Analog to Digital Converter

If enabled, the ADC will be enabled in all sleep modes. To save power, the ADC should be disabled before entering any sleep mode. When the ADC is turned off and on again, the next conversion will be an extended conversion. See "Analog to Digital Converter" on page 137 for details on ADC operation.

7.3.2 Analog Comparator

When entering Idle mode, the Analog Comparator should be disabled if not used. When entering ADC Noise Reduction mode, the Analog Comparator should be disabled. In the other sleep modes, the Analog Comparator is automatically disabled. However, if the Analog Comparator is set up to use the Internal Voltage Reference as input, the Analog Comparator should be disabled in all sleep modes. Otherwise, the Internal Voltage Reference will be enabled, independent of sleep mode. See "Analog Comparator" on page 133 for details on how to configure the Analog Comparator.

7.3.3 Brown-out Detector

If the Brown-out Detector is not needed in the application, this module should be turned off. If the Brown-out Detector is enabled by the BODPD Fuses, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. If the Brown-out Detector is needed in the application, this module can also be set to Sampled BOD mode to save power. See "Brown-out Detection" on page 41 for details on how to configure the Brown-out Detector.

7.3.4 Internal Voltage Reference

The Internal Voltage Reference will be enabled when needed by the Brown-out Detection, the Analog Comparator or the ADC. If these modules are disabled as described in the sections above, the internal voltage reference will be disabled and it will not be consuming power. When turned on again, the user must allow the reference to start up before the output is used. If the reference is kept on in sleep mode, the output can be used immediately. See Internal Bandgap Reference in Table 107 on page 250 for details on the start-up time.

7.3.5 Watchdog Timer

If the Watchdog Timer is not needed in the application, this module should be turned off. If the Watchdog Timer is enabled, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute to the total current consumption. See "Brown-out Detection" on page 41 for details on how to configure the Watchdog Timer.

7.3.6 Port Pins

When entering a sleep mode, all port pins should be configured to use minimum power. The most important thing is then to ensure that no pins drive resistive loads. In sleep modes where both the I/O clock ($clk_{I/O}$) and the ADC clock (clk_{ADC}) are stopped, the input buffers of the device will be disabled. This ensures that no power is consumed by the input logic when not needed. In some cases, the input logic is needed for detecting wake-up conditions, and it will then be enabled. See the section "Digital Input Enable and Sleep Modes" on page 62 for details on which pins are enabled. If the input buffer is enabled and the input signal is left floating or has an analog signal level close to $V_{CC}/2$, the input buffer will use excessive power.

For analog input pins, the digital input buffer should be disabled at all times. An analog signal level close to $V_{CC}/2$ on an input pin can cause significant current even in active mode. Digital input buffers can be disabled by writing to the Digital Input Disable Register (DIDR0). See "DIDR3 – Digital Input Disable Register 3" on page 154 for details.

7.4 Register Description

7.4.1 SMCR – Sleep Mode Control Register

The Sleep Mode Control Register contains control bits for power management.

Bit	7	6	5	4	3	2	1	0	
0x33 (0x53)	–	–	–	–	–	SM1	SM0	SE	SMCR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:3 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bits 2:1 – SM[1:0]: Sleep Mode Select Bits 1 and 0**

These bits select the sleep mode, as shown in Table 10.

Table 10. Sleep Mode Select

SM1	SM0	Sleep Mode
0	0	Idle
0	1	ADC Noise Reduction
1	0	Power-down
1	1	Reserved

- **Bit 0 – SE: Sleep Enable**

This bit must be written to logic one to make the MCU enter the sleep mode when the SLEEP instruction is executed. To avoid the MCU entering sleep mode unintentionally, it is recommended to write the Sleep Enable (SE) bit to one just before the execution of the SLEEP instruction and to clear it immediately after waking up.

7.4.2 PRR – Power Reduction Register

The Power Reduction Register provides a method to reduce power consumption by allowing peripheral clock signals to be disabled.

Bit	7	6	5	4	3	2	1	0	
(0x64)	PRTWI	–	PRTIM0	–	PRTIM1	PRSPI	PRUSART0	PRADC	PRR
Read/Write	R/W	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – PRTWI: Power Reduction Two-Wire Interface**

Writing a logic one to this bit shuts down the Two-Wire Interface module.

- **Bits 6, 4 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 5 – PRTIM0: Power Reduction Timer/Counter0**

Writing a logic one to this bit shuts down the Timer/Counter0 module. When the Timer/Counter0 is enabled, operation will continue like before the shutdown.

- **Bit 3 – PRTIM1: Power Reduction Timer/Counter1**

Writing a logic one to this bit shuts down the Timer/Counter1 module. When the Timer/Counter1 is enabled, operation will continue like before the shutdown.

- **Bit 2 – PRSPI: Power Reduction SPI**

Writing a logic one to this bit shuts down the SPI by stopping the clock to the module. When waking up the SPI again, the SPI should be re-initialized to ensure proper operation.

- **Bit 1 – PRUSART0: Power Reduction USART0**

Writing a logic one to this bit shuts down the USART0 module. When the USART0 is enabled, operation will continue like before the shutdown.

- **Bit 0 – PRADC: Power Reduction ADC**

Writing a logic one to this bit shuts down the ADC. The ADC must be disabled before shut down. The analog comparator cannot be used when the ADC is shut down.

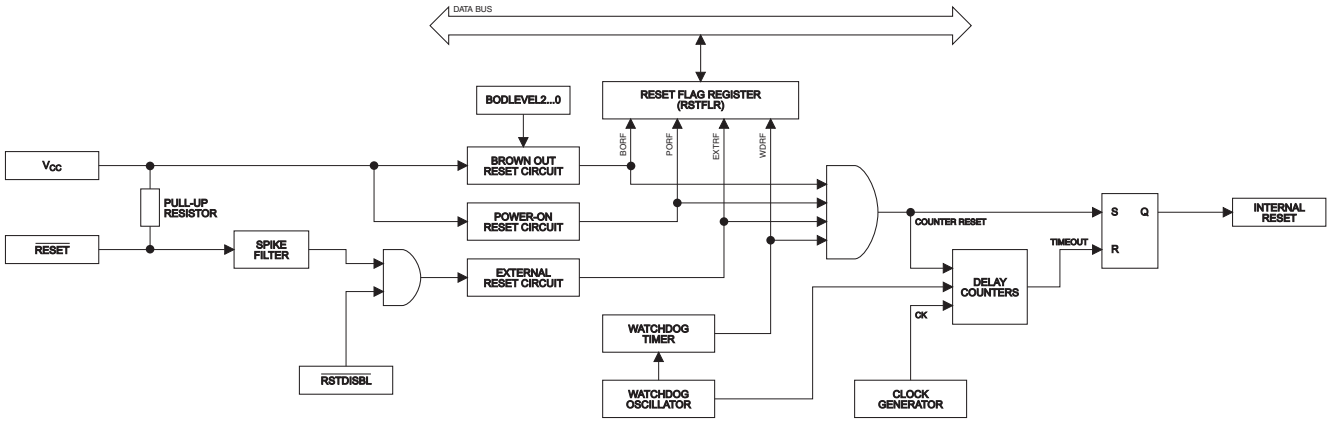
8. System Control and Reset

8.1 Resetting the AVR

During reset, all I/O registers are set to their initial values, and the program starts execution from the Reset Vector. The instruction placed at the Reset Vector must be a RJMP – Relative Jump – instruction to the reset handling routine. 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.

The circuit diagram in Figure 13 shows the reset logic. Electrical parameters of the reset circuitry are defined in section "System and Reset Characteristics" on page 250.

Figure 13. Reset Logic



The I/O ports of the AVR are immediately reset to their initial state when a reset source goes active. This does not require any clock source to be running.

After all reset sources have gone inactive, a delay counter is invoked, stretching the internal reset. This allows the power to reach a stable level before normal operation starts.

8.2 Reset Sources

The ATtiny828 has four sources of reset:

- Power-on Reset. The MCU is reset when the supply voltage is below the Power-on Reset threshold (V_{POT})
- External Reset. The MCU is reset when a low level is present on the $\overline{\text{RESET}}$ pin for longer than the minimum pulse length
- Watchdog Reset. The MCU is reset when the Watchdog Timer period expires and the Watchdog is enabled
- Brown Out Reset. The MCU is reset when the Brown-Out Detector is enabled and supply voltage is below the brown-out threshold (V_{BOT})

8.2.1 Power-on Reset

A Power-on Reset (POR) pulse is generated by an on-chip detection circuit. The detection level is defined in section "System and Reset Characteristics" on page 250. The POR is activated whenever V_{CC} is below the detection level. The POR circuit can be used to trigger the Start-up Reset, as well as to detect a failure in supply voltage.

A Power-on Reset (POR) circuit ensures that the device is reset from Power-on. Reaching the Power-on Reset threshold voltage invokes the delay counter, which determines how long the device is kept in reset after V_{CC} rise. The reset signal is activated again, without any delay, when V_{CC} decreases below the detection level.

Figure 14. MCU Start-up, **$\overline{\text{RESET}}$** Tied to V_{CC}

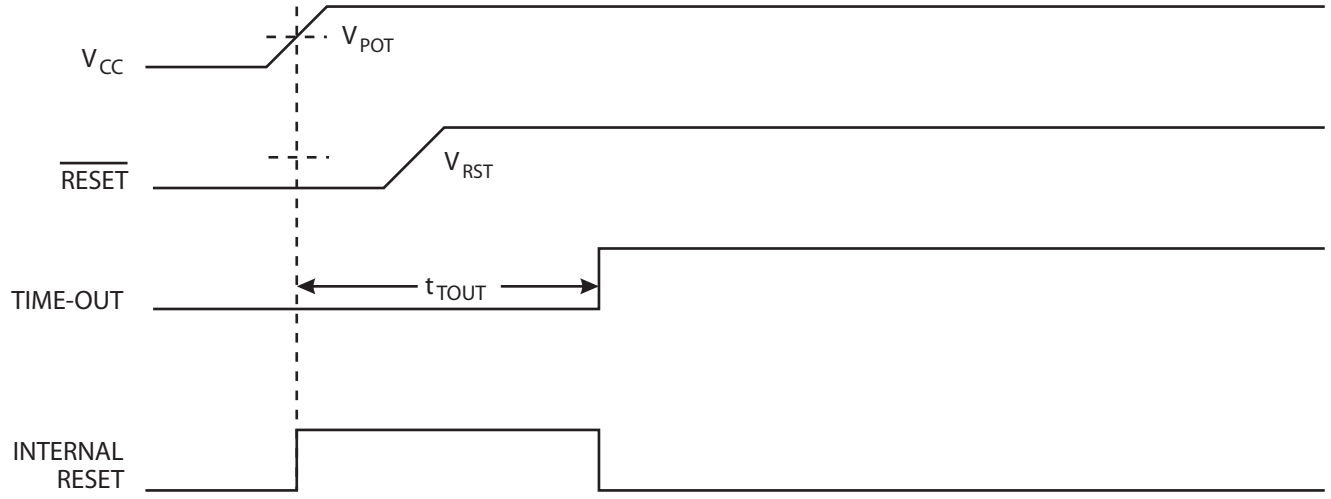
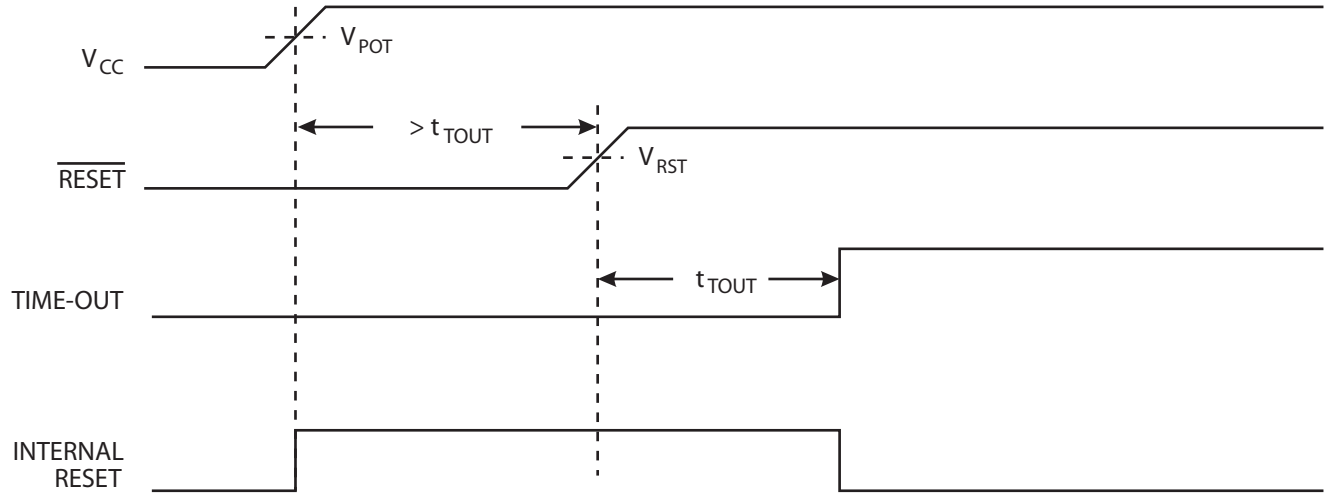


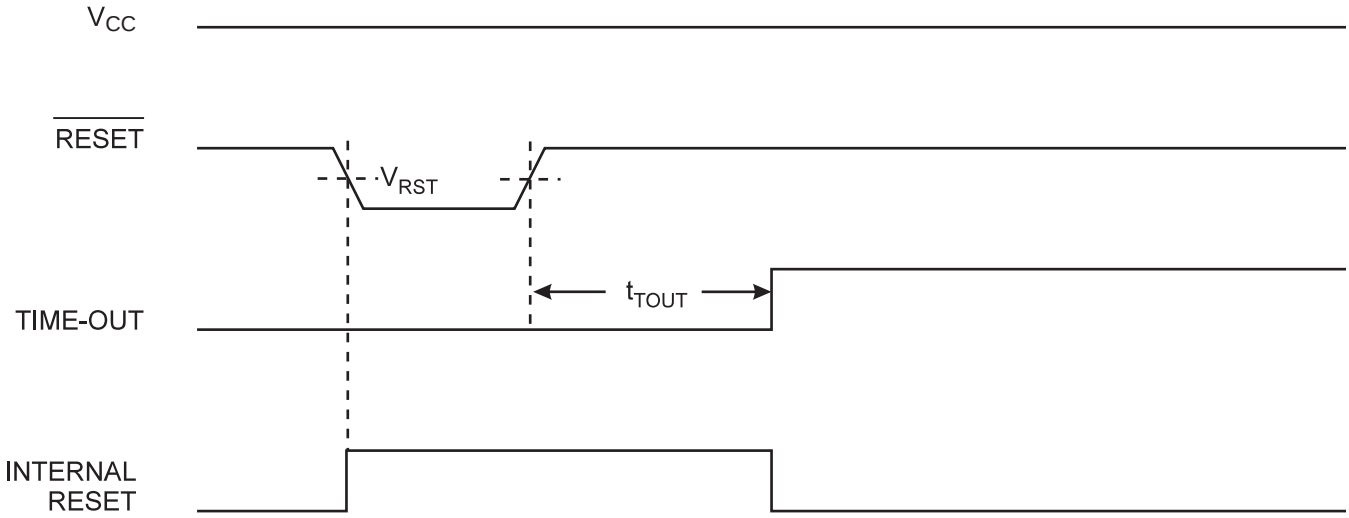
Figure 15. MCU Start-up, **$\overline{\text{RESET}}$** Extended Externally



8.2.2 External Reset

An External Reset is generated by a low level on the $\overline{\text{RESET}}$ pin if enabled. Reset pulses longer than the minimum pulse width (see section “System and Reset Characteristics” on page 250) will generate a reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a reset. When the applied signal reaches the Reset Threshold Voltage – V_{RST} – on its positive edge, the delay counter starts the MCU after the time-out period – t_{TOUT} – has expired.

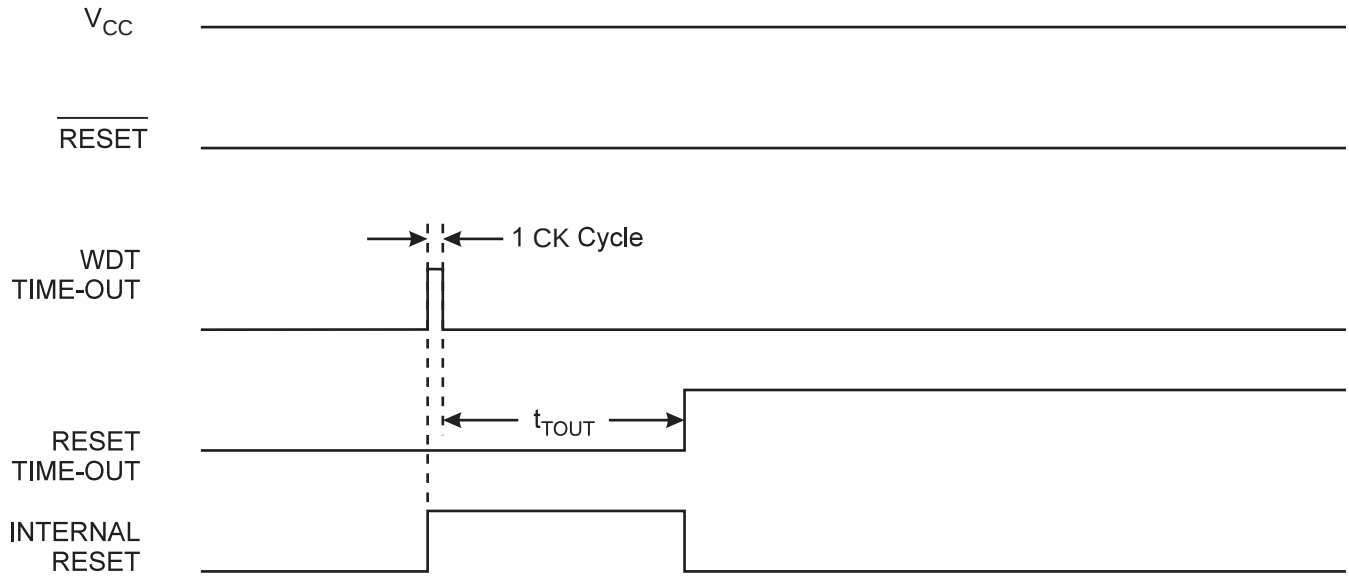
Figure 16. External Reset During Operation



8.2.3 Watchdog Reset

When the Watchdog times out, it will generate a short reset pulse. On the falling edge of this pulse, the delay timer starts counting the time-out period t_{TOUT}. See page 41 for details on operation of the Watchdog Timer.

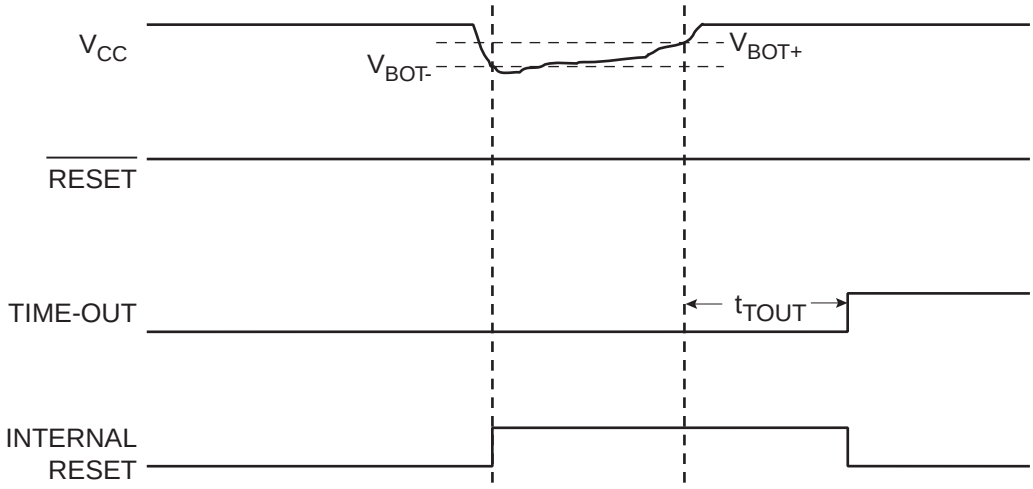
Figure 17. Watchdog Reset During Operation



8.2.4 Brown-out Detection

The Brown-Out Detection (BOD) circuit monitors that the V_{CC} level is kept above a configurable trigger level, V_{BOT}. When the BOD is enabled, a BOD reset will be given when V_{CC} falls and remains below the trigger level for the length of the detection time, t_{BOD}. The reset is kept active until V_{CC} again rises above the trigger level.

Figure 18. Brown-out Reset During Operation



The BOD circuit will not detect a drop in V_{CC} unless the voltage stays below the trigger level for the detection time, t_{BOD} (see "System and Reset Characteristics" on page 250).

The BOD circuit has three modes of operation:

- **Disabled:** In this mode of operation V_{CC} is not monitored and, hence, it is recommended only for applications where the power supply remains stable.
- **Enabled:** In this mode the V_{CC} level is continuously monitored. If V_{CC} drops below V_{BOT} for at least t_{BOD} a brown-out reset will be generated.
- **Sampled:** In this mode the V_{CC} level is sampled on each negative edge of a 1kHz clock that has been derived from the 32kHz ULP oscillator. Between each sample the BOD is turned off. Compared to the mode where BOD is constantly enabled this mode of operation reduces power consumption but fails to detect drops in V_{CC} between two positive edges of the 1kHz clock. When a brown-out is detected in this mode, the BOD circuit is set to enabled mode to ensure that the device is kept in reset until V_{CC} has risen above V_{BOT} . The BOD will return to sampled mode after reset has been released and the fuses have been read in.

The BOD mode of operation is selected using BODACT and BODPD fuse bits. The BODACT fuse bits determine how the BOD operates in active and idle mode, as shown in Table 11.

Table 11. Setting BOD Mode of Operation in Active and Idle Modes

BODACT1	BODACT0	Mode of Operation
0	0	Reserved
0	1	Sampled
1	0	Enabled
1	1	Disabled

The BODPD fuse bits determine the mode of operation in all sleep modes except idle mode, as shown in Table 12.

Table 12. Setting BOD Mode of Operation in Sleep Modes Other Than Idle

BODPD1	BODPD0	Mode of Operation
0	0	Reserved
0	1	Sampled
1	0	Enabled
1	1	Disabled

See “Fuse Bits” on page 226.

8.3 Internal Voltage Reference

ATtiny828 features an internal bandgap reference. This reference is used for Brown-out Detection, and it can be used as an input to the Analog Comparator or the ADC. The bandgap voltage varies with supply voltage and temperature.

8.3.1 Voltage Reference Enable Signals and Start-up Time

The voltage reference has a start-up time that may influence the way it should be used. The start-up time is given in “System and Reset Characteristics” on page 250. To save power, the reference is not always turned on. The reference is on during the following situations:

- When the BOD is enabled.
- When the internal reference is connected to the Analog Comparator.
- When the ADC is enabled.

Thus, when the BOD is not enabled, after setting the ACBG bit or enabling the ADC, the user must always allow the reference to start up before the output from the Analog Comparator or ADC is used. To reduce power consumption in Power-down mode, the user can avoid the three conditions above to ensure that the reference is turned off before entering Power-down mode.

8.4 Watchdog Timer

The Watchdog Timer is clocked from the internal 32kHz ultra low power oscillator (see page 29). By controlling the Watchdog Timer prescaler, the Watchdog Reset interval can be adjusted as shown in Table 15 on page 47. The WDR – Watchdog Reset – instruction resets the Watchdog Timer. The Watchdog Timer is also reset when it is disabled and when a Chip Reset occurs. Ten different clock cycle periods can be selected to determine the reset period. If the reset period expires without another Watchdog Reset, the ATtiny828 resets and executes from the Reset Vector.

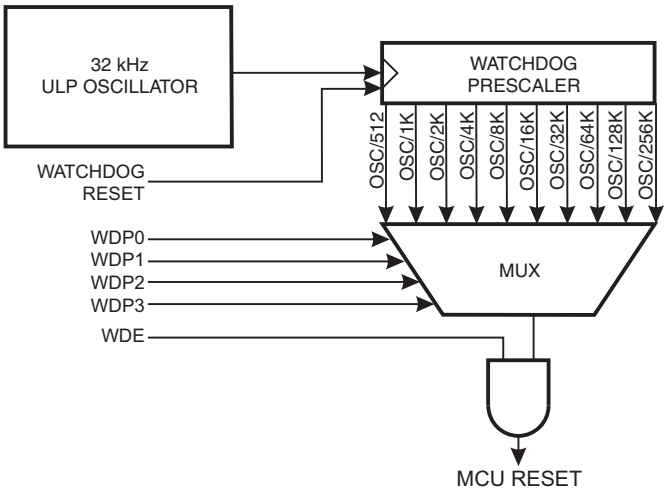
The Wathdog Timer can also be configured to generate an interrupt instead of a reset. This can be very helpful when using the Watchdog to wake-up from Power-down.

To prevent unintentional disabling of the Watchdog or unintentional change of time-out period, two different safety levels are selected by the fuse WDTON as shown in Table 13 See “Timed Sequences for Changing the Configuration of the Watchdog Timer” on page 44 for details.

Table 13. WDT Configuration as a Function of the Fuse Settings of WDTON

WDTON	Safety Level	WDT Initial State	How to Disable the WDT	How to Change Time-out
Unprogrammed	1	Disabled	Timed sequence	No limitations
Programmed	2	Enabled	Always enabled	Timed sequence

Figure 19. Watchdog Timer



8.4.1 Timed Sequences for Changing the Configuration of the Watchdog Timer

The sequence for changing configuration differs slightly between the two safety levels. Separate procedures are described for each level.

- Safety Level 1

  In this mode, the Watchdog Timer is initially disabled, but can be enabled by writing the WDE bit to one without any restriction. A timed sequence is needed when disabling an enabled Watchdog Timer. To disable an enabled Watchdog Timer, the following procedure must be followed:

  1. Write the signature for change enable of protected I/O registers to register CCP
  2. Within four instruction cycles, in the same operation, write WDE and WDP bits

- Safety Level 2

  In this mode, the Watchdog Timer is always enabled, and the WDE bit will always read as one. A timed sequence is needed when changing the Watchdog Time-out period. To change the Watchdog Time-out, the following procedure must be followed:

  1. Write the signature for change enable of protected I/O registers to register CCP
  2. Within four instruction cycles, write the WDP bit. The value written to WDE is irrelevant

8.4.2 Code Examples

The following code example shows how to turn off the WDT. The example assumes that interrupts are controlled (e.g., by disabling interrupts globally) so that no interrupts will occur during execution of these functions.

Assembly Code Example

```
WDT_off:
      wdr

      ; Clear WDRF in MCUSR
      in            r16, MCUSR
      andi  r16, ~(1<<WDRF)
      out               MCUSR, r16

      ; Write signature for change enable of protected I/O register
      ldi           r16, 0xD8
      out           CCP, r16

      ; Within four instruction cycles, turn off WDT
      ldi           r16, (0<<WDE)
      out           WDTCSR, r16
      ret
```

Note: See “Code Examples” on page 7.

8.5 Register Description

8.5.1 MCUSR – MCU Status Register

The MCU Status Register provides information on which reset source caused an MCU Reset.

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	–	–	–	–	WDRF	BORF	EXTRF	PORF	MCUSR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	See Bit Description				

- **Bits 7:4 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 3 – WDRF: Watchdog Reset Flag**

This bit is set if a Watchdog Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

- **Bit 2 – BORF: Brown-out Reset Flag**

This bit is set if a Brown-out Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

- **Bit 1 – EXTRF: External Reset Flag**

This bit is set if an External Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

- **Bit 0 – PORF: Power-on Reset Flag**

This bit is set if a Power-on Reset occurs. The bit is reset only by writing a logic zero to the flag.

To make use of the Reset Flags to identify a reset condition, the user should read and then reset the MCUSR as early as possible in the program. If the register is cleared before another reset occurs, the source of the reset can be found by examining the Reset Flags.

8.5.2 WDTCSR – Watchdog Timer Control and Status Register

Bit	7	6	5	4	3	2	1	0	
(0x60)	**WDIF**	**WDIE**	**WDP3**	**–**	**WDE**	**WDP2**	**WDP1**	**WDP0**	**WDTCSR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	0	0	0	

- **Bit 7 – WDIF: Watchdog Timeout Interrupt Flag**

This bit is set when a time-out occurs in the Watchdog Timer and the Watchdog Timer is configured for interrupt. WDIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, WDIF is cleared by writing a logic one to the flag. When the I-bit in SREG and WDIE are set, the Watchdog Time-out Interrupt is executed.

- **Bit 6 – WDIE: Watchdog Timeout Interrupt Enable**

When this bit is written to one, WDE is cleared, and the I-bit in the Status Register is set, the Watchdog Time-out Interrupt is enabled. In this mode the corresponding interrupt is executed instead of a reset if a timeout in the Watchdog Timer occurs.

If WDE is set, WDIE is automatically cleared by hardware when a time-out occurs. This is useful for keeping the Watchdog Reset security while using the interrupt. After the WDIE bit is cleared, the next time-out will generate a reset. To avoid the Watchdog Reset, WDIE must be set after each interrupt.

Table 14. Watchdog Timer Configuration

WDE	WDIE	Watchdog Timer State	Action on Time-out
0	0	Stopped	None
0	1	Running	Interrupt
1	0	Running	Reset
1	1	Running	Interrupt

- **Bit 4 – Res: Reserved**

This bit is reserved and will always read zero.

- **Bit 3 – WDE: Watchdog Enable**

When the WDE is written to logic one, the Watchdog Timer is enabled, and if the WDE is written to logic zero, the Watchdog Timer function is disabled.

In safety level 2, it is not possible to disable the Watchdog Timer, even with the algorithm described above. See “Timed Sequences for Changing the Configuration of the Watchdog Timer” on page 44.

In safety level 1, WDE is overridden by WDRF in MCUSR. See “MCUSR – MCU Status Register” on page 45 for description of WDRF. This means that WDE is always set when WDRF is set. To clear WDE, WDRF must be cleared before disabling the Watchdog with the procedure described above. This feature ensures multiple resets during conditions causing failure, and a safe start-up after the failure.

If the watchdog timer is not going to be used in the application, it is important to go through a watchdog disable procedure in the initialization of the device. If the Watchdog is accidentally enabled, for example by a runaway pointer or brown-out condition, the device will be reset, which in turn will lead to a new watchdog reset. To avoid this situation, the application software should always clear the WDRF flag and the WDE control bit in the initialization routine.

- **Bits 5, 2:0 – WDP[3:0]: Watchdog Timer Prescaler 3 - 0**

The WDP[3:0] bits determine the Watchdog Timer prescaling when the Watchdog Timer is enabled. The different prescaling values and their corresponding Timeout Periods are shown in Table 15.

Table 15. Watchdog Timer Prescale Select

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 5.0V
0	0	0	0	512 cycles	16 ms
0	0	0	1	1K cycles	32 ms
0	0	1	0	2K cycles	64 ms
0	0	1	1	4K cycles	0.125 s
0	1	0	0	8K cycles	0.25 s
0	1	0	1	16K cycles	0.5 s
0	1	1	0	32K cycles	1.0 s
0	1	1	1	64K cycles	2.0 s
1	0	0	0	128K cycles	4.0 s
1	0	0	1	256K cycles	8.0 s
1	0	1	0	Reserved(1)	
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

Note: 1. If selected, one of the valid settings below 0b1010 will be used.

To avoid unintentional changes of these bits, the following sequence must be followed:

1. Write the required signature to the CCP register. See page 14.
2. Within four instruction cycles, write the desired bit value.

9. Interrupts

This section describes the specifics of the interrupt handling. For a general explanation of the AVR interrupt handling, see “Reset and Interrupt Handling” on page 12.

9.1 Interrupt Vectors

The interrupt vectors of ATtiny828 are described in Table 16 below.

Table 16. Reset and Interrupt Vectors

Vector No.	Program Address (1)	Label	Interrupt Source
1	0x0000 (2)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset
2	0x0001	INT0	External Interrupt Request 0
3	0x0002	INT1	External Interrupt Request 1
4	0x0003	PCINT0	Pin Change Interrupt Request 0
5	0x0004	PCINT1	Pin Change Interrupt Request 1
6	0x0005	PCINT2	Pin Change Interrupt Request 2
7	0x0006	PCINT3	Pin Change Interrupt Request 3
8	0x0007	WDT	Watchdog Time-out
9	0x0008	TIM1_CAPT	Timer/Counter1 Input Capture
10	0x0009	TIM1_COMPA	Timer/Counter1 Compare Match A
11	0x000A	TIM1_COMPB	Timer/Counter1 Compare Match B
12	0x000B	TIM1_OVF	Timer/Counter1 Overflow
13	0x000C	TIM0_COMPA	Timer/Counter0 Compare Match A
14	0x000D	TIM0_COMPB	Timer/Counter0 Compare Match B
15	0x000E	TIM0_OVF	Timer/Counter0 Overflow
16	0x000F	SPI	SPI Serial Transfer Complete
17	0x0010	USART0_RXS	USART0 Rx Start
18	0x0011	USART0_RXC	USART0 Rx Complete
19	0x0012	USART0_DRE	USART0 Data Register Empty
20	0x0013	USART0_TXC	USART0 Tx Complete
21	0x0014	ADC_READY	ADC Conversion Complete
22	0x0015	EE_RDY	EEPROM Ready
23	0x0016	ANA_COMP	Analog Comparator
24	0x0017	TWI	Two-Wire Interface
25	0x0018	SPM_RDY	Store Program Memory Ready
26	0x0019	RESERVED	Reserved

Note: 1. When the IVSEL bit in MCUCR is set, interrupt vectors are moved to the start of the Flash boot section. In this case, the address of each interrupt vector will be the address in this table added to the start address of the Flash boot section.

2. When the BOOTRST fuse is programmed, the device will jump to the boot loader address at reset. See “Entering the Boot Loader Program” on page 216.

In case the program never enables an interrupt source, the interrupt vectors will not be used and, consequently, 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.

A typical and general setup for interrupt vector addresses in ATtiny828 is shown in the program example below.

Assembly Code Example

```
      .org 0x0000                                ; Set address of next
      statement

            rjmp            RESET                ; Address 0x0000
            rjmp            INT0_ISR             ; Address 0x0001
            rjmp            INT1_ISR             ; Address 0x0002
            rjmp            PCINT0_ISR           ; Address 0x0003
            rjmp            PCINT1_ISR           ; Address 0x0004
            rjmp            PCINT2_ISR           ; Address 0x0005
            rjmp            PCINT3_ISR           ; Address 0x0006
            rjmp            WDT_ISR              ; Address 0x0007
            rjmp            TIM1_CAPT_ISR        ; Address 0x0008
            rjmp            TIM1_COMPA_ISR       ; Address 0x0009
            rjmp            TIM1_COMPB_ISR       ; Address 0x000A
            rjmp            TIM1_OVF_ISR         ; Address 0x000B
            rjmp            TIM0_COMPA_ISR       ; Address 0x000C
            rjmp            TIM0_COMPB_ISR       ; Address 0x000D
            rjmp            TIM0_OVF_ISR         ; Address 0x000E
            rjmp            SPI_ISR              ; Address 0x000F
            rjmp            USART0_RXS_ISR       ; Address 0x0010
            rjmp            USART0_RXC_ISR       ; Address 0x0011
            rjmp            USART0_DRE_ISR       ; Address 0x0012
            rjmp            USART0_TXC_ISR       ; Address 0x0013
            rjmp            ADC_ISR              ; Address 0x0014
            rjmp            EE_RDY_ISR           ; Address 0x0015
            rjmp            ANA_COMP_ISR         ; Address 0x0016
            rjmp            TWI_ISR              ; Address 0x0017
            rjmp            SPM_RDY_ISR          ; Address 0x0018
            rjmp            RESERVED             ; Address 0x0019

      RESET:                                     ; Main program start
            <instr>                              ; Address 0x001A
            ...
```

Note: See “Code Examples” on page 7.

Table 17 shows reset and interrupt vector placement for combinations of BOOTRST and IVSEL settings.

Table 17. Reset and Interrupt Vector Placement

BOOTRST [1]	IVSEL	Reset Address	Start of Interrupt Vector Table
1	0	0x000	0x001
1	1	0x000	Boot reset address [2] + 0x001
0	0	Boot reset address [2]	0x001
0	1	Boot reset address [2]	Boot reset address [2] + 0x001

Note: 1. For the BOOTRST fuse “1” means unprogrammed while “0” means programmed.
2. The boot reset address is shown in Table 82 on page 217.

The following is a program example for the case where:

- The BOOTRST fuse is unprogrammed
- Boot section size set to 2K bytes
- The IVSEL bit in MCUCR is set before any interrupts are enabled

Assembly Code Example

```
      .org 0x0000               ; Set address of next statement

      RESET:                    ; Main program start
            <instr>             ; Address 0x0000
            ...
      .org 0x0C01               ; Set address of next statement

            rjmp   INT0_ISR     ; Address 0x0C01
            rjmp   INT1_ISR     ; Address 0x0C02
            ...
            rjmp   SPM_RDY_ISR  ; Address 0x0C18
            rjmp   RESERVED     ; Address 0x0C19
```

Note: See “Code Examples” on page 7.

The following is a program example for the case where:

- The BOOTRST fuse is programmed
- Boot section size set to 2K bytes

Assembly Code Example

```
.org 0x0001                    ; Set address of next statement

      rjmp   INT0_ISR          ; Address 0x0001
      rjmp   INT1_ISR          ; Address 0x0002
      ...
      rjmp   SPM_RDY_ISR       ; Address 0x0018
      rjmp   RESERVED          ; Address 0x0019

.org 0x0C00              ; Set address of next statement

RESET:                   ; Main program start
      <instr>            ; Address 0x0C00
      ...
```

Note: See “Code Examples” on page 7.

The following is a program example for the case where:

- The BOOTRST fuse is programmed
- Boot section size set to 2K bytes
- The IVSEL bit in MCUCR is set before any interrupts are enabled

Assembly Code Example

```
.org 0x0C00                    ; Set address of next statement

      rjmp   RESET             ; Address 0x0C00
      rjmp   INT0_ISR          ; Address 0x0C01
      rjmp   INT1_ISR          ; Address 0x0C02
      ...
      rjmp   SPM_RDY_ISR       ; Address 0x0C18
      rjmp   RESERVED          ; Address 0x0C19

RESET:                         ; Main program start
      <instr>                  ; Address 0x0C1A
      ...
```

9.2 External Interrupts

External Interrupts are triggered by the INT0 and INT1 pins, or by any of the PCINTn pins. Note that, if enabled, the interrupts will trigger even if the INTn or PCINTn pins are configured as outputs. This feature provides a way of generating software interrupts.

The pin change interrupts trigger as follows:

- Pin Change Interrupt 0 (PCI0): triggers if any enabled PCINT[7:0] pin toggles
- Pin Change Interrupt 1 (PCI1): triggers if any enabled PCINT[15:8] pin toggles
- Pin Change Interrupt 2 (PCI2): triggers if any enabled PCINT[23:16] pin toggles
- Pin Change Interrupt 3(PCI3): triggers if any enabled PCINT[27:24] pin toggles

Registers PCMSK0, PCMSK1, PCMSK2, and PCMSK3 control which pins contribute to the pin change interrupts.

Pin change interrupts on PCINT[27:0] are detected asynchronously, which means that these interrupts can be used for waking the part also from sleep modes other than Idle mode.

In order for a pin change interrupt (PCINT) to be generated, the device must have an active I/O clock. As shown in Table 9 on page 34, the I/O clock domain is active in Idle Mode, but not in deeper sleep modes. In sleep modes deeper than Idle Mode, a toggled pin must remain in its toggled state until the device has fully woken up. See Table 7 on page 30 for wake up times. If the pin toggles back to its initial state during wake up the device will still complete the procedure but will not generate an interrupt once awake.

External interrupts INT0 and INT1 can be triggered by a falling or rising edge, or a low level. When INT0 or INT1 is enabled and configured as level triggered, the interrupt will trigger as long as the pin is held low.

Note that recognition of falling or rising edge interrupts on INT0 and INT1 requires the presence of an I/O clock, as described in “Clock System” on page 27.

9.2.1 Low Level Interrupt

A low level interrupt on INT0 or INT1 is detected asynchronously. This means that the interrupt source can be used for waking the part also from sleep modes other than Idle (the I/O clock is halted in all sleep modes except Idle).

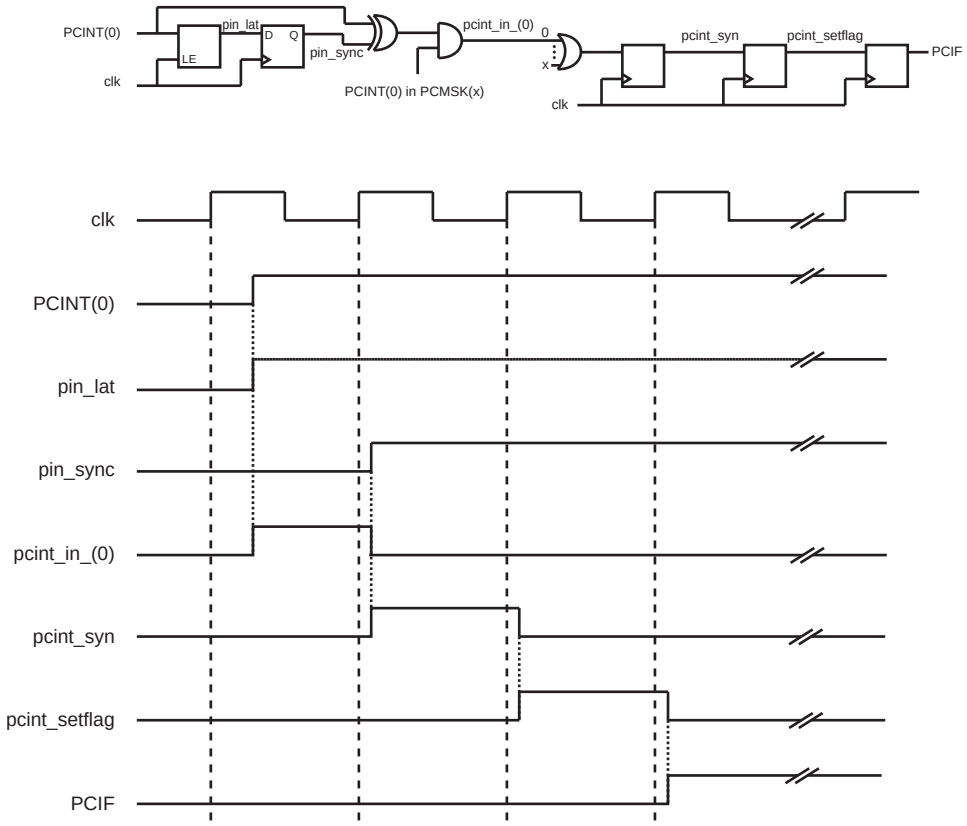
Note that if a level triggered interrupt is used for wake-up from Power-down, the required level must be held long enough for the MCU to complete the wake-up to trigger the level interrupt. If the level disappears before the end of the Start-up Time, the MCU will still wake up, but no interrupt will be generated. The start-up time is defined as described in “Clock System” on page 27.

If the low level on the interrupt pin is removed before the device has woken up then program execution will not be diverted to the interrupt service routine but continue from the instruction following the SLEEP command.

9.2.2 Pin Change Interrupt Timing

A timing example of a pin change interrupt is shown in Figure 20.

Figure 20. Timing of pin change interrupts



9.3 Register Description

9.3.1 MCUCR – MCU Control Register

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

- **Bits 7:2, 0 – Res: Reserved Bits**

These bits are reserved and will always read zero.

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

When this bit is cleared, interrupt vectors are placed at the start of Flash memory. When this bit is set, interrupt vectors are moved to the beginning of the boot loader section.

The start address of the boot section is determined by the BOOTSZ Fuses. See “Configuring the Boot Loader” on page 216 for details.

If interrupt vectors are placed in the boot loader section and boot lock bit BLB02 is programmed, interrupts will be disabled while executing from the application section.

If interrupt vectors are placed in the application section and boot lock bit BLB12 is programmed, interrupts will be disabled while executing from the boot loader section.

To avoid unintentional changes to this bit, the following sequence must be followed:

1. Write the required signature to the CCP register. See page 14.
2. Within four instruction cycles, write the desired value to IVSEL.

9.3.2 PCMSK3 – Pin Change Mask Register 3

Bit	7	6	5	4	3	2	1	0	
(0x73)	–	–	–	–	PCINT27	PCINT26	PCINT25	PCINT24	PCMSK3
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 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bits 3:0 – PCINT[27:24] : Pin Change Interrupt Mask Bits**

Each PCINTn bit selects if the pin change interrupt of the corresponding I/O pin is enabled. Pin change interrupt on a pin is enabled by setting the mask bit for the pin (PCINTn) and the corresponding group bit (PCIEn) in PCICR.

When this bit is cleared the pin change interrupt on the corresponding pin is disabled.

9.3.3 PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – PCINT[23:16] : Pin Change Interrupt Mask Bits**

Each PCINTn bit selects if the pin change interrupt of the corresponding I/O pin is enabled. Pin change interrupt on a pin is enabled by setting the mask bit for the pin (PCINTn) and the corresponding group bit (PCIEn) in PCICR.

When this bit is cleared the pin change interrupt on the corresponding pin is disabled.

9.3.4 PCMSK1 – Pin Change Mask Register 1

Bit	7	6	5	4	3	2	1	0	
(0x6C)	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – PCINT[15:8] : Pin Change Interrupt Mask Bits**

Each PCINTn bit selects if the pin change interrupt of the corresponding I/O pin is enabled. Pin change interrupt on a pin is enabled by setting the mask bit for the pin (PCINTn) and the corresponding group bit (PCIEn) in PCICR.

When this bit is cleared the pin change interrupt on the corresponding pin is disabled.

9.3.5 PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – PCINT[7:0] : Pin Change Interrupt Mask Bits**

Each PCINTn bit selects if the pin change interrupt of the corresponding I/O pin is enabled. Pin change interrupt on a pin is enabled by setting the mask bit for the pin (PCINTn) and the corresponding group bit (PCIEn) in PCICR.

When this bit is cleared the pin change interrupt on the corresponding pin is disabled.

9.3.6 EICRA – External Interrupt Control Register A

The External Interrupt Control Register contains bits for controlling external interrupt sensing and power management.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
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 3:2 – ISC11, ISC10: Interrupt Sense Control, INT1**
- **Bits 1:0 – ISC01, ISC00: Interrupt Sense Control, INT0**

External interrupts INT0 and INT1 are triggered by activity on pin INT0 and INT1, provided that the SREG I-flag and the corresponding interrupt mask are set. The conditions required to trigger the interrupt are defined in Table 18.

Table 18. External Interrupt Sense Control

ISCn1	ISCn0	Description
0	0	The low level of INT0/INT1 generates an interrupt request (1)
0	1	Any logical change on INT0/INT1 generates an interrupt request (2)
1	0	The falling edge of INT0/INT1 generates an interrupt request (2)
1	1	The rising edge of INT0/INT1 generates an interrupt request (2)

Note: 1. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

2. The value on the INT0/INT1 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt.

9.3.7 PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	PCIE3	PCIE2	PCIE1	PCIE0	PCICR
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 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 3 – PCIE3: Pin Change Interrupt Enable 3**

When this bit and the I-bit of SREG are set the Pin Change Interrupt 3 is enabled. Any change on an enabled PCINT[27:24] pin will cause a PCINT3 interrupt. See Table 17 on page 50.

Each pin can be individually enabled. See “PCMSK3 – Pin Change Mask Register 3” on page 54.

- **Bit 2 – PCIE2: Pin Change Interrupt Enable 2**

When this bit and the I-bit of SREG are set the Pin Change Interrupt 2 is enabled. Any change on an enabled PCINT[23:16] pin will cause a PCINT2 interrupt. See Table 17 on page 50.

Each pin can be individually enabled. See “PCMSK2 – Pin Change Mask Register 2” on page 54.

- **Bit 1 – PCIE1: Pin Change Interrupt Enable 1**

When this bit and the I-bit of SREG are set the Pin Change Interrupt 1 is enabled. Any change on an enabled PCINT[15:8] pin will cause a PCINT1 interrupt. See Table 17 on page 50.

Each pin can be individually enabled. See “PCMSK1 – Pin Change Mask Register 1” on page 54.

- **Bit 0 – PCIE0: Pin Change Interrupt Enable 0**

When this bit and the I-bit of SREG are set the Pin Change Interrupt 0 is enabled. Any change on an enabled PCINT[7:0] pin will cause a PCINT0 interrupt. See Table 17 on page 50.

Each pin can be individually enabled. See “PCMSK0 – Pin Change Mask Register 0” on page 55.

9.3.8 EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:2 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 1 – INT1: External Interrupt Request 1 Enable**

The external interrupt for pin INT1 is enabled when this bit and the I-bit in the Status Register (SREG) are set. The trigger conditions are set with the ISC1n bits.

Activity on the pin will cause an interrupt request even if INT0 has been configured as an output.

- **Bit 0 – INT0: External Interrupt Request 0 Enable**

The external interrupt for pin INT0 is enabled when this bit and the I-bit in the Status Register (SREG) are set. The trigger conditions are set with the ISC0n bits.

Activity on the pin will cause an interrupt request even if INT1 has been configured as an output.

9.3.9 EIFR – External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	–	–	–	–	–	–	**INTF1**	**INTF0**	**EIFR**
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:2 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 1 – INTF1: External Interrupt Flag 0**

This bit is set when activity on INT1 has triggered an interrupt request. Provided that the I-bit in SREG and the INT1 bit in EIMSK are set, the MCU will jump to the corresponding interrupt vector.

The flag is cleared when the interrupt service routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

This flag is always cleared when INT1 is configured as a level interrupt.

- **Bit 0 – INTF0: External Interrupt Flag 0**

This bit is set when activity on INT0 has triggered an interrupt request. Provided that the I-bit in SREG and the INT0 bit in EIMSK are set, the MCU will jump to the corresponding interrupt vector.

The flag is cleared when the interrupt service routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

This flag is always cleared when INT0 is configured as a level interrupt.

9.3.10 PCIFR – Pin Change Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)	–	–	–	–	**PCIF3**	**PCIF2**	**PCIF1**	**PCIF0**	**PCIFR**
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 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 3 – PCIF3: Pin Change Interrupt Flag 3**

This bit is set when a logic change on any PCINT[27:24] pin has triggered an interrupt request. Provided that the I-bit in SREG and the PCIE3 bit in PCICR are set, the MCU will jump to the corresponding interrupt vector.

The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

- **Bit 2 – PCIF2: Pin Change Interrupt Flag 2**

This bit is set when a logic change on any PCINT[23:16] pin has triggered an interrupt request. Provided that the I-bit in SREG and the PCIE2 bit in PCICR are set, the MCU will jump to the corresponding interrupt vector.

The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

- **Bit 1 – PCIF1: Pin Change Interrupt Flag 1**

This bit is set when a logic change on any PCINT[15:8] pin has triggered an interrupt request. Provided that the I-bit in SREG and the PCIE1 bit in PCICR are set, the MCU will jump to the corresponding interrupt vector.

The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

- **Bit 0 – PCIF0: Pin Change Interrupt Flag 0**

This bit is set when a logic change on any PCINT[7:0] pin has triggered an interrupt request. Provided that the I-bit in SREG and the PCIE0 bit in PCICR are set, the MCU will jump to the corresponding interrupt vector.

The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

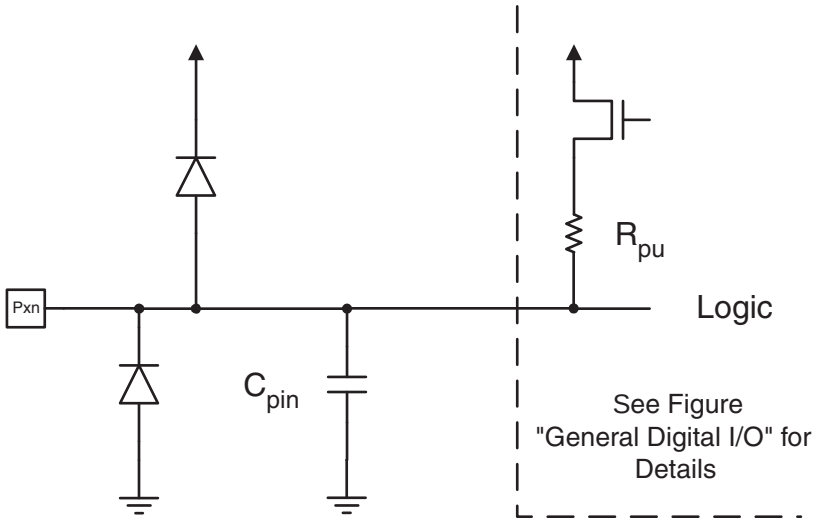
10. I/O Ports

10.1 Overview

All AVR ports have true Read-Modify-Write functionality when used as general digital I/O ports. This means that the direction of one port pin can be changed without unintentionally changing the direction of any other pin with the SBI and CBI instructions. The same applies when changing drive value (if configured as output) or enabling/disabling of pull-up resistors.

The pin driver is strong enough to drive LED displays directly. All port pins have individually selectable pull-up resistors with a supply-voltage invariant resistance. All I/O pins have protection diodes to both V_{CC} and Ground as indicated in Figure 21 on page 59. See "Electrical Characteristics" on page 247 for a complete list of parameters.

Figure 21. I/O Pin Equivalent Schematic



All registers and bit references in this section are written in general form. A lower case "x" represents the numbering letter for the port, and a lower case "n" represents the bit number. However, when using the register or bit defines in a program, the precise form must be used. For example, PORTB3 for bit no. 3 in Port B, here documented generally as PORTxn. The physical I/O Registers and bit locations are listed in "Register Description" on page 81.

Four I/O memory address locations are allocated for each port, one each for the Data Register – PORTx, Data Direction Register – DDRx, Pull-up Enable Register – PUEx, and the Port Input Pins – PINx. The Port Input Pins I/O location is read only, while the Data Register, the Data Direction Register, and the Pull-up Enable Register are read/write. However, writing a logic one to a bit in the PINx Register, will result in a toggle in the corresponding bit in the Data Register.

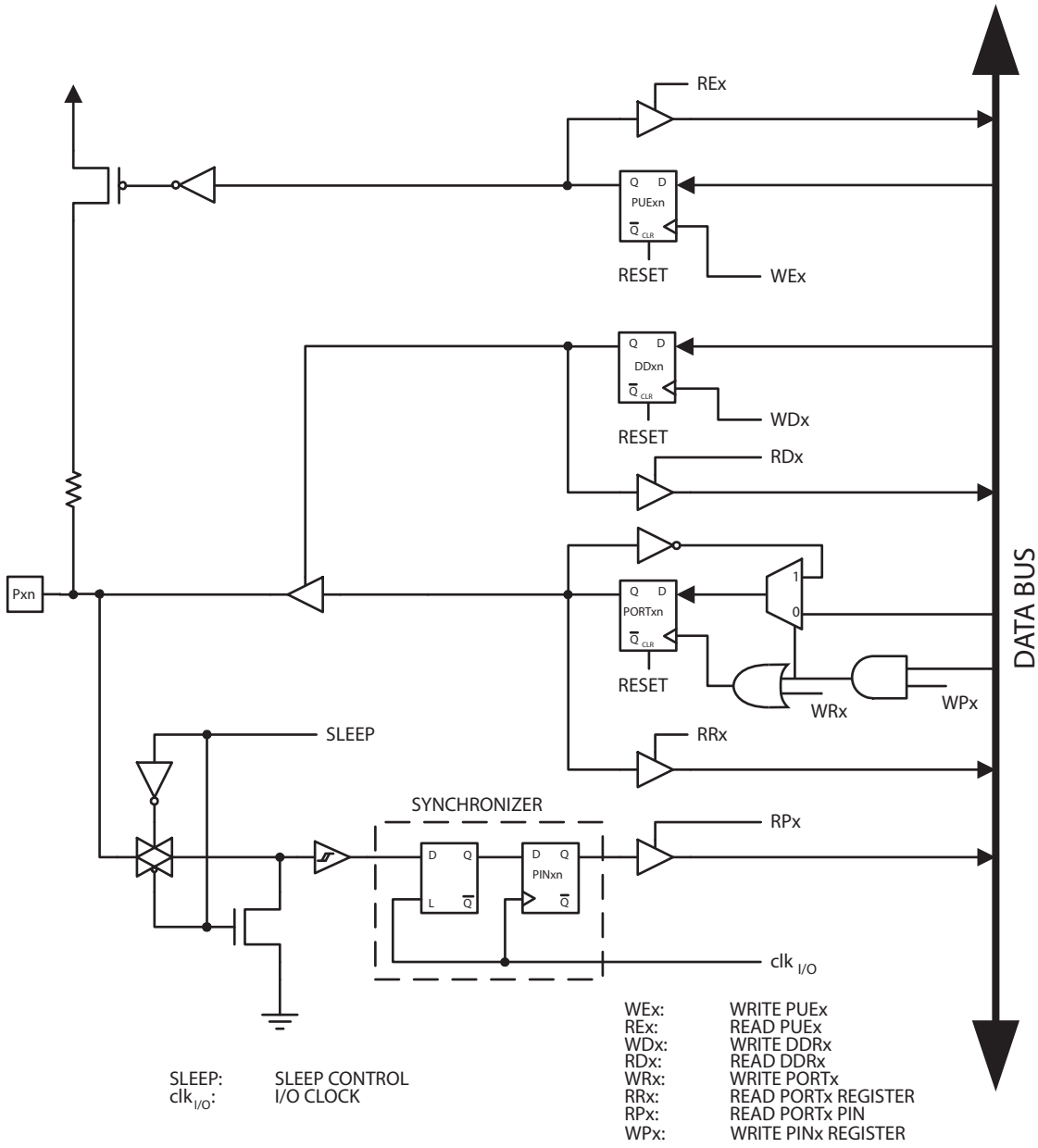
Using the I/O port as General Digital I/O is described in "Ports as General Digital I/O" on page 59. Most port pins are multiplexed with alternative functions for the peripheral features on the device. How each alternative function interferes with the port pin is described in "Alternative Port Functions" on page 63. Refer to the individual module sections for a full description of the alternative functions.

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

10.2 Ports as General Digital I/O

The ports are bi-directional I/O ports with optional internal pull-ups. Figure 22 shows a functional description of one I/O-port pin, here generically called Pxn.

Figure 22. General Digital I/O[1]



Note: 1. WEx, WRx, WPx, WDx, REx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, and SLEEP are common to all ports.

10.2.1 Configuring the Pin

Each port pin consists of four register bits: DDxn, PORTxn, PUExn, and PINxn. As shown in “Register Description” on page 81, the DDxn bits are accessed at the DDRx I/O address, the PORTxn bits at the PORTx I/O address, the PUExn bits at the PUEx I/O address, and the PINxn bits at the PINx I/O address.

The DDxn bit in the DDRx Register selects the direction of this pin. If DDxn is written logic one, Pxn is configured as an output pin. If DDxn is written logic zero, Pxn is configured as an input pin.

If PORTxn is written logic one when the pin is configured as an output pin, the port pin is driven high (one). If PORTxn is written logic zero when the pin is configured as an output pin, the port pin is driven low (zero).

The pull-up resistor is activated, if the PUExn is written logic one. To switch the pull-up resistor off, PUExn has to be written logic zero.

Table 19 summarizes the control signals for the pin value.

Table 19. Port Pin Configurations

DDxn	PORTxn	PUExn	I/O	Pull-up	Comment
0	X	0	Input	No	Tri-state (hi-Z)
0	X	1	Input	Yes	Sources current if pulled low externally
1	0	0	Output	No	Output low (sink)
1	0	1	Output	Yes	NOT RECOMMENDED. Output low (sink) and internal pull-up active. Sources current through the internal pull-up resistor and consumes power constantly
1	1	0	Output	No	Output high (source)
1	1	1	Output	Yes	Output high (source) and internal pull-up active

Port pins are tri-stated when a reset condition becomes active, even when no clocks are running.

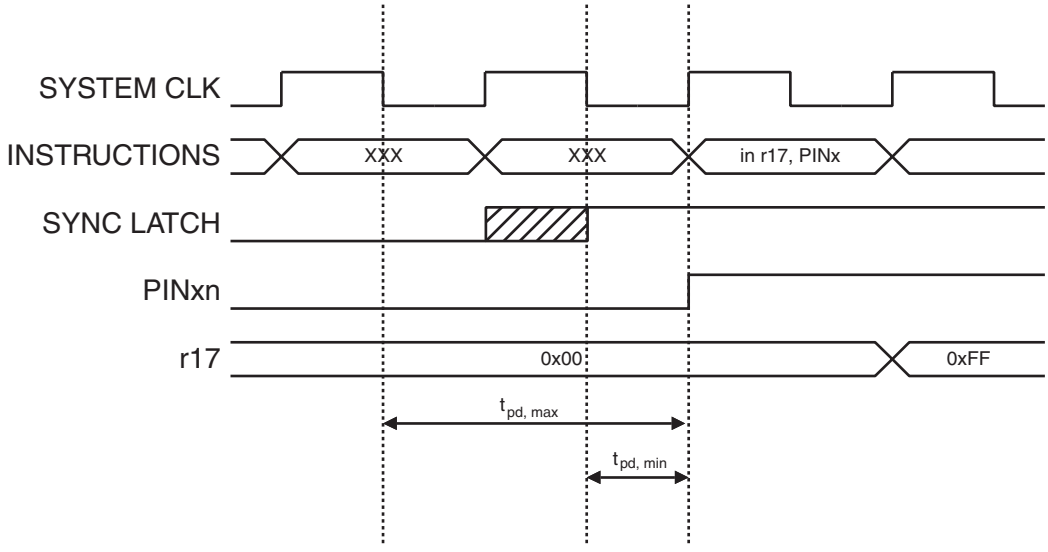
10.2.2 Toggling the Pin

Writing a logic one to PINxn toggles the value of PORTxn, independent on the value of DDRxn. Note that the SBI instruction can be used to toggle one single bit in a port.

10.2.3 Reading the Pin Value

Independent of the setting of Data Direction bit DDxn, the port pin can be read through the PINxn Register bit. As shown in Figure 22 on page 60, the PINxn Register bit and the preceding latch constitute a synchronizer. This is needed to avoid metastability if the physical pin changes value near the edge of the internal clock, but it also introduces a delay. Figure 23 shows a timing diagram of the synchronization when reading an externally applied pin value. The maximum and minimum propagation delays are denoted $t_{pd,max}$ and $t_{pd,min}$ respectively.

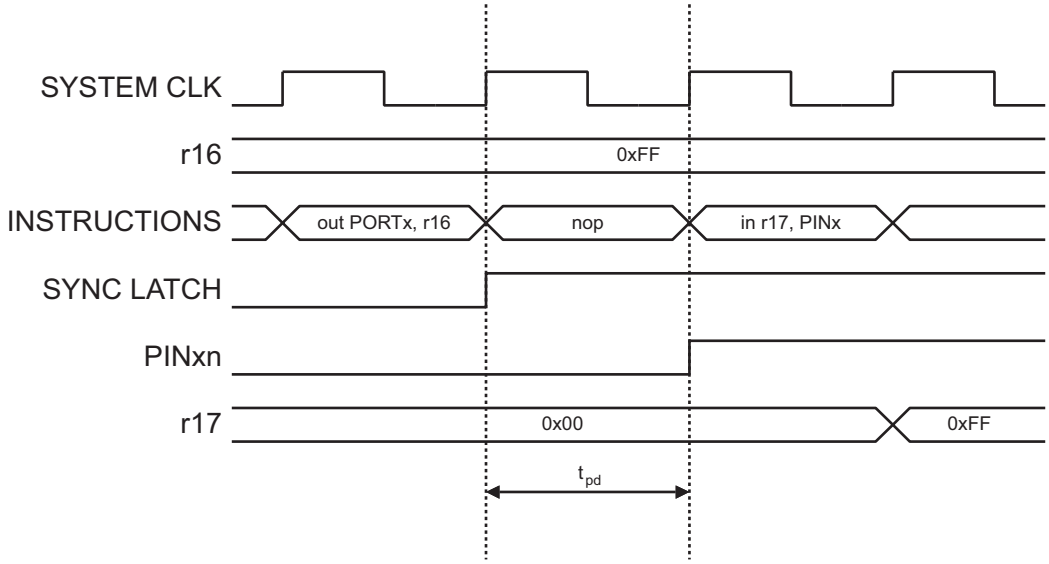
Figure 23. Synchronization when Reading an Externally Applied Pin value



Consider the clock period starting shortly after the first falling edge of the system clock. The latch is closed when the clock is low, and goes transparent when the clock is high, as indicated by the shaded region of the “SYNC LATCH” signal. The signal value is latched when the system clock goes low. It is clocked into the PINxn Register at the succeeding positive clock edge. As indicated by the two arrows tpd,max and tpd,min, a single signal transition on the pin will be delayed between ½ and 1½ system clock period depending upon the time of assertion.

When reading back a software assigned pin value, a nop instruction must be inserted as indicated in Figure 24 on page 62. The out instruction sets the “SYNC LATCH” signal at the positive edge of the clock. In this case, the delay tpd through the synchronizer is one system clock period.

Figure 24. Synchronization when Reading a Software Assigned Pin Value



10.2.4 Digital Input Enable and Sleep Modes

As shown in Figure 22 on page 60, the digital input signal can be clamped to ground at the input of the schmitt-trigger. The signal denoted SLEEP in the figure, is set by the MCU sleep controller in Power-down and Standby modes to avoid high power consumption if some input signals are left floating, or have an analog signal level close to $V_{CC}/2$.

SLEEP is overridden for port pins enabled as external interrupt pins. If the external interrupt request is not enabled, SLEEP is active also for these pins. SLEEP is also overridden by various other alternative functions as described in “Alternative Port Functions” on page 63.

If a logic high level (“one”) is present on an asynchronous external interrupt pin configured as “Interrupt on Rising Edge, Falling Edge, or Any Logic Change on Pin” while the external interrupt is *not* enabled, the corresponding External Interrupt Flag will be set when resuming from the above mentioned Sleep mode, as the clamping in these sleep mode produces the requested logic change.

10.2.5 Unconnected Pins

If some pins are unused, it is recommended to ensure that these pins have a defined level. Even though most of the digital inputs are disabled in the deep sleep modes as described above, floating inputs should be avoided to reduce current consumption in all other modes where the digital inputs are enabled (Reset, Active mode and Idle mode).

The simplest method to ensure a defined level of an unused pin, is to enable the internal pull-up. In this case, the pull-up will be disabled during reset. If low power consumption during reset is important, it is recommended to use an external pull-up or pulldown. Connecting unused pins directly to V_{CC} or GND is not recommended, since this may cause excessive currents if the pin is accidentally configured as an output.

10.2.6 Program Example

The following code example shows how to set port B pin 0 high, pin 1 low, and define the port pins from 2 to 3 as input with a pull-up assigned to port pin 2. The resulting pin values are read back again, but as previously discussed, a *nop* instruction is included to be able to read back the value recently assigned to some of the pins.

Assembly Code Example

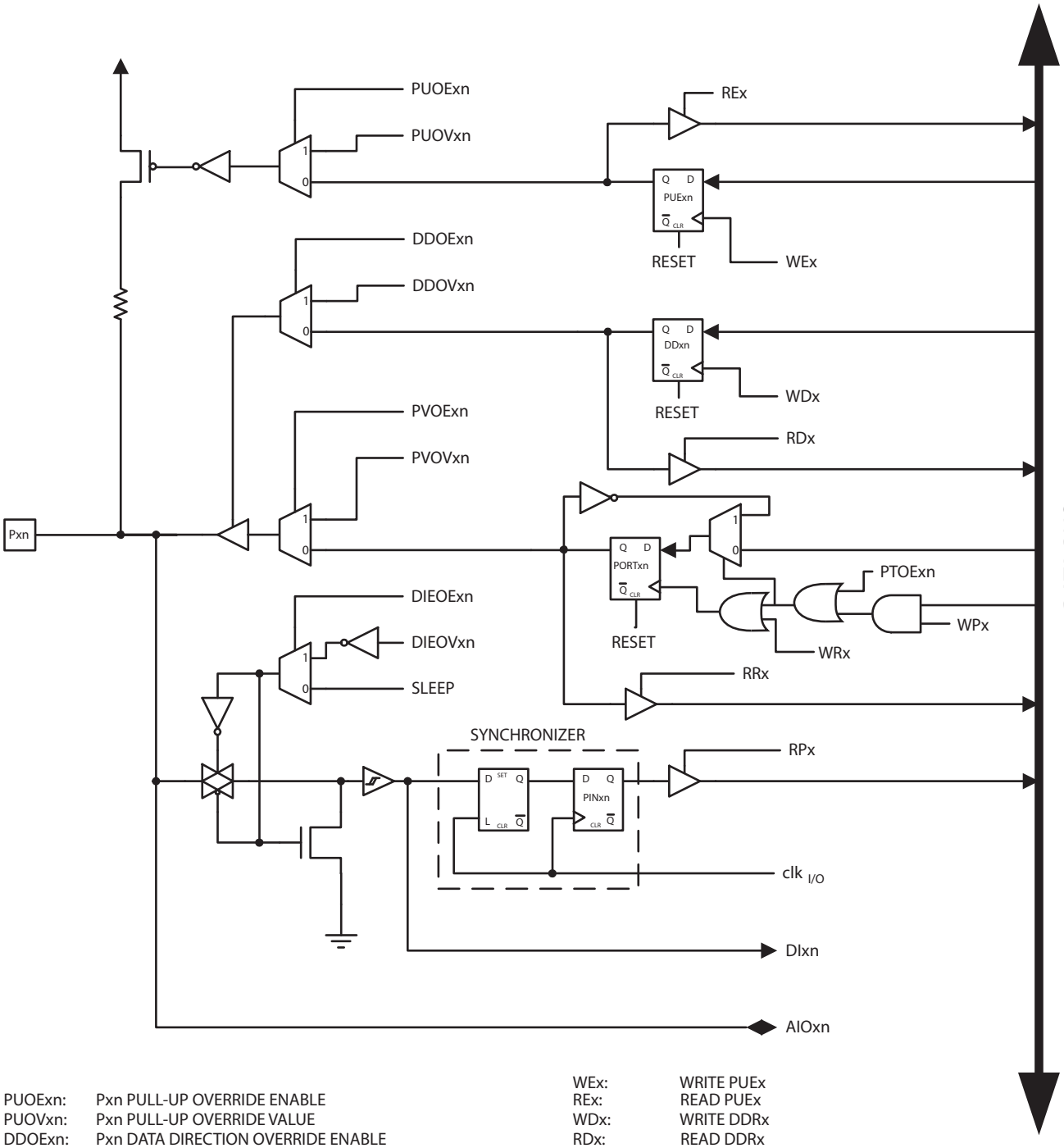
```
; Define pull-ups and set outputs high
; Define directions for port pins
ldi                 r16,(1<<PUEB2)
ldi                 r17,(1<<PB0)
ldi                 r18,(1<<DDB1)|(1<<DDB0)
out                 PUEB,r16
out                 PORTB,r17
out                 DDRB,r18

; Insert nop for synchronization
nop

; Read port pins
in                  r16,PINB
```

Note: See “Code Examples” on page 7.

10.3 Alternative Port Functions

Most port pins have alternative functions in addition to being general digital I/Os. In Figure 25 below is shown how the port pin control signals from the simplified Figure 22 on page 60 can be overridden by alternative functions.

Figure 25. Alternative Port Functions



PUOExn:	Pxn PULL-UP OVERRIDE ENABLE
PUOVxn:	Pxn PULL-UP OVERRIDE VALUE
DDOExn:	Pxn DATA DIRECTION OVERRIDE ENABLE
DDOVxn:	Pxn DATA DIRECTION OVERRIDE VALUE
PVOExn:	Pxn PORT VALUE OVERRIDE ENABLE
PVOVxn:	Pxn PORT VALUE OVERRIDE VALUE
DIEOExn:	Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE
DIEOVxn:	Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE
SLEEP:	SLEEP CONTROL
PTOExn:	Pxn, PORT TOGGLE OVERRIDE ENABLE

WEx:	WRITE PUEx
REx:	READ PUEx
WDx:	WRITE DDRx
RDx:	READ DDRx
RRx:	READ PORTx REGISTER
WRx:	WRITE PORTx
RPx:	READ PORTx PIN
WPx:	WRITE PINx
$clk_{I/O}$:	I/O CLOCK
DIxn:	DIGITAL INPUT PIN n ON PORTx
AIOxn:	ANALOG INPUT/OUTPUT PIN n ON PORTx

Note: WEx, WRx, WPx, WDx, REx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, and SLEEP are common to all ports. All other signals are unique for each pin.

The illustration in the figure above serves as a generic description applicable to all port pins in the AVR microcontroller family. Some overriding signals may not be present in all port pins.

Table 20 summarizes the function of the overriding signals. The pin and port indexes from Figure 25 on page 64 are not shown in the succeeding tables. The overriding signals are generated internally in the modules having the alternative function.

Table 20. Generic Description of Overriding Signals for Alternative Functions

Signal	Full Name	Description
PUOE	Pull-Up Override Enable	If this signal is set, the pull-up enable is controlled by the PUOV signal. If this signal is cleared, the pull-up is enabled when PUExn = 0b1.
PUOV	Pull-Up Override Value	If PUOE is set, the pull-up is enabled/disabled when PUOV is set/cleared, regardless of the setting of the PUExn Register bit.
DDOE	Data Direction Override Enable	If this signal is set, the Output Driver Enable is controlled by the DDOV signal. If this signal is cleared, the Output driver is enabled by the DDxn Register bit.
DDOV	Data Direction Override Value	If DDOE is set, the Output Driver is enabled/disabled when DDOV is set/cleared, regardless of the setting of the DDxn Register bit.
PVOE	Port Value Override Enable	If this signal is set and the Output Driver is enabled, the port value is controlled by the PVOV signal. If PVOE is cleared, and the Output Driver is enabled, the port Value is controlled by the PORTxn Register bit.
PVOV	Port Value Override Value	If PVOE is set, the port value is set to PVOV, regardless of the setting of the PORTxn Register bit.
PTOE	Port Toggle Override Enable	If PTOE is set, the PORTxn Register bit is inverted.
DIEOE	Digital Input Enable Override Enable	If this bit is set, the Digital Input Enable is controlled by the DIEOV signal. If this signal is cleared, the Digital Input Enable is determined by MCU state (Normal mode, sleep mode).
DIEOV	Digital Input Enable Override Value	If DIEOE is set, the Digital Input is enabled/disabled when DIEOV is set/cleared, regardless of the MCU state (Normal mode, sleep mode).
DI	Digital Input	This is the Digital Input to alternative functions. In the figure, the signal is connected to the output of the schmitt-trigger but before the synchronizer. Unless the Digital Input is used as a clock source, the module with the alternative function will use its own synchronizer.
AIO	Analog Input/Output	This is the Analog Input/Output to/from alternative functions. The signal is connected directly to the pad, and can be used bi-directionally.

The following subsections shortly describe the alternative functions for each port, and relate the overriding signals to the alternative function. Refer to the alternative function description for further details.

10.3.1 Alternative Functions of Port A

The alternative functions of port A are shown in Table 21.

Table 21. Alternative Functions of Port A

Pin	Function	Description of Alternative Function
PA0	PCINT0	Pin change interrupt source
	ADC0	Input channel for analog to digital converter (ADC)
PA1	PCINT1	Pin change interrupt source
	ADC1	Input channel for analog to digital converter (ADC)
	AIN0	Positive input of analog comparator (1)
PA2	PCINT2	Pin change interrupt source
	ADC2	Input channel for analog to digital converter (ADC)
	AIN1	Negative input channel of analog comparator (1)
PA3	PCINT3	Pin change interrupt source
	ADC3	Input channel for analog to digital converter (ADC)
PA4	PCINT4	Pin change interrupt source
	ADC4	Input channel for analog to digital converter (ADC)
PA5	PCINT5	Pin change interrupt source
	ADC5	Input channel for analog to digital converter (ADC)
PA6	PCINT6	Pin change interrupt source
	ADC6	Input channel for analog to digital converter (ADC)
PA7	PCINT7	Pin change interrupt source
	ADC7	Input channel for analog to digital converter (ADC)

Note: 1. Configure port pin as input with the internal pull-up switched off to avoid the digital port function from interfering with the function of the analog comparator.

Table 22, below, summarises the override signals used by the alternative functions of the port. For an illustration on how signals are used, see Figure 25 on page 64.

Table 22. Override Signals of Port A

Pin	Signal	Composition
PA0	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	0
	PVOV	0
	PTOE	0
	DIEOE	(PCINT0 • PCIE0) + ADC0D
	DIEOV	PCINT0 • PCIE0
	DI	PCINT0 Input
	AIO	ADC0 Input
PA1	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	0
	PVOV	0
	PTOE	0
	DIEOE	(PCINT1 • PCIE0) + ADC1D
	DIEOV	PCINT1 • PCIE0
	DI	PCINT1 Input
	AIO	ADC1 Input
PA2	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	0
	PVOV	0
	PTOE	0
	DIEOE	(PCINT2 • PCIE0) + ADC2D
	DIEOV	PCINT2 • PCIE0
	DI	PCINT2 Input
	AIO	ADC2 Input

Pin	Signal	Composition
PA3	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	0
	PVOV	0
	PTOE	0
	DIEOE	(PCINT3 • PCIE0) + ADC3D
	DIEOV	PCINT3 • PCIE0
	DI	PCINT3 Input
	AIO	ADC3 Input
PA4	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	0
	PVOV	0
	PTOE	0
	DIEOE	(PCINT4 • PCIE0) + ADC4D
	DIEOV	PCINT4 • PCIE0
	DI	PCINT4 Input
	AIO	ADC4 Input
PA5	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	0
	PVOV	0
	PTOE	0
	DIEOE	(PCINT5 • PCIE0) + ADC5D
	DIEOV	PCINT5 • PCIE0
	DI	PCINT5 Input
	AIO	ADC5 Input

Pin	Signal	Composition
PA6	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	0
	PVOV	0
	PTOE	0
	DIEOE	(PCINT6 • PCIE0) + ADC6D
	DIEOV	PCINT6 • PCIE0
	DI	PCINT6 Input
	AIO	ADC6 Input
PA7	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	0
	PVOV	0
	PTOE	0
	DIEOE	(PCINT7 • PCIE0) + ADC7D
	DIEOV	PCINT7 • PCIE0
	DI	PCINT7 Input
	AIO	ADC7 Input

10.3.2 Alternative Functions of Port B

The alternative functions of port B are shown in Table 23.

Table 23. Alternative Functions of Port B

Pin	Function	Description of Alternative Function
PB0	PCINT8	Pin change interrupt source
	ADC8	Input channel for analog to digital converter (ADC)
PB1	PCINT9	Pin change interrupt source
	ADC9	Input channel for analog to digital converter (ADC)

Pin	Function	Description of Alternative Function
PB2	PCINT10	Pin change interrupt source
	ADC10	Input channel for analog to digital converter (ADC)
PB3	PCINT11	Pin change interrupt source
	ADC11	Input channel for analog to digital converter (ADC)
PB4	PCINT12	Pin change interrupt source
	ADC12	Input channel for analog to digital converter (ADC)
PB5	PCINT13	Pin change interrupt source
	ADC13	Input channel for analog to digital converter (ADC)
PB6	PCINT14	Pin change interrupt source
	ADC14	Input channel for analog to digital converter (ADC)
PB7	PCINT15	Pin change interrupt source
	ADC15	Input channel for analog to digital converter (ADC)

Table 24, below, summarises the override signals used by the alternative functions of the port. For an illustration on how signals are used, see Figure 25 on page 64.

Table 24. Override Signals of Port B

Pin	Signal	Composition
PB0	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	0
	PVOV	0
	PTOE	0
	DIEOE	(PCINT8 • PCIE1) + ADC8D
	DIEOV	PCINT8 • PCIE1
	DI	PCINT8 Input
	AIO	ADC8 Input

Pin	Signal	Composition
PB1	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	0
	PVOV	0
	PTOE	0
	DIEOE	(PCINT9 • PCIE1) + ADC9D
	DIEOV	PCINT9 • PCIE1
	DI	PCINT9 Input
	AIO	ADC9 Input
PB2	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	0
	PVOV	0
	PTOE	0
	DIEOE	(PCINT10 • PCIE1) + ADC10D
	DIEOV	PCINT10 • PCIE1
	DI	PCINT10 Input
	AIO	ADC10 Input

Pin	Signal	Composition
PB3	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	0
	PVOV	0
	PTOE	0
	DIEOE	(PCINT11 • PCIE1) + ADC11D
	DIEOV	PCINT11 • PCIE1
	DI	PCINT11 Input
	AIO	ADC11 Input
PB4	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	0
	PVOV	0
	PTOE	0
	DIEOE	(PCINT12 • PCIE1) + ADC12D
	DIEOV	PCINT12 • PCIE1
	DI	PCINT12 Input
	AIO	ADC12 Input
PB5	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	0
	PVOV	0
	PTOE	0
	DIEOE	(PCINT13 • PCIE1) + ADC13D
	DIEOV	PCINT13 • PCIE1
	DI	PCINT13 Input
	AIO	ADC13 Input

Pin	Signal	Composition
PB6	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	0
	PVOV	0
	PTOE	0
	DIEOE	(PCINT14 • PCIE1) + ADC14D
	DIEOV	PCINT14 • PCIE1
	DI	PCINT14 Input
	AIO	ADC14 Input
PB7	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	0
	PVOV	0
	PTOE	0
	DIEOE	(PCINT15 • PCIE1) + ADC15D
	DIEOV	PCINT15 • PCIE1
	DI	PCINT15 Input
	AIO	ADC15 Input

10.3.3 Alternative Functions of Port C

The alternative functions of port C are shown in Table 25.

Table 25. Alternative Functions of Port C

Pin	Function	Description of Alternative Function
PC0	PCINT16	Pin change interrupt source
	ADC16	Input channel for analog to digital converter (ADC)
	TOCC0	Timer/counter output compare, channel 0 (1)
	SS	SPI Slave Select input (2)
	XCK	USART transfer clock

Pin	Function	Description of Alternative Function
PC1	PCINT17	Pin change interrupt source
	ADC17	Input channel for analog to digital converter (ADC)
	TOCC1	Timer/counter output compare, channel 1 (1)
	INT0	External interrupt request 0
	CLKO	System clock output (3)
PC2	PCINT18	Pin change interrupt source
	ADC18	Input channel for analog to digital converter (ADC)
	TOCC2	Timer/counter output compare, channel 2 (1)
	RXD	USART serial data input
	INT1	External interrupt request 1
PC3	PCINT19	Pin change interrupt source
	ADC19	Input channel for analog to digital converter (ADC)
	TOCC3	Timer/counter output compare, channel 3 (1)
	TXD	USART serial data output
PC4	PCINT20	Pin change interrupt source
	ADC20	Input channel for analog to digital converter (ADC)
	TOCC4	Timer/counter output compare, channel 4 (1)
PC5	PCINT21	Pin change interrupt source
	ADC21	Input channel for analog to digital converter (ADC)
	TOCC5	Timer/counter output compare, channel 5 (1)
	ICP1	Input capture pin
	T0	Timer/Counter0 Clock Source
PC6	PCINT22	Pin change interrupt source
	ADC22	Input channel for analog to digital converter (ADC)
	TOCC6	Timer/counter output compare, channel 6 (1)
	CLKI	Clock input from external source
PC7	PCINT23	Pin change interrupt source
	ADC23	Input channel for analog to digital converter (ADC)
	TOCC7	Timer/counter output compare, channel 7 (1)
	T1	Timer/Counter1 Clock Source

Note: 1. See “TOCPMSA1 and TOCPMSA0 – Timer/Counter Output Compare Pin Mux Selection Registers” on page 127.

2. When SPI is enabled as a slave, this pin is automatically configured as an input, regardless of the data direction bit of the pin. When SPI is enabled as a master normal pin control of data direction is resumed.
3. When the CKOUT fuse is programmed, the system clock is output on this pin, regardless of pin settings. The clock is also output when the device is reset.

Table 26, below, summarises the override signals used by the alternative functions of the port. For an illustration on how signals are used, see Figure 25 on page 64.

Table 26. Override Signals of Port C

Pin	Signal	Composition
PC0	PUOE	0
	PUOV	0
	DDOE	SPE • $\overline{\text{MSTR}}$
	DDOV	0
	PVOE	TOCC0OE + XCK_MASTER (1)
	PVOV	XCK_MASTER • XCK_OUT + $\overline{\text{XCK_MASTER}}$ • TOCC0_OUT
	PTOE	0
	DIEOE	(PCINT16 • PCIE2) + ADC16D + (XCK_SLAVE(2) • RXEN • SFDE)
	DIEOV	(PCINT16 • PCIE2) + (XCK_SLAVE(2) • RXEN • SFDE)
	DI	PCINT16 Input / XCK_IN / $\overline{\text{SS}}$ Input
	AIO	ADC16 Input
PC1	PUOE	0
	PUOV	0
	DDOE	CKOUT(3)
	DDOV	CKOUT(3)
	PVOE	0TOCC1E + CKOUT(3)
	PVOV	CKOUT(3) • SYSTEM_CLOCK + $\overline{\text{CKOUT}}$ • TOCC1_OUT
	PTOE	0
	DIEOE	(PCINT17 • PCIE2) + ADC17D + INT0
	DIEOV	PCINT17 • PCIE2 + INT0
	DI	PCINT17 Input / INT0 Input
	AIO	ADC17 Input

Pin	Signal	Composition
PC2	PUOE	0
	PUOV	0
	DDOE	RXEN
	DDOV	0
	PVOE	TOCC2OE
	PVOV	TOCC2_OUT
	PTOE	0
	DIEOE	(PCINT18 • PCIE2) + ADC18D + (RXEN ● SFDE) + INT1
	DIEOV	PCINT18 • PCIE2 + (RXEN ● SFDE) + INT1
	DI	PCINT18 Input / INT1 Input / RXD_IN
	AIO	ADC18 Input

Pin	Signal	Composition
PC3	PUOE	TXEN
	PUOV	0
	DDOE	TXEN
	DDOV	0
	PVOE	TOCC3OE + TXEN
	PVOV	TXEN • TXD_OUT + $\overline{\text{TXEN}}$ • TOCC3_OUT
	PTOE	0
	DIEOE	(PCINT19 • PCIE2) + ADC19D
	DIEOV	PCINT19 • PCIE2
	DI	PCINT19 Input
	AIO	ADC19 Input
PC4	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	TOCC4OE
	PVOV	TOCC4_OUT
	PTOE	0
	DIEOE	(PCINT20 • PCIE2) + ADC20D
	DIEOV	PCINT20 • PCIE2
	DI	PCINT20 Input
	AIO	ADC20 Input
PC5	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	TOCC5OE
	PVOV	TOCC5_OUT
	PTOE	0
	DIEOE	(PCINT21 • PCIE2) + ADC21D
	DIEOV	PCINT21 • PCIE2
	DI	PCINT21 Input / T0_IN / ICP1_IN
	AIO	ADC21 Input

Pin	Signal	Composition
PC6	PUOE	0
	PUOV	0
	DDOE	EXT_CLOCK(4)
	DDOV	0
	PVOE	TOCC6OE + EXT_CLOCK(4)
	PVOV	TOCC6_OUT • $\overline{\text{EXT_CLOCK}}$(4)
	PTOE	0
	DIEOE	(PCINT22 • PCIE2) + ADC22D + EXT_CLOCK (4)
	DIEOV	(EXT_CLOCK(4) • $\overline{\text{PWR_DOWN}}$) + ($\overline{\text{EXT_CLOCK}}$ • PCINT22 • PCIE2)
	DI	PCINT22 Input / CLOCK
	AIO	ADC22 Input
PC7	PUOE	0
	PUOV	0
	DDOE	0
	DDOV	0
	PVOE	TOCC7OE
	PVOV	TOCC7_OUT
	PTOE	0
	DIEOE	(PCINT23 • PCIE2) + ADC23D
	DIEOV	PCINT23 • PCIE2
	DI	PCINT23 Input
	AIO	ADC23 Input / T1_IN

Notes:
1. XCK_MASTER = UMSEL1 • UMSEL0 + UMSEL0 • DDRC0
2. XCK_SLAVE = $\overline{\text{UMSEL1}}$ • UMSEL0 • $\overline{\text{DDRC0}}$
3. CKOUT is 1 when the fuse bit is “0” (programmed)
4. EXT_CLOCK means that external clock is selected (by the CKSEL fuses)

10.3.4 Alternative Functions of Port D

The alternative functions of port D are shown in Table 27.

Table 27. Alternative Functions of Port D

Pin	Function	Description of Alternative Function
PD0	PCINT24	Pin change interrupt source
	ADC24	Input channel for analog to digital converter (ADC)
	SDA	Two-Wire Interface (TWI) data (1)
	MOSI	Master Output / Slave Input of SPI (2)
PD1	PCINT25	Pin change interrupt source
	ADC25	Input channel for analog to digital converter (ADC)
	MISO	Master Input / Slave Output of SPI (3)
PD2	PCINT26	Pin change interrupt source
	ADC26	Input channel for analog to digital converter (ADC)
	RESET	External reset input, active low (4)
	dW	Input / Output of debugWire
PD3	PCINT27	Pin change interrupt source
	ADC27	Input channel for analog to digital converter (ADC)
	SCL	Two-Wire Interface (TWI) clock (5)
	SCK	Master clock output / slave clock input of SPI (2)

Note:
1. When TWEN in TWSCRA is set, this pin is disconnected from the port and becomes the serial data for the TWI . In this mode of operation, the pin is driven by an open-drain circuit with slew rate limitation and spike filter.
2. When SPI is enabled as a slave, this pin is automatically configured as an input, regardless of the data direction bit of the pin. When SPI is enabled as a master normal pin control of data direction is resumed.
3. When SPI is enabled as a master, this pin is automatically configured as an input, regardless of the data direction bit of the pin. When SPI is enabled as a slave normal pin control of data direction is resumed.
4. Enabled by unprogramming the RSTDISBL fuse. When used as a reset pin, the pin pullup resistor is activated and output driver and digital input are deactivated.
5. When TWEN in TWSCRA is set, this pin is disconnected from the port and becomes the serial clock for the TWI. In this mode of operation, the pin is driven by an open-drain circuit with slew rate limitation and spike filter.

Table 28, below, summarises the override signals used by the alternative functions of the port. For an illustration on how signals are used, see Figure 25 on page 64.

Table 28. Override Signals of Port D

Pin	Signal	Composition
PD0	PUOE	0
	PUOV	0
	DDOE	TWEN + (SPE • $\overline{\text{MSTR}}$)
	DDOV	TWEN • $\overline{\text{SDA_OUT}}$
	PVOE	TWEN + (SPE • MSTR)
	PVOV	$\overline{\text{TWEN}}$ • SPE • MSTR • SPI_MSTR_OUT
	PTOE	0
	DIEOE	(PCINT24 • PCIE3) + ADC24D
	DIEOV	PCINT24 • PCIE3
	DI	PCINT24 Input / SPI_SLAVE_IN
	AIO	ADC24 Input / SDA_IN
PD1	PUOE	0
	PUOV	0
	DDOE	SPE • MSTR
	DDOV	0
	PVOE	SPE • $\overline{\text{MSTR}}$
	PVOV	SPI_SLAVE_OUT
	PTOE	0
	DIEOE	(PCINT25 • PCIE3) + ADC25D
	DIEOV	PCINT25 • PCIE3
	DI	PCINT25 Input / SPI_MASTER_IN
	AIO	ADC25 Input
PD2	PUOE	$\overline{\text{RSTDISBL}}$[1]
	PUOV	1
	DDOE	$\overline{\text{RSTDISBL}}$[1]
	DDOV	0
	PVOE	$\overline{\text{RSTDISBL}}$[1]
	PVOV	0
	PTOE	0
	DIEOE	(PCINT26 • PCIE3) + ADC26D + $\overline{\text{RSTDISBL}}$[1]
	DIEOV	PCINT26 • PCIE3 • RSTDISBL[1]
	DI	PCINT26 Input
	AIO	ADC26 Input / RESET Input

Pin	Signal	Composition
PD3	PUOE	0
	PUOV	0
	DDOE	TWEN + (SPE • $\overline{\text{MSTR}}$)
	DDOV	TWEN • $\overline{\text{SCL_OUT}}$
	PVOE	TWEN + (SPE • MSTR)
	PVOV	$\overline{\text{TWEN}}$ • SPE • MSTR • SCK_OUT
	PTOE	0
	DIEOE	(PCINT27 • PCIE3) + ADC27D
	DIEOV	PCINT27 • PCIE3
	DI	PCINT27 Input / SCK_IN
	AIO	ADC27 Input / SCL_IN

Note: 1. RSTDISBL is 1 when the fuse bit is “0” (programmed)

10.4 Register Description

10.4.1 PHDE – Port High Drive Enable Register

Bit	7	6	5	4	3	2	1	0	
0x14 (0x34)	–	–	–	–	–	**PHDEC**	–	–	**PHDE**
Read/Write	R	R	R	R	R	R/W	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:3 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 2 – PHDEC: Port C High Drive Enable**

When this bit is set the extra high sink capability of port C is enabled.

- **Bits 1:0 – Res: Reserved Bits**

These bits are reserved and will always read zero.

10.4.2 PUED – Port D Pull-Up Enable Control Register

Bit	7	6	5	4	3	2	1	0	
0x0F (0x2F)	–	–	–	–	PUED3	PUED2	PUED1	PUED0	PUED
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 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bits 3:0 – PUED[3:0]: Pull-Up Enable Bits**

When a pull-up enable bit, PUEDn, is set the pull-up resistor on the equivalent port pin, PDn, is enabled.

10.4.3 PORTD – Port D Data Register

Bit	7	6	5	4	3	2	1	0	
0x0E (0x2E)	–	–	–	–	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
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 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bits 3:0 – PORTD[3:0]: Port Data Bits**

When pin PDn is configured as an output, setting PORTDn will drive PDn high. Clearing PORTDn will drive PDn low. When the pin is configured as an input the value of the PORTxn bit doesn't matter. See Table 19 on page 61.

10.4.4 DDRD – Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0D (0x2D)	–	–	–	–	DDD3	DDD2	DDD1	DDD0	DDRD
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 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bits 3:0 – DDD[3:0]: Data Direction Bits**

When DDDn is set, the pin PDn is configured as an output. When DDDn is cleared, the pin is configured as an input.

10.4.5 PIND – Port D Input Pins

Bit	7	6	5	4	3	2	1	0	
0x0C (0x2C)	–	–	–	–	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

- **Bits 7:4 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bits 3:0 – PIND[3:0]: Port Input Data**

Regardless of the setting of the data direction bit, the value of the port pin PDn can be read through the PINDn bit. Writing a logic one to PINDn toggles the value of PORTDn, regardless of the value in DDDn.

10.4.6 PUEC – Port C Pull-Up Enable Control Register

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	PUEC7	PUEC6	PUEC5	PUEC4	PUEC3	PUEC2	PUEC1	PUEC0	PUEC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – PUEC[7:0]: Pull-Up Enable Bits**

When a pull-up enable bit, PUECn, is set the pull-up resistor on the equivalent port pin, PCn, is enabled.

10.4.7 PORTC – Port C Data Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – PORTC[3:0]: Port Data Bits**

When pin PCn is configured as an output, setting PORTCn will drive PCn high. Clearing PORTCn will drive PCn low. When the pin is configured as an input the value of the PORTxn bit doesn't matter. See Table 19 on page 61.

10.4.8 DDRC – Port C Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – DDC[7:0]: Data Direction Bits**

When DDCn is set, the pin PCn is configured as an output. When DDCn is cleared, the pin is configured as an input.

10.4.9 PINC – Port C Input Pins

Bit	7	6	5	4	3	2	1	0	
0x08 (0x28)	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

- **Bits 7:0 – PINC[7:0]: Port Input Data**

Regardless of the setting of the data direction bit, the value of the port pin PCn can be read through the PINCn bit. Writing a logic one to PINCn toggles the value of PORTCn, regardless of the value in DDCn.

10.4.10 PUEB – Port B Pull-Up Enable Control Register

Bit	7	6	5	4	3	2	1	0	
0x07 (0x27)	PUEB7	PUEB6	PUEB5	PUEB4	PUEB3	PUEB2	PUEB1	PUEB0	PUEB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – PUEB[7:0]: Pull-Up Enable Bits**

When a pull-up enable bit, PUEBn, is set the pull-up resistor on the equivalent port pin, PBn, is enabled.

10.4.11 PORTB – Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x06 (0x26)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – PORTB[3:0]: Port Data Bits**

When pin PBn is configured as an output, setting PORTBn will drive PBn high. Clearing PORTBn will drive PBn low. When the pin is configured as an input the value of the PORTxn bit doesn't matter. See Table 19 on page 61.

10.4.12 DDRB – Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x06 (0x26)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – DDB[7:0]: Data Direction Bits**

When DDBn is set, the pin PBn is configured as an output. When DDBn is cleared, the pin is configured as an input.

10.4.13 PINB – Port B Input Pins

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

- **Bits 7:0 – PINB[7:0]: Port Input Data**

Regardless of the setting of the data direction bit, the value of the port pin PBn can be read through the PINBn bit. Writing a logic one to PINBn toggles the value of PORTBn, regardless of the value in DDBn.

10.4.14 PUEA – Port A Pull-Up Enable Control Register

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PUEA7	PUEA6	PUEA5	PUEA4	PUEA3	PUEA2	PUEA1	PUEA0	PUEA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – PUEA[7:0]: Pull-Up Enable Bits**

When a pull-up enable bit, PUEAn, is set the pull-up resistor on the equivalent port pin, PAn, is enabled.

10.4.15 PORTA – Port A Data Register

Bit	7	6	5	4	3	2	1	0	
0x02 (0x22)	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – PORTA[3:0]: Port Data Bits**

When pin PAn is configured as an output, setting PORTAn will drive PAn high. Clearing PORTAn will drive PAn low. When the pin is configured as an input the value of the PORTxn bit doesn't matter. See Table 19 on page 61.

10.4.16 DDRA – Port A Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x01 (0x21)	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – DDA[7:0]: Data Direction Bits**

When DDAn is set, the pin PAn is configured as an output. When DDAn is cleared, the pin is configured as an input.

10.4.17 PINA – Port A Input Pins

Bit	7	6	5	4	3	2	1	0	
0x00 (0x20)	**PINA7**	**PINA6**	**PINA5**	**PINA4**	**PINA3**	**PINA2**	**PINA1**	**PINA0**	**PINA**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

- **Bits 7:0 – PINA[7:0]: Port Input Data**

Regardless of the setting of the data direction bit, the value of the port pin PAn can be read through the PINAn bit. Writing a logic one to PINAn toggles the value of PORTAn, regardless of the value in DDAn.

11. 8-bit Timer/Counter0 with PWM

11.1 Features

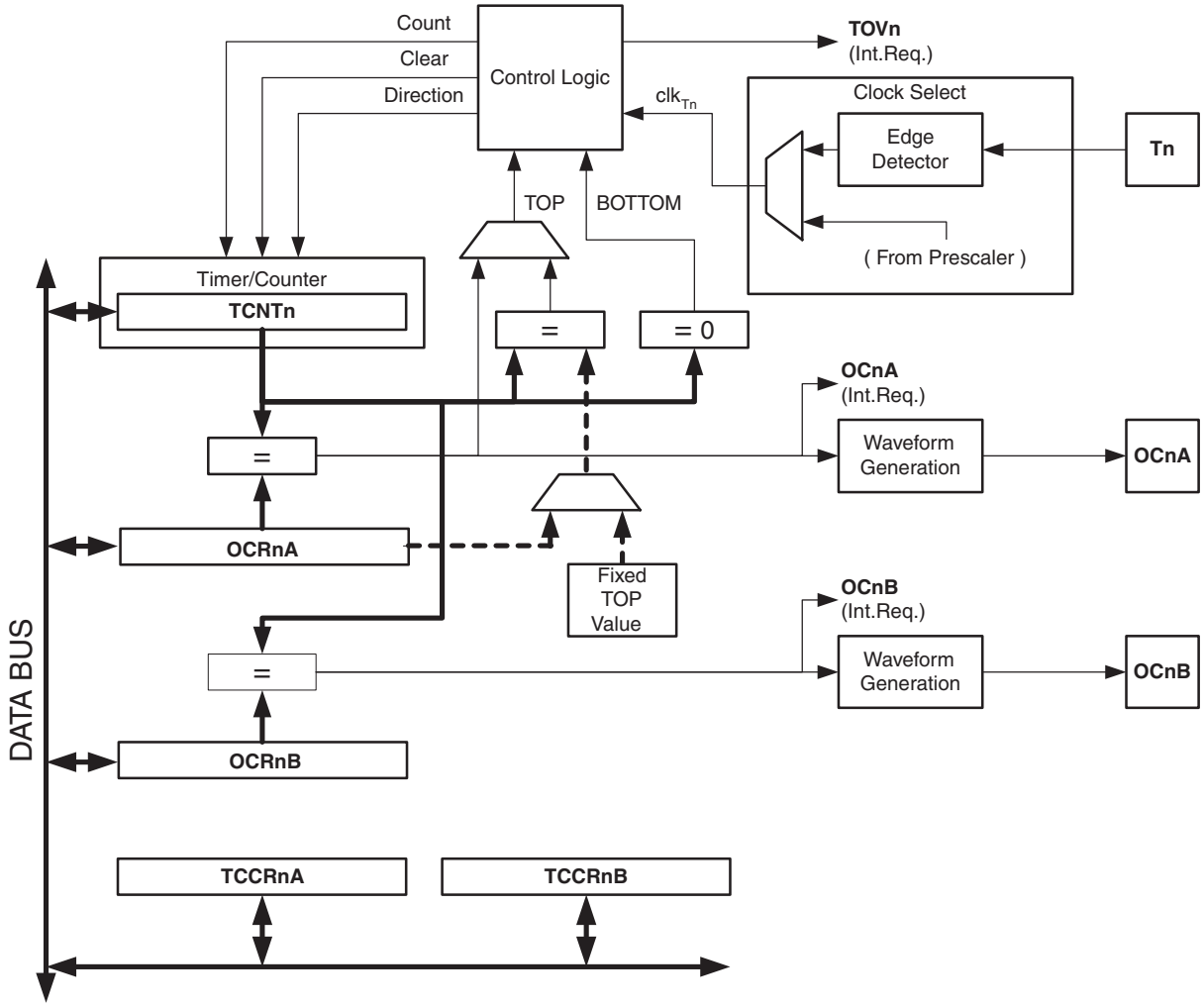
- Two Independent Output Compare Units
- Double Buffered Output Compare Registers
- Clear Timer on Compare Match (Auto Reload)
- Glitch Free, Phase Correct Pulse Width Modulator (PWM)
- Variable PWM Period
- Frequency Generator
- Three Independent Interrupt Sources (TOV0, OCF0A, and OCF0B)

11.2 Overview

Timer/Counter0 is a general purpose 8-bit Timer/Counter module, with two independent Output Compare Units, and with PWM support. It allows accurate program execution timing (event management) and wave generation.

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 26. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the “Register Description” on page 97.

Figure 26. 8-bit Timer/Counter Block Diagram



For actual placement of I/O pins, refer to Figure 1 on page 2 (MLF), and Figure 2 on page 2 (TQFP). Also, see "TOCPMSA1 and TOCPMSA0 – Timer/Counter Output Compare Pin Mux Selection Registers" on page 127, and "TOCPMCOE – Timer/Counter Output Compare Pin Mux Channel Output Enable" on page 128.

11.2.1 Registers

The Timer/Counter (TCNT0) and Output Compare Registers (OCR0A and OCR0B) are 8-bit registers. Interrupt request (abbreviated to Int.Req. in Figure 26) signals are all visible in the Timer Interrupt Flag Register (TIFR). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK). TIFR and TIMSK are not shown in the figure.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the T0 pin. The Clock Select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the Clock Select logic is referred to as the timer clock (clk_{T0}).

The double buffered Output Compare Registers (OCR0A and OCR0B) is compared with the Timer/Counter value at all times. The result of the compare can be used by the Waveform Generator to generate a PWM or variable frequency output on the Output Compare pins (OC0A and OC0B). See "Output Compare Unit" on page 89 for details. The Compare Match event will also set the Compare Flag (OCF0A or OCF0B) which can be used to generate an Output Compare interrupt request.

11.2.2 Definitions

Many register and bit references in this section are written in general form. A lower case "n" replaces the Timer/Counter number, in this case 0. A lower case "x" replaces the Output Compare Unit, in this case Compare Unit A or Compare Unit B. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT0 for accessing Timer/Counter0 counter value and so on.

The definitions in Table 29 are also used extensively throughout the document.

Table 29. Definitions

Constant	Description
BOTTOM	The counter reaches BOTTOM when it becomes 0x00
MAX	The counter reaches its MAXimum when it becomes 0xFF (decimal 255)
TOP	The counter reaches the TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be the fixed value 0xFF (MAX) or the value stored in the OCR0A Register. The assignment depends on the mode of operation

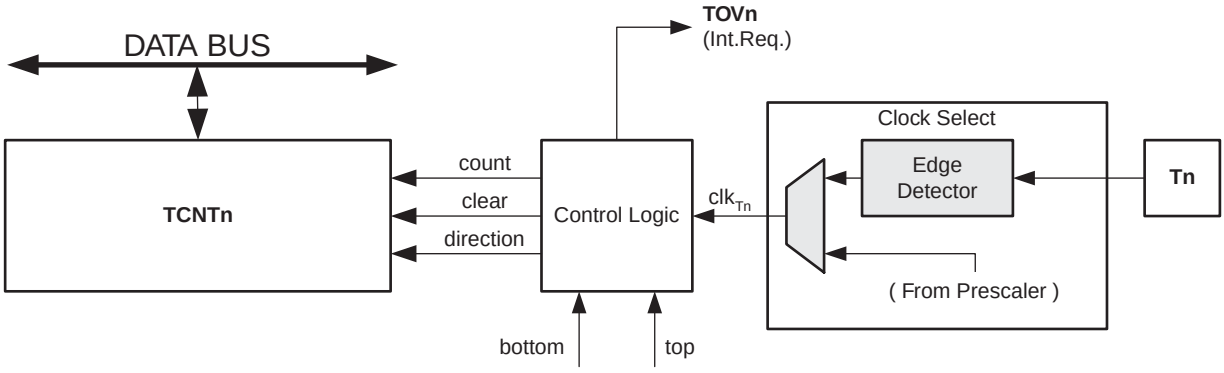
11.3 Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the Clock Select (CS0[2:0]) bits located in the Timer/Counter Control Register (TCCR0B). For details on clock sources and prescaler, see "Timer/Counter Prescaler" on page 131.

11.4 Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 27 on page 89 shows a block diagram of the counter and its surroundings.

Figure 27. Counter Unit Block Diagram



Signal description (internal signals):

count	Increment or decrement TCNT0 by 1.
direction	Select between increment and decrement.
clear	Clear TCNT0 (set all bits to zero).
clk_{Tn}	Timer/Counter clock, referred to as clk_T0 in the following.
top	Signalize that TCNT0 has reached maximum value.
bottom	Signalize that TCNT0 has reached minimum value (zero).

Depending of the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock (clk_{T0}). clk_{T0} can be generated from an external or internal clock source, selected by the Clock Select bits (CS0[2:0]). When no clock source is selected (CS0[2:0] = 0) the timer is stopped. However, the TCNT0 value can be accessed by the CPU, regardless of whether clk_{T0} is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the WGM01 and WGM00 bits located in the Timer/Counter Control Register (TCCR0A) and the WGM02 bit located in the Timer/Counter Control Register B (TCCR0B). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare output OC0A. For more details about advanced counting sequences and waveform generation, see “Modes of Operation” on page 92.

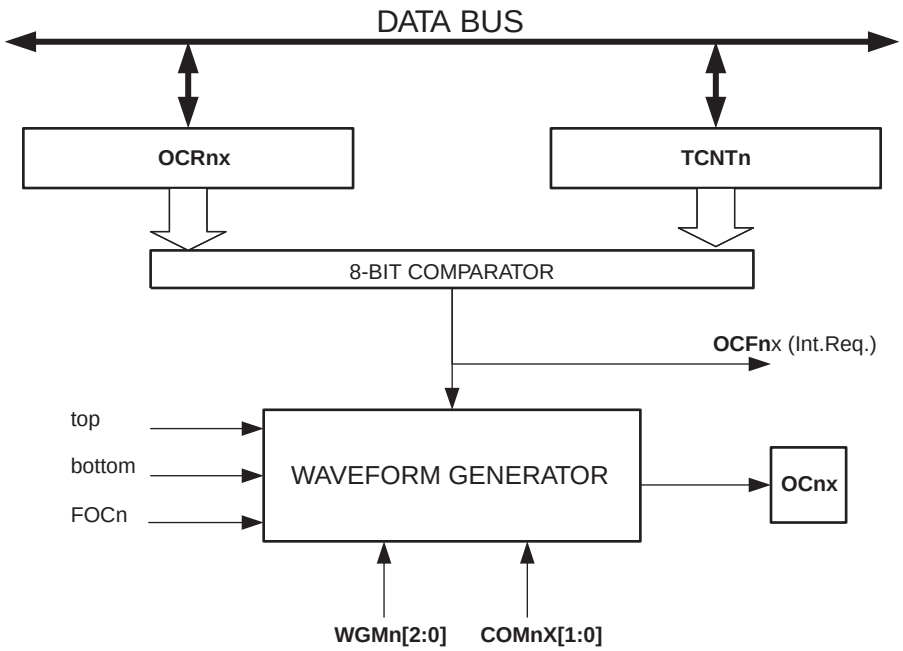
The Timer/Counter Overflow Flag (TOV0) is set according to the mode of operation selected by the WGM0[1:0] bits. TOV0 can be used for generating a CPU interrupt.

11.5 Output Compare Unit

The 8-bit comparator continuously compares TCNT0 with the Output Compare Registers (OCR0A and OCR0B). Whenever TCNT0 equals OCR0A or OCR0B, the comparator signals a match. A match will set the Output Compare Flag (OCF0A or OCF0B) at the next timer clock cycle. If the corresponding interrupt is enabled, the Output Compare Flag generates an Output Compare interrupt. The Output Compare Flag is automatically cleared when the interrupt is executed. Alternatively, the flag can be cleared by software by writing a logical one to its I/O bit location. The Waveform Generator uses the match signal to generate an output according to operating mode set by the WGM0[2:0] bits and Compare Output mode (COM0x[1:0]) bits. The max and bottom signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation. See “Modes of Operation” on page 92.

Figure 28 on page 90 shows a block diagram of the Output Compare unit.

Figure 28. Output Compare Unit, Block Diagram



The OCR0x Registers are double buffered when using any of the Pulse Width Modulation (PWM) modes. For the normal and Clear Timer on Compare (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR0x Compare Registers to either top or bottom of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR0x Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR0x Buffer Register, and if double buffering is disabled the CPU will access the OCR0x directly.

11.5.1 Force Output Compare

In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the Force Output Compare (0x) bit. Forcing Compare Match will not set the OCF0x Flag or reload/clear the timer, but the OC0x pin will be updated as if a real Compare Match had occurred (the COM0x[1:0] bits settings define whether the OC0x pin is set, cleared or toggled).

11.5.2 Compare Match Blocking by TCNT0 Write

All CPU write operations to the TCNT0 Register will block any Compare Match that occur in the next timer clock cycle, even when the timer is stopped. This feature allows OCR0x to be initialized to the same value as TCNT0 without triggering an interrupt when the Timer/Counter clock is enabled.

11.5.3 Using the Output Compare Unit

Since writing TCNT0 in any mode of operation will block all Compare Matches for one timer clock cycle, there are risks involved when changing TCNT0 when using the Output Compare Unit, independently of whether the Timer/Counter is running or not. If the value written to TCNT0 equals the OCR0x value, the Compare Match will be missed, resulting in incorrect waveform generation. Similarly, do not write the TCNT0 value equal to BOTTOM when the counter is down-counting.

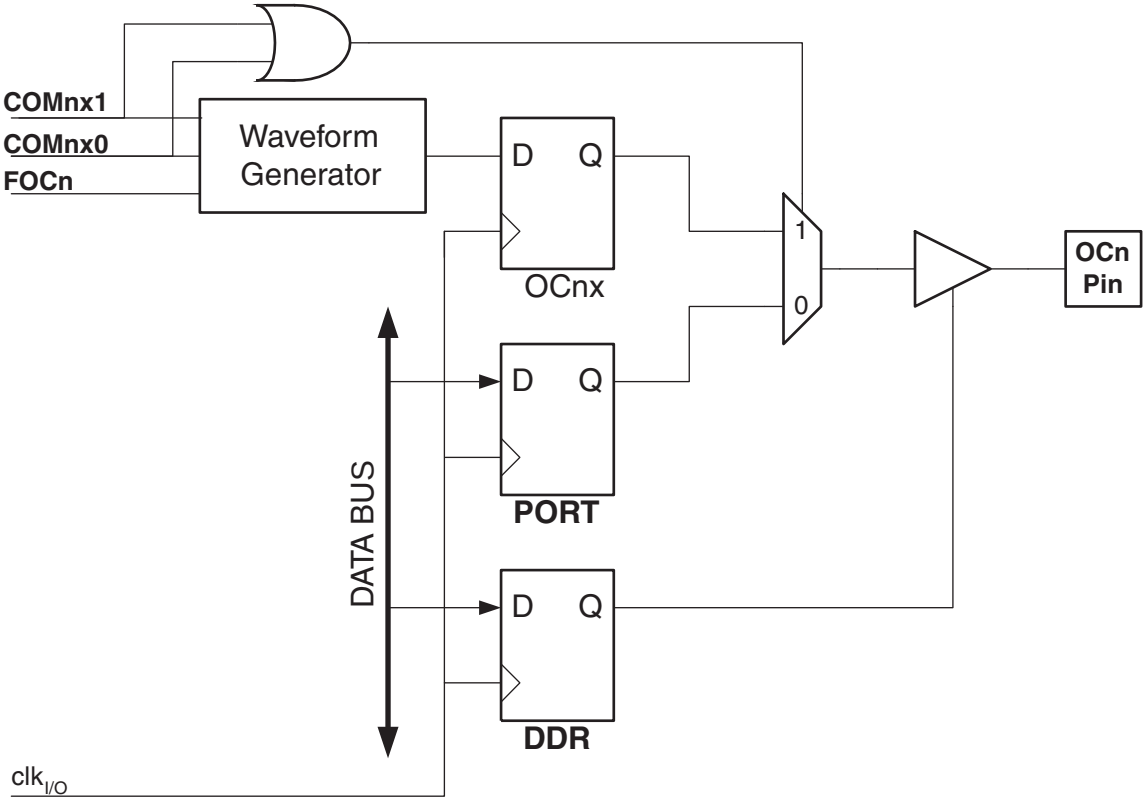
The setup of the OC0x should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC0x value is to use the Force Output Compare (0x) strobe bits in Normal mode. The OC0x Registers keep their values even when changing between Waveform Generation modes.

Be aware that the COM0x[1:0] bits are not double buffered together with the compare value. Changing the COM0x[1:0] bits will take effect immediately.

11.6 Compare Match Output Unit

The Compare Output mode (COM0x[1:0]) bits have two functions. The Waveform Generator uses the COM0x[1:0] bits for defining the Output Compare (OC0x) state at the next Compare Match. Also, the COM0x[1:0] bits control the OC0x pin output source. Figure 29 on page 91 shows a simplified schematic of the logic affected by the COM0x[1:0] bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O Port Control Registers (DDR and PORT) that are affected by the COM0x[1:0] bits are shown. When referring to the OC0x state, the reference is for the internal OC0x Register, not the OC0x pin. If a system reset occur, the OC0x Register is reset to “0”.

Figure 29. Compare Match Output Unit, Schematic



The general I/O port function is overridden by the Output Compare (OC0x) from the Waveform Generator if either of the COM0x[1:0] bits are set. However, the OC0x pin direction (input or output) is still controlled by the Data Direction Register (DDR) for the port pin. The Data Direction Register bit for the OC0x pin (DDR_OC0x) must be set as output before the OC0x value is visible on the pin. The port override function is independent of the Waveform Generation mode.

The design of the Output Compare pin logic allows initialization of the OC0x state before the output is enabled. Note that some COM0x[1:0] bit settings are reserved for certain modes of operation, see “Register Description” on page 97

11.6.1 Compare Output Mode and Waveform Generation

The Waveform Generator uses the COM0x[1:0] bits differently in Normal, CTC, and PWM modes. For all modes, setting the COM0x[1:0] = 0 tells the Waveform Generator that no action on the OC0x Register is to be performed on the next Compare Match. For compare output actions in the non-PWM modes refer to Table 30 on page 98. For fast PWM mode, refer to Table 31 on page 98, and for phase correct PWM refer to Table 32 on page 98.

A change of the COM0x[1:0] bits state will have effect at the first Compare Match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the Force Output Compare bits. See “TCCR0B – Timer/Counter Control Register B” on page 100.

11.7 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the Waveform Generation mode (WGM[2:0]) and Compare Output mode (COM0x[1:0]) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM0x[1:0] bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM0x[1:0] bits control whether the output should be set, cleared, or toggled at a Compare Match (See “Modes of Operation” on page 92).

For detailed timing information refer to Figure 33 on page 96, Figure 34 on page 96, Figure 35 on page 97 and Figure 36 on page 97 in “Timer/Counter Timing Diagrams” on page 96.

11.7.1 Normal Mode

The simplest mode of operation is the Normal mode (WGM0[2:0] = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 8-bit value (TOP = 0xFF) and then restarts from the bottom (0x00). In normal operation the Timer/Counter Overflow Flag (TOV0) will be set in the same timer clock cycle as the TCNT0 becomes zero. The TOV0 Flag in this case behaves like a ninth bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV0 Flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime.

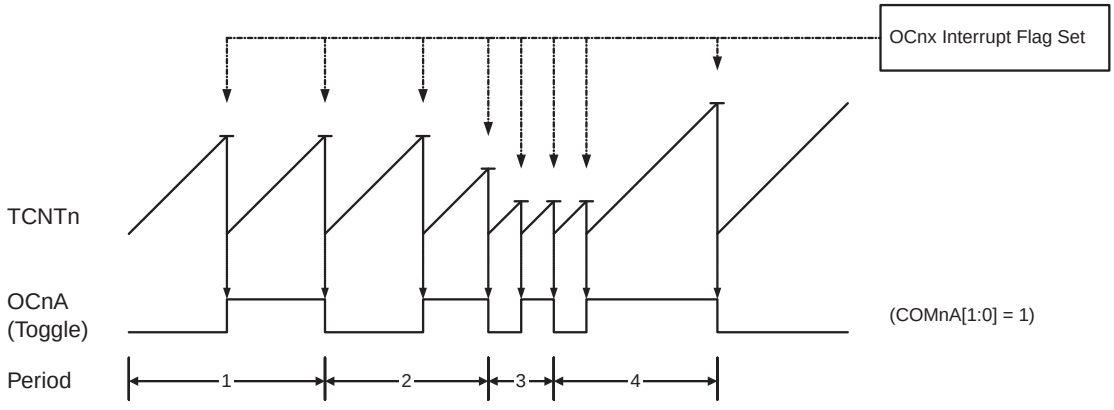
The Output Compare Unit can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

11.7.2 Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGM0[2:0] = 2), the OCR0A Register is used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT0) matches the OCR0A. The OCR0A defines the top value for the counter, hence also its resolution. This mode allows greater control of the Compare Match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 30 on page 92. The counter value (TCNT0) increases until a Compare Match occurs between TCNT0 and OCR0A, and then counter (TCNT0) is cleared.

Figure 30. CTC Mode, Timing Diagram



An interrupt can be generated each time the counter value reaches the TOP value by using the OCF0A Flag. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the

CTC mode does not have the double buffering feature. If the new value written to OCR0A is lower than the current value of TCNT0, the counter will miss the Compare Match. The counter will then have to count to its maximum value (0xFF) and wrap around starting at 0x00 before the Compare Match can occur.

For generating a waveform output in CTC mode, the OC0A output can be set to toggle its logical level on each Compare Match by setting the Compare Output mode bits to toggle mode (COM0A[1:0] = 1). The OC0A value will not be visible on the port pin unless the data direction for the pin is set to output. When OCR0A is set to zero (0x00) the waveform generated will have a maximum frequency of $f_{clk_I/O}/2$. The waveform frequency is defined by the following equation:

$$f_{OCnx} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRnA)}$$

The *N* variable represents the prescale factor (1, 8, 64, 256, or 1024).

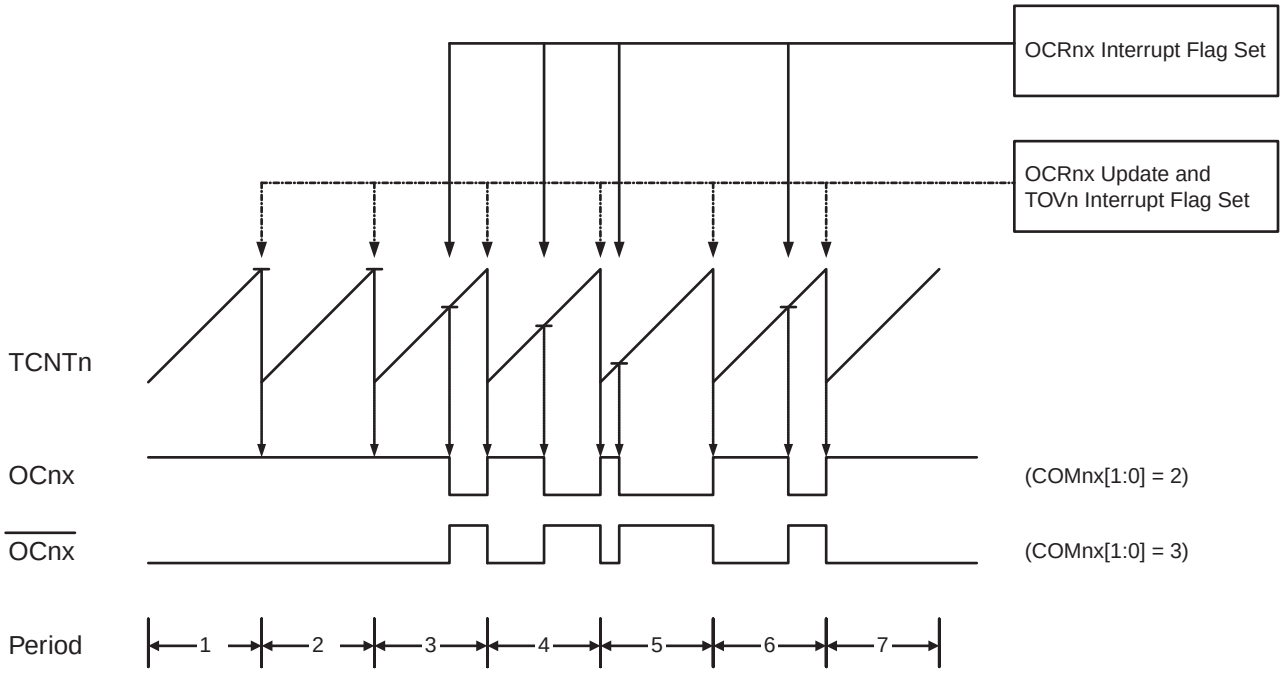
As for the Normal mode of operation, the TOV0 Flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

11.7.3 Fast PWM Mode

The fast Pulse Width Modulation or fast PWM mode (WGM0[2:0] = 3 or 7) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM option by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. TOP is defined as 0xFF when WGM0[2:0] = 3, and OCR0A when WGM0[2:0] = 7. In non-inverting Compare Output mode, the Output Compare (OC0x) is cleared on the Compare Match between TCNT0 and OCR0x, and set at BOTTOM. In inverting Compare Output mode, the output is set on Compare Match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct PWM mode that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), and therefore reduces total system cost.

In fast PWM mode, the counter is incremented until the counter value matches the TOP value. The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 31 on page 93. The TCNT0 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT0 slopes represent Compare Matches between OCR0x and TCNT0.

Figure 31. Fast PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV0) is set each time the counter reaches TOP. If the interrupt is enabled, the interrupt handler routine can be used for updating the compare value.

In fast PWM mode, the compare unit allows generation of PWM waveforms on the OC0x pins. Setting the COM0x[1:0] bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM0x[1:0] to three: Setting the COM0A[1:0] bits to one allowes the OC0A pin to toggle on Compare Matches if the WGM02 bit is set. This option is not available for the OC0B pin (See Table 31 on page 98). The actual OC0x value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by setting (or clearing) the OC0x Register at the Compare Match between OCR0x and TCNT0, and clearing (or setting) the OC0x Register at the timer clock cycle the counter is cleared (changes from TOP to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnxPWM} = \frac{f_{\mathrm{clk_I/O}}}{N \cdot (TOP + 1)}$$

The *N* variable represents the prescale factor (1, 8, 64, 256, or 1024).

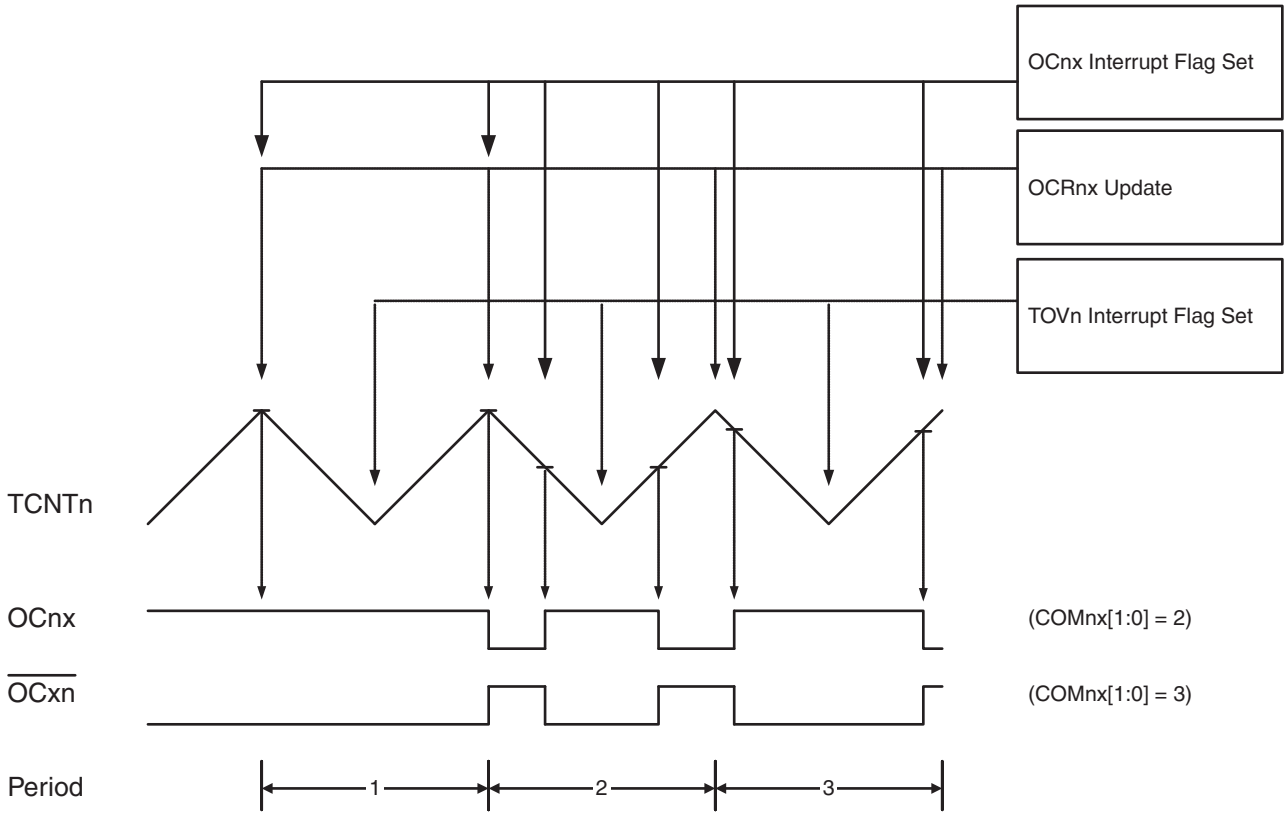
The extreme values for the OCR0x Register represents special cases when generating a PWM waveform output in the fast PWM mode. If OCR0x is set equal to BOTTOM, the output will be a narrow spike for each TOP+1 timer clock cycle. Setting the OCR0x equal to TOP will result in a constantly high or low output (depending on the polarity of the output set by the COM0x[1:0] bits.)

A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC0A to toggle its logical level on each Compare Match (COM0A[1:0] = 1). The waveform generated will have a maximum frequency of $_0$ = $f_{clk_I/O}/2$ when OCR0A is set to zero. This feature is similar to the OC0A toggle in CTC mode, except the double buffer feature of the Output Compare unit is enabled in the fast PWM mode.

11.7.4 Phase Correct PWM Mode

The phase correct PWM mode (WGM0[2:0] = 1 or 5) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is based on a dual-slope operation. The counter counts repeatedly from BOTTOM to TOP and then from TOP to BOTTOM. TOP is defined as 0xFF when WGM0[2:0] = 1, and OCR0A when WGM0[2:0] = 5. In non-inverting Compare Output mode, the Output Compare (OC0x) is cleared on the Compare Match between TCNT0 and OCR0x while upcounting, and set on the Compare Match while down-counting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

Figure 32. Phase Correct PWM Mode, Timing Diagram



In phase correct PWM mode the counter is incremented until the counter value matches TOP. When the counter reaches TOP, it changes the count direction. The TCNT0 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 32. The TCNT0 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT0 slopes represent Compare Matches between OCR0x and TCNT0.

The Timer/Counter Overflow Flag (TOV0) is set each time the counter reaches BOTTOM. The Interrupt Flag can be used to generate an interrupt each time the counter reaches the BOTTOM value.

In phase correct PWM mode, the compare unit allows generation of PWM waveforms on the OC0x pins. Setting the COM0x[1:0] bits to two will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM0x[1:0] to three: Setting the COM0A0 bits to one allows the OC0A pin to toggle on Compare Matches if the WGM02 bit is set. This option is not available for the OC0B pin (See Table 33 on page 99). The actual OC0x value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by clearing (or setting) the OC0x Register at the Compare Match between OCR0x and TCNT0 when the counter increments, and setting (or clearing) the OC0x Register at Compare Match between OCR0x and TCNT0 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnxPCPWM} = \frac{f_{clk_I/O}}{2 \times N \times TOP}$$

The N variable represents the prescale factor (1, 8, 64, 256, or 1024).

The extreme values for the OCR0x Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR0x is set equal to BOTTOM, the output will be continuously low and if set equal to TOP the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

At the very start of period 2 in Figure 32 on page 95 OCnx has a transition from high to low even though there is no Compare Match. The point of this transition is to guaratee symmetry around BOTTOM. There are two cases that give a transition without Compare Match.

- OCR0x changes its value from TOP, like in Figure 32 on page 95. When the OCR0x value is TOP the OCnx pin value is the same as the result of a down-counting Compare Match. To ensure symmetry around BOTTOM the OCnx value at TOP must correspond to the result of an up-counting Compare Match.
- The timer starts counting from a value higher than the one in OCR0x, and for that reason misses the Compare Match and hence the OCnx change that would have happened on the way up.

11.8 Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock (clk_{T0}) is therefore shown as a clock enable signal in the following figures. The figures include information on when Interrupt Flags are set. Figure 33 on page 96 contains timing data for basic Timer/Counter operation. The figure shows the count sequence close to the MAX value in all modes other than phase correct PWM mode.

Figure 33. Timer/Counter Timing Diagram, no Prescaling

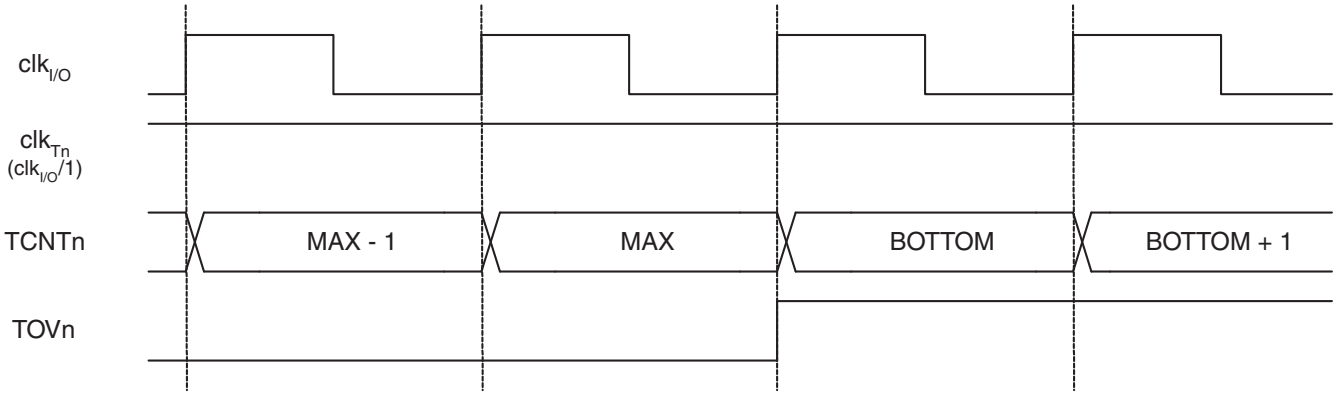


Figure 34 on page 96 shows the same timing data, but with the prescaler enabled.

Figure 34. Timer/Counter Timing Diagram, with Prescaler ($f_{clk_I/O}$/8)

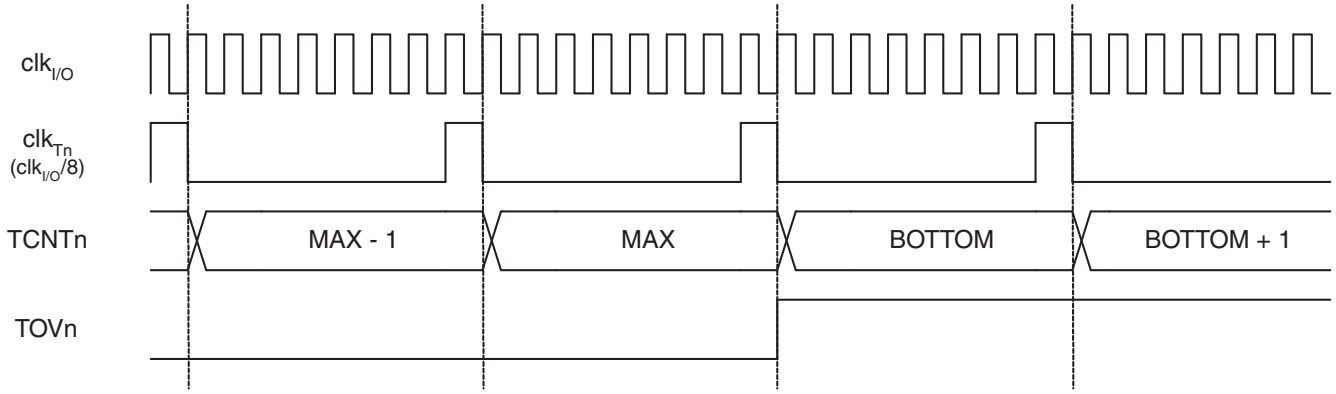


Figure 35 on page 97 shows the setting of OCF0B in all modes and OCF0A in all modes except CTC mode and PWM mode, where OCR0A is TOP.

Figure 35. Timer/Counter Timing Diagram, Setting of OCF0x, with Prescaler ($f_{clk_I/O}$/8)

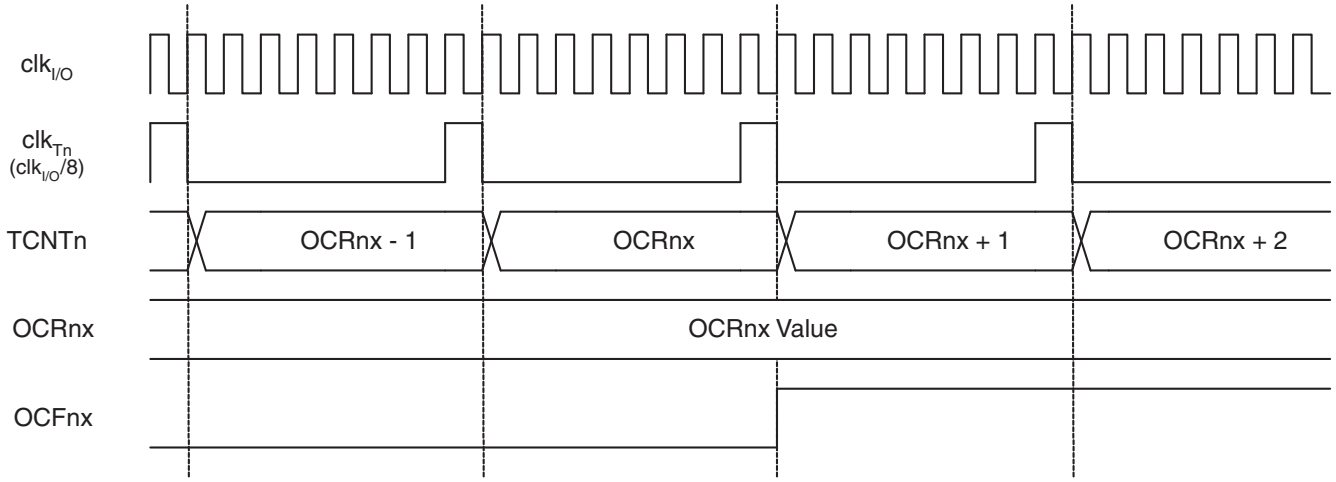
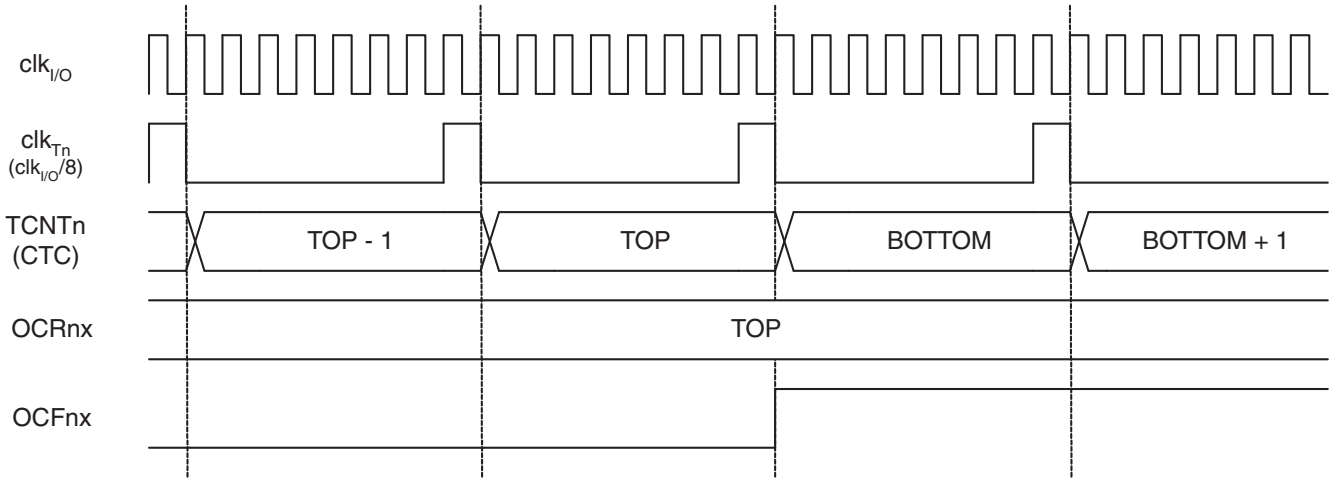


Figure 36 on page 97 shows the setting of OCF0A and the clearing of TCNT0 in CTC mode and fast PWM mode where OCR0A is TOP.

Figure 36. Timer/Counter Timing Diagram, Clear Timer on Compare Match mode, with Prescaler ($f_{clk_I/O}$/8)



11.9 Register Description

11.9.1 TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	**COM0A1**	**COM0A0**	**COM0B1**	**COM0B0**	**–**	**–**	**WGM01**	**WGM00**	**TCCR0A**
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	

- **Bits 7:6 – COM0A[1:0] : Compare Match Output A Mode**

These bits control the Output Compare pin (OC0A) behavior. If one or both of the COM0A[1:0] bits are set, the OC0A output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to the OC0A pin must be set in order to enable the output driver.

When OC0A is connected to the pin, the function of the COM0A[1:0] bits depends on the WGM0[2:0] bit setting. Table 30 shows the COM0A[1:0] bit functionality when the WGM0[2:0] bits are set to a normal or CTC mode (non-PWM).

Table 30. Compare Output Mode, non-PWM Mode

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match
1	1	Set OC0A on Compare Match

Table 31 shows COM0A[1:0] bit functionality when WGM0[2:0] bits are set to fast PWM mode.

Table 31. Compare Output Mode, Fast PWM Mode

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected
0	1	WGM02 = 0: Normal Port Operation, OC0A Disconnected WGM02 = 1: Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match Set OC0A at BOTTOM (non-inverting mode)
1	1	Set OC0A on Compare Match Clear OC0A at BOTTOM (inverting mode)

Note: 1. A special case occurs when OCR0A equals TOP and COM0A1 is set. In this case, the Compare Match is ignored, but the set or clear is done at BOTTOM. See “Fast PWM Mode” on page 93 for more details.

Table 32 shows COM0A[1:0] bit functionality when WGM0[2:0] bits are set to phase correct PWM mode.

Table 32. Compare Output Mode, Phase Correct PWM Mode

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	WGM02 = 0: Normal Port Operation, OC0A Disconnected. WGM02 = 1: Toggle OC0A on Compare Match.
1	0	Clear OC0A on Compare Match when up-counting. Set OC0A on Compare Match when down-counting.
1	1	Set OC0A on Compare Match when up-counting. Clear OC0A on Compare Match when down-counting.

Note: 1. When OCR0A equals TOP and COM0A1 is set, the Compare Match is ignored, but the set or clear is done at TOP. See “Phase Correct PWM Mode” on page 94 for more details.

- **Bits 5:4 – COM0B[1:0] : Compare Match Output B Mode**

These bits control the Output Compare pin (OC0B) behavior. If one or both of COM0B[1:0] bits are set, the OC0B output overrides the normal port functionality of the I/O pin it is connected to. The Data Direction Register (DDR) bit corresponding to the OC0B pin must be set in order to enable the output driver.

When OC0B is connected to the pin, the function of COM0B[1:0] bits depend on WGM0[2:0] bit setting. Table 33 shows COM0B[1:0] bit functionality when WGM0[2:0] bits are set to normal or CTC mode (non-PWM).

Table 33. Compare Output Mode, non-PWM Mode

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected.
0	1	Toggle OC0B on Compare Match
1	0	Clear OC0B on Compare Match
1	1	Set OC0B on Compare Match

Table 34 shows COM0B[1:0] bit functionality when WGM0[2:0] bits are set to fast PWM mode.

Table 34. Compare Output Mode, Fast PWM Mode

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected.
0	1	Reserved
1	0	Clear OC0B on Compare Match, set OC0B at BOTTOM (non-inverting mode)
1	1	Set OC0B on Compare Match, clear OC0B at BOTTOM (inverting mode)

Note: 1. A special case occurs when OCR0B equals TOP and COM0B1 is set. In this case, the Compare Match is ignored, but the set or clear is done at BOTTOM. See “Fast PWM Mode” on page 93 for more details.

Table 35 shows the COM0B[1:0] bit functionality when the WGM0[2:0] bits are set to phase correct PWM mode.

Table 35. Compare Output Mode, Phase Correct PWM Mode

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected.
0	1	Reserved
1	0	Clear OC0B on Compare Match when up-counting. Set OC0B on Compare Match when down-counting.
1	1	Set OC0B on Compare Match when up-counting. Clear OC0B on Compare Match when down-counting.

Note: 1. A special case occurs when OCR0B equals TOP and COM0B1 is set. In this case, the Compare Match is ignored, but the set or clear is done at TOP. See “Phase Correct PWM Mode” on page 94 for more details.

- **Bits 3:2 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bits 1:0 – WGM0[1:0] : Waveform Generation Mode**

Combined with the WGM02 bit found in the TCCR0B Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 36. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare Match (CTC) mode, and two types of Pulse Width Modulation (PWM) modes (see “Modes of Operation” on page 92).

Table 36. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Note: 1. MAX = 0xFF
BOTTOM = 0x00

11.9.2 TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC0A: Force Output Compare A**

The FOC0A bit is only active when the WGM bits specify a non-PWM mode.

However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR0B is written when operating in PWM mode. When writing a logical one to the FOC0A bit, an immediate Compare Match is forced on the Waveform Generation unit. The OC0A output is changed according to its COM0A[1:0] bits setting. Note that the FOC0A bit is implemented as a strobe. Therefore it is the value present in the COM0A[1:0] bits that determines the effect of the forced compare.

A FOC0A strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR0A as TOP.

The FOC0A bit always reads as zero.

- **Bit 6 – FOC0B: Force Output Compare B**

The FOC0B bit is only active when the WGM bits specify a non-PWM mode.

However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR0B is written when operating in PWM mode. When writing a logical one to the FOC0B bit, an immediate Compare Match is forced on the Waveform Generation unit. The OC0B output is changed according to its COM0B[1:0] bits setting. Note that the FOC0B bit is implemented as a strobe. Therefore it is the value present in the COM0B[1:0] bits that determines the effect of the forced compare.

A FOC0B strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR0B as TOP.

The FOC0B bit always reads as zero.

- **Bits 5:4 – Res: Reserved Bits**

These bits are reserved bits in the ATtiny828 and will always read as zero.

- **Bit 3 – WGM02: Waveform Generation Mode**

See the description in the “TCCR0A – Timer/Counter Control Register A” on page 97.

- **Bits 2:0 – CS0[2:0]: Clock Select**

The three Clock Select bits select the clock source to be used by the Timer/Counter.

Table 37. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter0, transitions on the T0 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

11.9.3 TCNT0 – Timer/Counter Register

Bit	7	6	5	4	3	2	1	0	
0x26 (0x46)	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNT0 Register blocks (removes) the Compare Match on the following timer clock. Modifying the

counter (TCNT0) while the counter is running, introduces a risk of missing a Compare Match between TCNT0 and the OCR0x Registers.

11.9.4 OCR0A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0	
0x27 (0x47)	OCR0A[7:0]								OCR0A
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 Output Compare Register A contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC0A pin.

11.9.5 OCR0B – Output Compare Register B

Bit	7	6	5	4	3	2	1	0	
0x28 (0x48)	OCR0B[7:0]								OCR0B
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 Output Compare Register B contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC0B pin.

11.9.6 TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:3 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 2 – OCIE0B: Timer/Counter Output Compare Match B Interrupt Enable**

When the OCIE0B bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter Compare Match B interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter occurs, i.e., when the OCF0B bit is set in the Timer/Counter Interrupt Flag Register – TIFR.

- **Bit 1 – OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable**

When the OCIE0A bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Compare Match A interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter0 occurs, i.e., when the OCF0A bit is set in the Timer/Counter Interrupt Flag Register – TIFR.

- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

When the TOIE0 bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter Interrupt Flag Register – TIFR.

11.9.7 TIFR0 – Timer/Counter Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x15 (0x35)	–	–	–	–	–	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:3 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 2 – OCF0B: Output Compare Flag 0 B**

The OCF0B bit is set when a Compare Match occurs between the Timer/Counter and the data in OCR0B. OCF0B is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF0B is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE0B (Timer/Counter Compare B Match Interrupt Enable), and OCF0B are set, the Timer/Counter Compare Match Interrupt is executed.

- **Bit 1 – OCF0A: Output Compare Flag 0 A**

The OCF0A bit is set when a Compare Match occurs between the Timer/Counter0 and the data in OCR0A – Output Compare Register0. OCF0A is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF0A is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE0A (Timer/Counter0 Compare Match Interrupt Enable), and OCF0A are set, the Timer/Counter0 Compare Match Interrupt is executed.

- **Bit 0 – TOV0: Timer/Counter0 Overflow Flag**

The bit TOV0 is set when an overflow occurs in Timer/Counter0. TOV0 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TOV0 is cleared by writing a logic one to the flag. When the SREG I-bit, TOIE0 (Timer/Counter0 Overflow Interrupt Enable), and TOV0 are set, the Timer/Counter0 Overflow interrupt is executed.

The setting of this flag is dependent of the WGM0[2:0] bit setting. See Table 36 on page 100.

12. 16-bit Timer/Counter1

12.1 Features

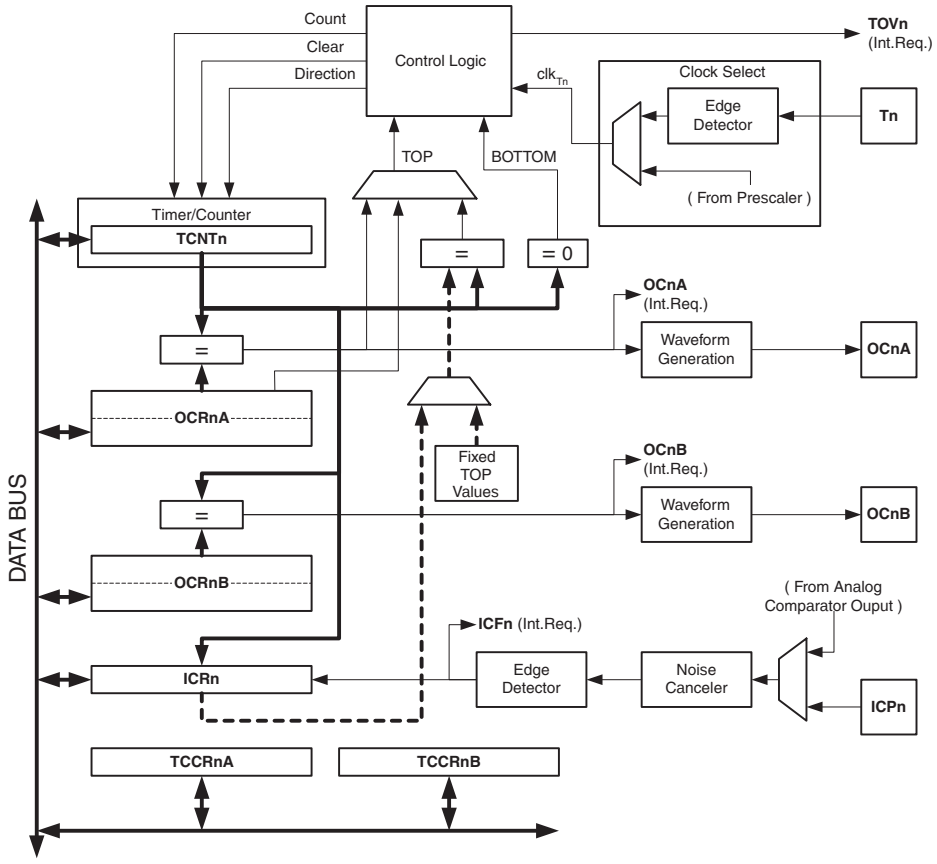
- True 16-bit Design (i.e., Allows 16-bit PWM)
- Two independent Output Compare Units
- Double Buffered Output Compare Registers
- One Input Capture Unit
- Input Capture Noise Canceler
- Clear Timer on Compare Match (Auto Reload)
- Glitch-free, Phase Correct Pulse Width Modulator (PWM)
- Variable PWM Period
- Frequency Generator
- External Event Counter
- Four independent interrupt Sources (TOV1, OCF1A, OCF1B, and ICF1)

12.2 Overview

The 16-bit Timer/Counter unit allows accurate program execution timing (event management), wave generation, and signal timing measurement.

A simplified block diagram of the 16-bit Timer/Counter is shown in Figure 37. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in section “Register Description” on page 123.

Figure 37. 16-bit Timer/Counter Block Diagram



For actual placement of I/O pins, refer to Figure 1 on page 2 (MLF), and Figure 2 on page 2 (TQFP). Also, see "TOCPMSA1 and TOCPMSA0 – Timer/Counter Output Compare Pin Mux Selection Registers" on page 127, and "TOCPMCOE – Timer/Counter Output Compare Pin Mux Channel Output Enable" on page 128.

Most register and bit references in this section are written in general form. A lower case "n" replaces the Timer/Counter number, and a lower case "x" replaces the Output Compare unit channel. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT1 for accessing Timer/Counter1 counter value and so on.

12.2.1 Registers

The Timer/Counter (TCNT1), Output Compare Registers (OCR1A/B), and Input Capture Register (ICR1) are all 16-bit registers. Special procedures must be followed when accessing the 16-bit registers. These procedures are described in section "Accessing 16-bit Registers" on page 120. The Timer/Counter Control Registers (TCCR1A/B) are 8-bit registers and have no CPU access restrictions. Interrupt requests (abbreviated to Int.Req. in the figure) signals are all visible in the Timer Interrupt Flag Register (TIFR). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK). TIFR and TIMSK are not shown in the figure.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the T1 pin. The Clock Select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the Clock Select logic is referred to as the timer clock (clk_{T1}).

The double buffered Output Compare Registers (OCR1A/B) are compared with the Timer/Counter value at all time. The result of the compare can be used by the Waveform Generator to generate a PWM or variable frequency output on the Output Compare pin (OC1A/B). See "Output Compare Units" on page 108. The compare match event will also set the Compare Match Flag (OCF1A/B) which can be used to generate an Output Compare interrupt request.

The Input Capture Register can capture the Timer/Counter value at a given external (edge triggered) event on either the Input Capture pin (ICP1) or on the Analog Comparator pins (See "Analog Comparator" on page 133). The Input Capture unit includes a digital filtering unit (Noise Canceler) for reducing the chance of capturing noise spikes.

The TOP value, or maximum Timer/Counter value, can in some modes of operation be defined by either the OCR1A Register, the ICR1 Register, or by a set of fixed values. When using OCR1A as TOP value in a PWM mode, the OCR1A Register can not be used for generating a PWM output. However, the TOP value will in this case be double buffered allowing the TOP value to be changed in run time. If a fixed TOP value is required, the ICR1 Register can be used as an alternative, freeing the OCR1A to be used as PWM output.

12.2.2 Definitions

The following definitions are used extensively throughout the section:

Table 38. Definitions

Constant	Description
BOTTOM	The counter reaches BOTTOM when it becomes 0x00
MAX	The counter reaches its MAXimum when it becomes 0xFF (decimal 255)
TOP	The counter reaches TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be the fixed value 0xFF (MAX), the value stored in the OCR1A register, or the value stored in the ICR1 register. The assignment depends on the mode of operation

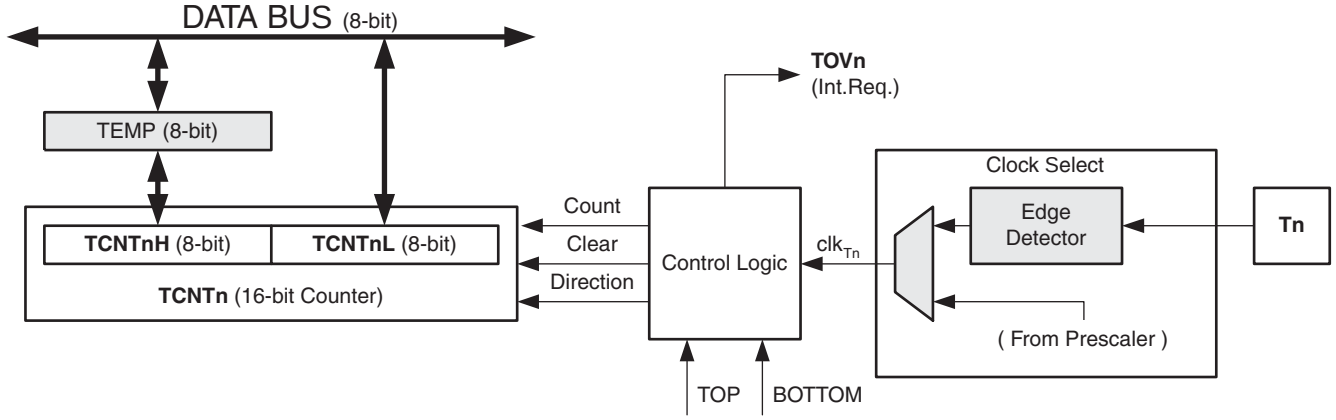
12.3 Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the Clock Select (CS1[2:0]) bits located in the Timer/Counter control Register B (TCCR1B). For details on clock sources and prescaler, see "Timer/Counter Prescaler" on page 131.

12.4 Counter Unit

The main part of the 16-bit Timer/Counter is the programmable 16-bit bi-directional counter unit. Figure 38 shows a block diagram of the counter and its surroundings.

Figure 38. Counter Unit Block Diagram



Description of internal signals used in Figure 38:

Count	Increment or decrement TCNT1 by 1.
Direction	Select between increment and decrement.
Clear	Clear TCNT1 (set all bits to zero).
clk_{T1}	Timer/Counter1 clock.
TOP	Signalize that TCNT1 has reached maximum value.
BOTTOM	Signalize that TCNT1 has reached minimum value (zero).

The 16-bit counter is mapped into two 8-bit I/O memory locations: *Counter High* (TCNT1H) containing the upper eight bits of the counter, and Counter Low (TCNT1L) containing the lower eight bits. The TCNT1H Register can only be indirectly accessed by the CPU. When the CPU does an access to the TCNT1H I/O location, the CPU accesses the high byte temporary register (TEMP). The temporary register is updated with the TCNT1H value when the TCNT1L is read, and TCNT1H is updated with the temporary register value when TCNT1L is written. This allows the CPU to read or write the entire 16-bit counter value within one clock cycle via the 8-bit data bus. It is important to notice that there are special cases of writing to the TCNT1 Register when the counter is counting that will give unpredictable results. The special cases are described in the sections where they are of importance.

Depending on the mode of operation the counter is cleared, incremented, or decremented at each timer clock (clk_{T1}). The clk_{T1} can be generated from an external or internal clock source, selected by the Clock Select bits (CS1[2:0]). When no clock source is selected the timer is stopped. However, the TCNT1 value can be accessed by the CPU, independent of whether clk_{T1} is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the Waveform Generation mode bits (WGM1[3:0]) located in the Timer/Counter Control Registers A and B (TCCR1A and TCCR1B). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare outputs OC1x. For more details about advanced counting sequences and waveform generation, see “Modes of Operation” on page 111.

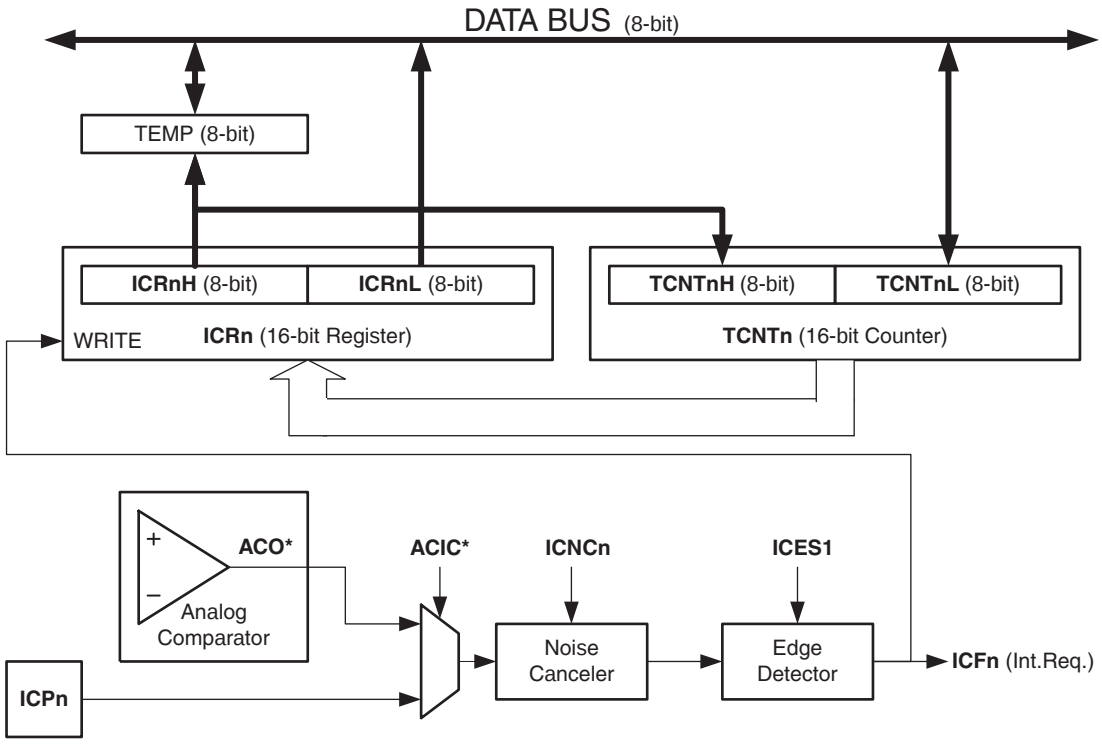
The Timer/Counter Overflow Flag (TOV1) is set according to the mode of operation selected by the WGM1[3:0] bits. TOV1 can be used for generating a CPU interrupt.

12.5 Input Capture Unit

The Timer/Counter incorporates an Input Capture unit that can capture external events and give them a time-stamp indicating time of occurrence. The external signal indicating an event, or multiple events, can be applied via the ICP1 pin

or alternatively, via the analog-comparator unit. The time-stamps can then be used to calculate frequency, duty-cycle, and other features of the signal applied. Alternatively the time-stamps can be used for creating a log of the events.

Figure 39. Input Capture Unit Block Diagram



The Input Capture unit is illustrated by the block diagram shown in Figure 39. The elements of the block diagram that are not directly a part of the Input Capture unit are gray shaded. The small “n” in register and bit names indicates the Timer/Counter number.

When a change of the logic level (an event) occurs on the Input Capture pin (ICP1), alternatively on the Analog Comparator output (ACO), and this change confirms to the setting of the edge detector, a capture will be triggered. When a capture is triggered, the 16-bit value of the counter (TCNT1) is written to the Input Capture Register (ICR1). The Input Capture Flag (ICF1) is set at the same system clock as the TCNT1 value is copied into ICR1 Register. If enabled (ICIE1 = 1), the Input Capture Flag generates an Input Capture interrupt. The ICF1 flag is automatically cleared when the interrupt is executed. Alternatively the ICF1 flag can be cleared by software by writing a logical one to its I/O bit location.

Reading the 16-bit value in the Input Capture Register (ICR1) is done by first reading the low byte (ICR1L) and then the high byte (ICR1H). When the low byte is read the high byte is copied into the high byte temporary register (TEMP). When the CPU reads the ICR1H I/O location it will access the TEMP Register.

The ICR1 Register can only be written when using a Waveform Generation mode that utilizes the ICR1 Register for defining the counter’s TOP value. In these cases the Waveform Generation mode (WGM1[3:0]) bits must be set before the TOP value can be written to the ICR1 Register. When writing the ICR1 Register the high byte must be written to the ICR1H I/O location before the low byte is written to ICR1L.

For more information on how to access the 16-bit registers refer to “Accessing 16-bit Registers” on page 120.

12.5.1 Input Capture Trigger Source

The main trigger source for the Input Capture unit is the Input Capture pin (ICP1). Timer/Counter1 can alternatively use the Analog Comparator output as trigger source for the Input Capture unit. The Analog Comparator is selected as trigger source by setting the Analog Comparator Input Capture (ACIC) bit in the Analog Comparator Control and Status Register (ACSR). Be aware that changing trigger source can trigger a capture. The Input Capture Flag must therefore be cleared after the change.

Both the Input Capture pin (ICP1) and the Analog Comparator output (ACO) inputs are sampled using the same technique as for the T1 pin (Figure 50 on page 131). The edge detector is also identical. However, when the noise canceler is enabled, additional logic is inserted before the edge detector, which increases the delay by four system clock cycles. Note that the input of the noise canceler and edge detector is always enabled unless the Timer/Counter is set in a Waveform Generation mode that uses ICR1 to define TOP.

An Input Capture can be triggered by software by controlling the port of the ICP1 pin.

12.5.2 Noise Canceler

The noise canceler uses a simple digital filtering technique to improve noise immunity. Consecutive samples are monitored in a pipeline four units deep. The signal going to the edge detecter is allowed to change only when all four samples are equal.

The noise canceler is enabled by setting the Input Capture Noise Canceler (ICNC1) bit in Timer/Counter Control Register B (TCCR1B). When enabled, the noise canceler introduces an additional delay of four system clock cycles to a change applied to the input and before ICR1 is updated.

The noise canceler uses the system clock directly and is therefore not affected by the prescaler.

12.5.3 Using the Input Capture Unit

The main challenge when using the Input Capture unit is to assign enough processor capacity for handling the incoming events. The time between two events is critical. If the processor has not read the captured value in the ICR1 Register before the next event occurs, the ICR1 will be overwritten with a new value. In this case the result of the capture will be incorrect.

When using the Input Capture interrupt, the ICR1 Register should be read as early in the interrupt handler routine as possible. Even though the Input Capture interrupt has relatively high priority, the maximum interrupt response time is dependent on the maximum number of clock cycles it takes to handle any of the other interrupt requests.

Using the Input Capture unit in any mode of operation when the TOP value (resolution) is actively changed during operation, is not recommended.

Measurement of an external signal's duty cycle requires that the trigger edge is changed after each capture. Changing the edge sensing must be done as early as possible after the ICR1 Register has been read. After a change of the edge, the Input Capture Flag (ICF1) must be cleared by software (writing a logical one to the I/O bit location). For measuring frequency only, the clearing of the ICF1 flag is not required (if an interrupt handler is used).

12.6 Output Compare Units

The 16-bit comparator continuously compares TCNT1 with the Output Compare Register (OCR1x). If TCNT equals OCR1x the comparator signals a match. A match will set the Output Compare Flag (OCF1x) at the next timer clock cycle. If enabled (OCIE1x = 1), the Output Compare Flag generates an Output Compare interrupt. The OCF1x flag is automatically cleared when the interrupt is executed. Alternatively the OCF1x flag can be cleared by software by writing a logical one to its I/O bit location. The Waveform Generator uses the match signal to generate an output according to operating mode set by the Waveform Generation mode (WGM1[3:0]) bits and Compare Output mode (COM1x[1:0]) bits. The TOP and BOTTOM signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation ("Modes of Operation" on page 111).

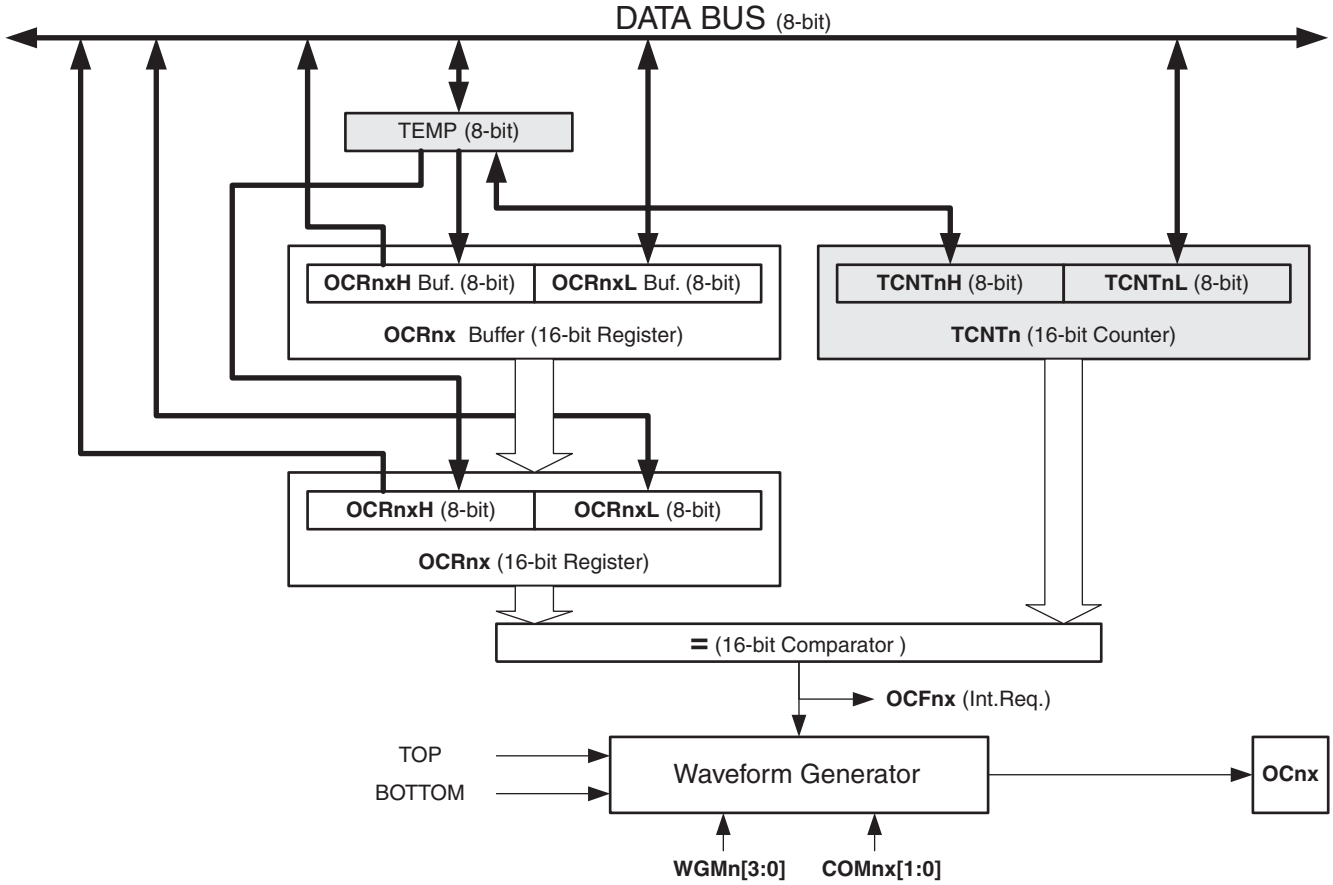
A special feature of Output Compare unit A allows it to define the Timer/Counter TOP value (i.e., counter resolution). In addition to the counter resolution, the TOP value defines the period time for waveforms generated by the Waveform Generator.

Figure 40 on page 109 shows a block diagram of the Output Compare unit. The small "n" in the register and bit names indicates the device number (n = 1 for Timer/Counter 1), and the "x" indicates Output Compare unit (A/B). The elements of the block diagram that are not directly a part of the Output Compare unit are gray shaded.

The OCR1x Register is double buffered when using any of the twelve Pulse Width Modulation (PWM) modes. For the Normal and Clear Timer on Compare (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR1x Compare Register to either TOP or BOTTOM of the counting sequence. The

synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

Figure 40. Output Compare Unit, Block Diagram



The OCR1x Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR1x Buffer Register, and if double buffering is disabled the CPU will access the OCR1x directly. The content of the OCR1x (Buffer or Compare) Register is only changed by a write operation (the Timer/Counter does not update this register automatically as the TCNT1 and ICR1 Register). Therefore OCR1x is not read via the high byte temporary register (TEMP). However, it is a good practice to read the low byte first as when accessing other 16-bit registers. Writing the OCR1x Registers must be done via the TEMP Register since the compare of all 16 bits is done continuously. The high byte (OCR1xH) has to be written first. When the high byte I/O location is written by the CPU, the TEMP Register will be updated by the value written. Then when the low byte (OCR1xL) is written to the lower eight bits, the high byte will be copied into the upper 8-bits of either the OCR1x buffer or OCR1x Compare Register in the same system clock cycle.

For more information of how to access the 16-bit registers refer to “Accessing 16-bit Registers” on page 120.

12.6.1 Force Output Compare

In non-PWM Waveform Generation modes, the match output of the comparator can be forced by writing a one to the Force Output Compare (1x) bit. Forcing compare match will not set the OCF1x flag or reload/clear the timer, but the OC1x pin will be updated as if a real compare match had occurred (the COM1[1:0] bits settings define whether the OC1x pin is set, cleared or toggled).

12.6.2 Compare Match Blocking by TCNT1 Write

All CPU writes to the TCNT1 Register will block any compare match that occurs in the next timer clock cycle, even when the timer is stopped. This feature allows OCR1x to be initialized to the same value as TCNT1 without triggering an interrupt when the Timer/Counter clock is enabled.

12.6.3 Using the Output Compare Unit

Since writing TCNT1 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT1 when using any of the Output Compare channels, independent of whether the Timer/Counter is running or not. If the value written to TCNT1 equals the OCR1x value, the compare match will be missed, resulting in incorrect waveform generation. Do not write the TCNT1 equal to TOP in PWM modes with variable TOP values. The compare match for the TOP will be ignored and the counter will continue to 0xFFFF. Similarly, do not write the TCNT1 value equal to BOTTOM when the counter is downcounting.

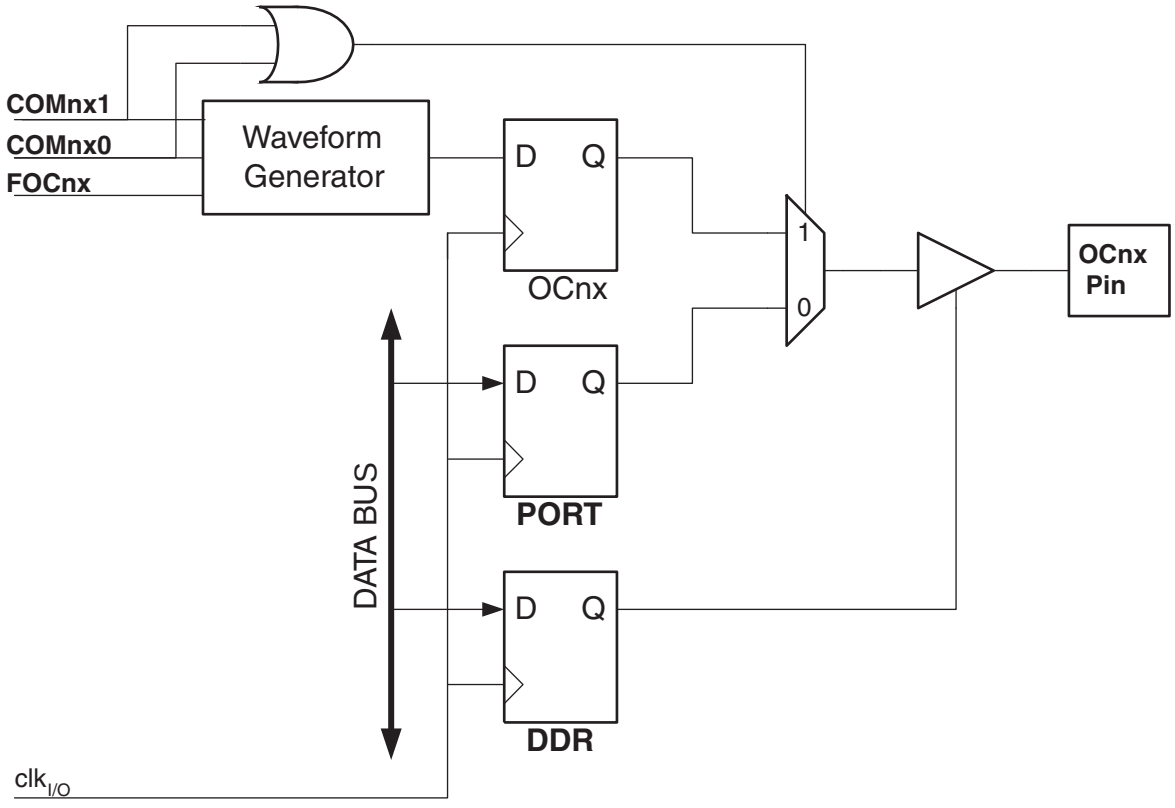
The setup of the OC1x should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC1x value is to use the Force Output Compare (1x) strobe bits in Normal mode. The OC1x Register keeps its value even when changing between Waveform Generation modes.

Be aware that the COM1x[1:0] bits are not double buffered together with the compare value. Changing the COM1x[1:0] bits will take effect immediately.

12.7 Compare Match Output Unit

The Compare Output Mode (COM1x[1:0]) bits have two functions. The Waveform Generator uses the COM1x[1:0] bits for defining the Output Compare (OC1x) state at the next compare match. Secondly the COM1x[1:0] bits control the OC1x pin output source. Figure 41 shows a simplified schematic of the logic affected by the COM1x[1:0] bit setting.

Figure 41. Compare Match Output Unit, Schematic (non-PWM Mode)



The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O port control registers (DDR and PORT) that are affected by the COM1x[1:0] bits are shown. When referring to the OC1x state, the reference is for the internal OC1x Register, not the OC1x pin. If a system reset occur, the OC1x Register is reset to “0”.

The general I/O port function is overridden by the Output Compare (OC1x) from the Waveform Generator if either of the COM1x[1:0] bits are set. However, the OC1x pin direction (input or output) is still controlled by the *Data Direction Register* (DDR) for the port pin. The Data Direction Register bit for the OC1x pin (DDR_OC1x) must be set as output before the OC1x value is visible on the pin. The port override function is generally independent of the Waveform Generation mode, but there are some exceptions. See Table 39 on page 124, Table 40 on page 124 and Table 41 on page 124 for details.

The design of the Output Compare pin logic allows initialization of the OC1x state before the output is enabled. Note that some COM1x[1:0] bit settings are reserved for certain modes of operation. See “Register Description” on page 123

The COM1x[1:0] bits have no effect on the Input Capture unit.

12.7.1 Compare Output Mode and Waveform Generation

The Waveform Generator uses the COM1x[1:0] bits differently in normal, CTC, and PWM modes. For all modes, setting the COM1x[1:0] = 0 tells the Waveform Generator that no action on the OC1x Register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 39 on page 124. For fast PWM mode refer to Table 40 on page 124, and for phase correct and phase and frequency correct PWM refer to Table 41 on page 124.

A change of the COM1x[1:0] bits state will have effect at the first compare match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOC1x strobe bits.

12.8 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the Waveform Generation mode (WGM1[3:0]) and Compare Output mode (COM1x[1:0]) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM1x[1:0] bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM1x[1:0] bits control whether the output should be set, cleared or toggle at a compare match (“Compare Match Output Unit” on page 110)

For detailed timing information refer to “Timer/Counter Timing Diagrams” on page 118.

12.8.1 Normal Mode

The simplest mode of operation is the Normal mode (WGM1[3:0] = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 16-bit value (MAX = 0xFFFF) and then restarts from the BOTTOM (0x0000). In normal operation the Timer/Counter Overflow Flag (TOV1) will be set in the same timer clock cycle as the TCNT1 becomes zero. The TOV1 flag in this case behaves like a 17th bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV1 flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime.

The Input Capture unit is easy to use in Normal mode. However, observe that the maximum interval between the external events must not exceed the resolution of the counter. If the interval between events are too long, the timer overflow interrupt or the prescaler must be used to extend the resolution for the capture unit.

The Output Compare units can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

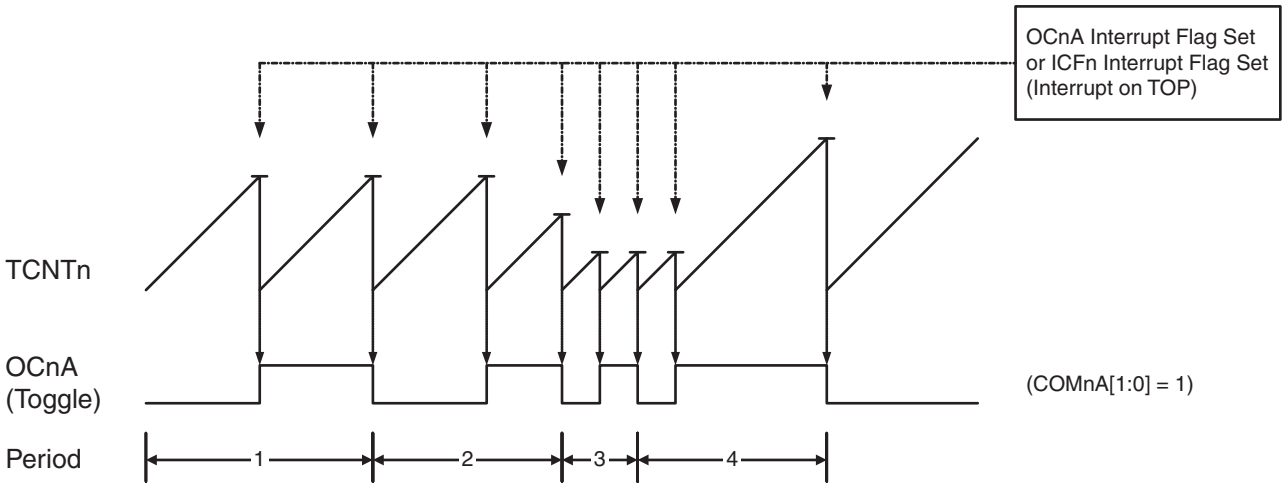
12.8.2 Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGM1[3:0] = 4 or 12), the OCR1A or ICR1 Register are used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT1) matches either the OCR1A (WGM1[3:0] = 4) or the ICR1 (WGM1[3:0] = 12). The OCR1A or ICR1 define the top value for the counter, hence

also its resolution. This mode allows greater control of the compare match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 42 on page 112. The counter value (TCNT1) increases until a compare match occurs with either OCR1A or ICR1, and then counter (TCNT1) is cleared.

Figure 42. CTC Mode, Timing Diagram



An interrupt can be generated at each time the counter value reaches the TOP value by either using the OCF1A or ICF1 flag according to the register used to define the TOP value. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing the TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCR1A or ICR1 is lower than the current value of TCNT1, the counter will miss the compare match. The counter will then have to count to its maximum value (0xFFFF) and wrap around starting at 0x0000 before the compare match can occur. In many cases this feature is not desirable. An alternative will then be to use the fast PWM mode using OCR1A for defining TOP (WGM1[3:0] = 15) since the OCR1A then will be double buffered.

For generating a waveform output in CTC mode, the OC1A output can be set to toggle its logical level on each compare match by setting the Compare Output mode bits to toggle mode (COM1A[1:0] = 1). The OC1A value will not be visible on the port pin unless the data direction for the pin is set to output (DDR_OC1A = 1). The waveform generated will have a maximum frequency of $_{1A} = f_{clk_I/O}/2$ when OCR1A is set to zero (0x0000). The waveform frequency is defined by the following equation:

$$f_{OCnA} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRnA)}$$

The *N* variable represents the prescaler factor (1, 8, 64, 256, or 1024).

As for the Normal mode of operation, the TOV1 flag is set in the same timer clock cycle that the counter counts from MAX to 0x0000.

12.8.3 Fast PWM Mode

The fast Pulse Width Modulation or fast PWM mode (WGM1[3:0] = 5, 6, 7, 14, or 15) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC1x) is cleared on the compare match between TCNT1 and OCR1x, and set at BOTTOM. In inverting Compare Output mode output is set on compare match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct and phase and frequency correct PWM modes that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation,

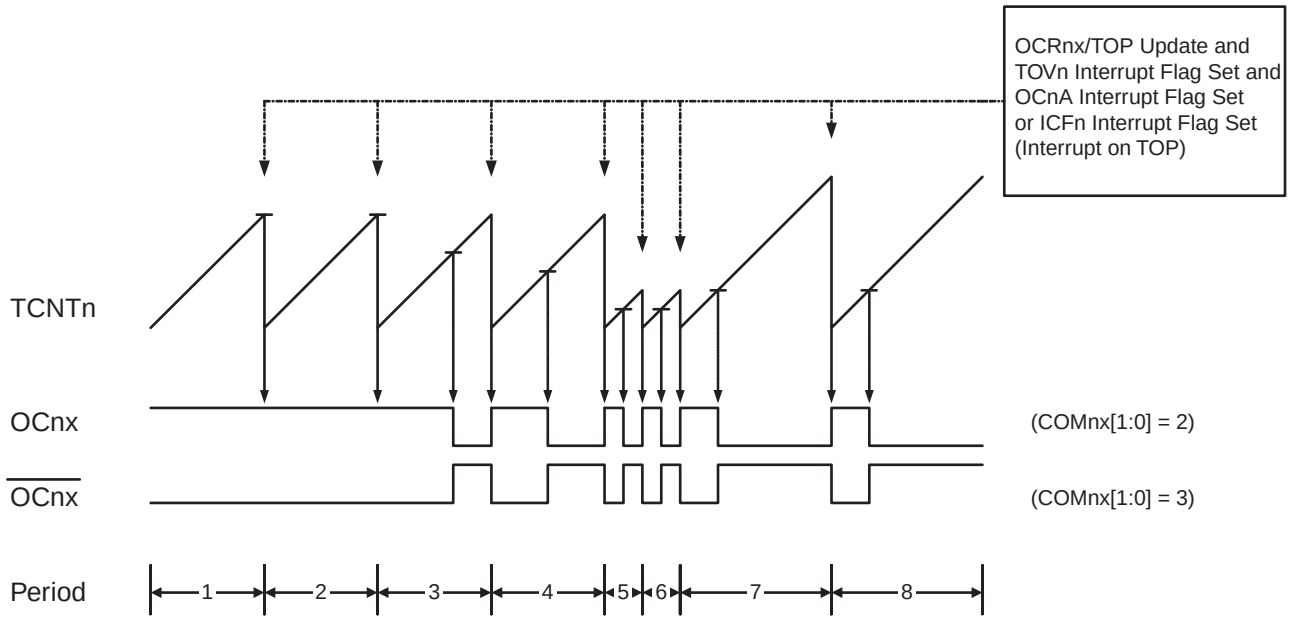
rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), hence reduces total system cost.

The PWM resolution for fast PWM can be fixed to 8-, 9-, or 10-bit, or defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or OCR1A set to 0x0003), and the maximum resolution is 16-bit (ICR1 or OCR1A set to MAX). The PWM resolution in bits can be calculated by using the following equation:

$$R_{FPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In fast PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGM1[3:0] = 5, 6, or 7), the value in ICR1 (WGM1[3:0] = 14), or the value in OCR1A (WGM1[3:0] = 15). The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 43 on page 113. The figure shows fast PWM mode when OCR1A or ICR1 is used to define TOP. The TCNT1 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1x and TCNT1. The OC1x interrupt flag will be set when a compare match occurs.

Figure 43. Fast PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV1) is set each time the counter reaches TOP. In addition the OC1A or ICF1 flag is set at the same timer clock cycle as TOV1 is set when either OCR1A or ICR1 is used for defining the TOP value. If one of the interrupts are enabled, the interrupt handler routine can be used for updating the TOP and compare values.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a compare match will never occur between the TCNT1 and the OCR1x. Note that when using fixed TOP values the unused bits are masked to zero when any of the OCR1x Registers are written.

The procedure for updating ICR1 differs from updating OCR1A when used for defining the TOP value. The ICR1 Register is not double buffered. This means that if ICR1 is changed to a low value when the counter is running with none or a low prescaler value, there is a risk that the new ICR1 value written is lower than the current value of TCNT1. The result will then be that the counter will miss the compare match at the TOP value. The counter will then have to count to the MAX value (0xFFFF) and wrap around starting at 0x0000 before the compare match can occur. The OCR1A Register however, is double buffered. This feature allows the OCR1A I/O location to be written anytime. When the OCR1A I/O location is written the value written will be put into the OCR1A Buffer Register. The OCR1A Compare Register will then

be updated with the value in the Buffer Register at the next timer clock cycle the TCNT1 matches TOP. The update is done at the same timer clock cycle as the TCNT1 is cleared and the TOV1 flag is set.

Using the ICR1 Register for defining TOP works well when using fixed TOP values. By using ICR1, the OCR1A Register is free to be used for generating a PWM output on OC1A. However, if the base PWM frequency is actively changed (by changing the TOP value), using the OCR1A as TOP is clearly a better choice due to its double buffer feature.

In fast PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x[1:0] bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x[1:0] to three (see Table 40 on page 124). The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OC1x). The PWM waveform is generated by setting (or clearing) the OC1x Register at the compare match between OCR1x and TCNT1, and clearing (or setting) the OC1x Register at the timer clock cycle the counter is cleared (changes from TOP to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnxPWM} = \frac{f_{\text{clk_I/O}}}{N \cdot (1 + TOP)}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR1x Register represents special cases when generating a PWM waveform output in the fast PWM mode. If the OCR1x is set equal to BOTTOM (0x0000) the output will be a narrow spike for each TOP+1 timer clock cycle. Setting the OCR1x equal to TOP will result in a constant high or low output (depending on the polarity of the output set by the COM1x[1:0] bits.)

A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC1A to toggle its logical level on each compare match (COM1A[1:0] = 1). The waveform generated will have a maximum frequency of $_{1A}$ = $f_{clk_I/O}/2$ when OCR1A is set to zero (0x0000). This feature is similar to the OC1A toggle in CTC mode, except the double buffer feature of the Output Compare unit is enabled in the fast PWM mode.

12.8.4 Phase Correct PWM Mode

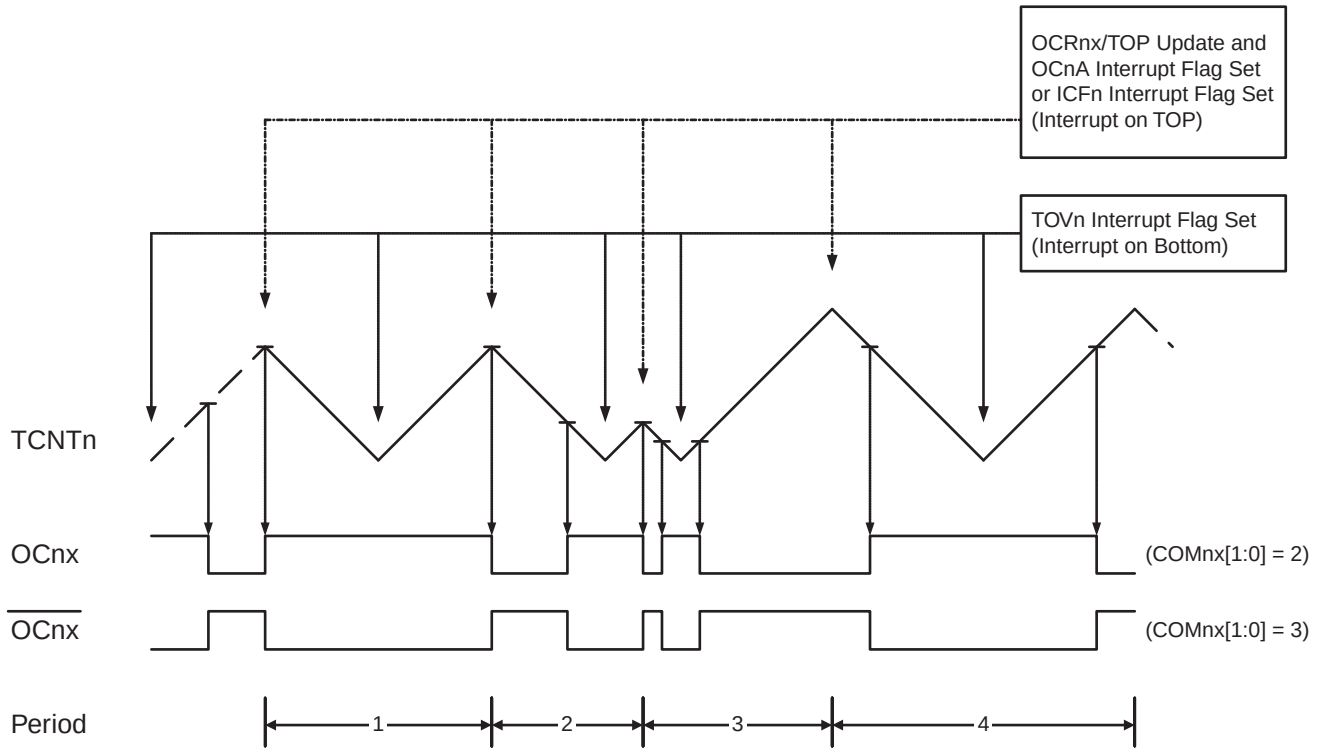
The phase correct Pulse Width Modulation or phase correct PWM mode (WGM1[3:0] = 1, 2, 3, 10, or 11) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is, like the phase and frequency correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC1x) is cleared on the compare match between TCNT1 and OCR1x while upcounting, and set on the compare match while downcounting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

The PWM resolution for the phase correct PWM mode can be fixed to 8-, 9-, or 10-bit, or defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or OCR1A set to 0x0003), and the maximum resolution is 16-bit (ICR1 or OCR1A set to MAX). The PWM resolution in bits can be calculated by using the following equation:

$$R_{PCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In phase correct PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGM1[3:0] = 1, 2, or 3), the value in ICR1 (WGM1[3:0] = 10), or the value in OCR1A (WGM1[3:0] = 11). The counter has then reached the TOP and changes the count direction. The TCNT1 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 44. The figure shows phase correct PWM mode when OCR1A or ICR1 is used to define TOP. The TCNT1 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1x and TCNT1. The OC1x interrupt flag will be set when a compare match occurs.

Figure 44. Phase Correct PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV1) is set each time the counter reaches BOTTOM. When either OCR1A or ICR1 is used for defining the TOP value, the OC1A or ICF1 flag is set accordingly at the same timer clock cycle as the OCR1x Registers are updated with the double buffer value (at TOP). The interrupt flags can be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a compare match will never occur between the TCNT1 and the OCR1x. Note that when using fixed TOP values, the unused bits are masked to zero when any of the OCR1x Registers are written. As the third period shown in Figure 44 illustrates, changing the TOP actively while the Timer/Counter is running in the phase correct mode can result in an unsymmetrical output. The reason for this can be found in the time of update of the OCR1x Register. Since the OCR1x update occurs at TOP, the PWM period starts and ends at TOP. This implies that the length of the falling slope is determined by the previous TOP value, while the length of the rising slope is determined by the new TOP value. When these two values differ the two slopes of the period will differ in length. The difference in length gives the unsymmetrical result on the output.

It is recommended to use the phase and frequency correct mode instead of the phase correct mode when changing the TOP value while the Timer/Counter is running. When using a static TOP value there are practically no differences between the two modes of operation.

In phase correct PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x[1:0] bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x[1:0] to three (See Table 41 on page 124). The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OC1x). The PWM waveform is generated by setting (or clearing) the OC1x Register at the compare match between OCR1x and TCNT1 when the counter increments, and clearing (or setting) the OC1x Register at compare match between OCR1x and TCNT1 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnxPCPWM} = \frac{f_{clk_I/O}}{2 \cdot N \cdot TOP}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR1x Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR1x is set equal to BOTTOM the output will be continuously low and if set equal to TOP the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

12.8.5 Phase and Frequency Correct PWM Mode

The phase and frequency correct Pulse Width Modulation, or phase and frequency correct PWM mode (WGM1[3:0] = 8 or 9) provides a high resolution phase and frequency correct PWM waveform generation option. The phase and frequency correct PWM mode is, like the phase correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC1x) is cleared on the compare match between TCNT1 and OCR1x while upcounting, and set on the compare match while downcounting. In inverting Compare Output mode, the operation is inverted. The dual-slope operation gives a lower maximum operation frequency compared to the single-slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

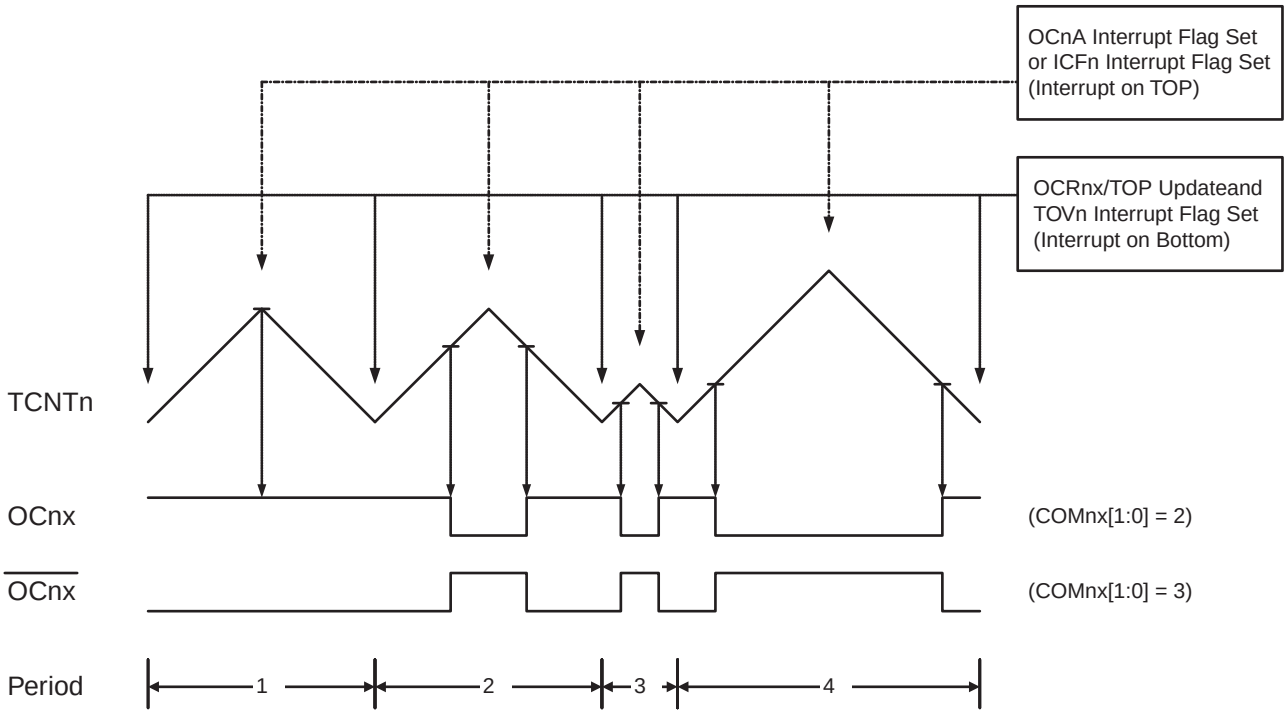
The main difference between the phase correct, and the phase and frequency correct PWM mode is the time the OCR1x Register is updated by the OCR1x Buffer Register, (see Figure 44 on page 115 and Figure 45 on page 117).

The PWM resolution for the phase and frequency correct PWM mode can be defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or OCR1A set to 0x0003), and the maximum resolution is 16-bit (ICR1 or OCR1A set to MAX). The PWM resolution in bits can be calculated using the following equation:

$$R_{PFCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In phase and frequency correct PWM mode the counter is incremented until the counter value matches either the value in ICR1 (WGM1[3:0] = 8), or the value in OCR1A (WGM1[3:0] = 9). The counter has then reached the TOP and changes the count direction. The TCNT1 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct and frequency correct PWM mode is shown on Figure 45. The figure shows phase and frequency correct PWM mode when OCR1A or ICR1 is used to define TOP. The TCNT1 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1x and TCNT1. The OC1x interrupt flag will be set when a compare match occurs.

Figure 45. Phase and Frequency Correct PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV1) is set at the same timer clock cycle as the OCR1x Registers are updated with the double buffer value (at BOTTOM). When either OCR1A or ICR1 is used for defining the TOP value, the OC1A or ICF1 flag set when TCNT1 has reached TOP. The interrupt flags can then be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a compare match will never occur between the TCNT1 and the OCR1x.

As Figure 45 shows the output generated is, in contrast to the phase correct mode, symmetrical in all periods. Since the OCR1x Registers are updated at BOTTOM, the length of the rising and the falling slopes will always be equal. This gives symmetrical output pulses and is therefore frequency correct.

Using the ICR1 Register for defining TOP works well when using fixed TOP values. By using ICR1, the OCR1A Register is free to be used for generating a PWM output on OC1A. However, if the base PWM frequency is actively changed by changing the TOP value, using the OCR1A as TOP is clearly a better choice due to its double buffer feature.

In phase and frequency correct PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x[1:0] bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x[1:0] to three (See Table 41 on page 124). The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OC1x). The PWM waveform is generated by setting (or clearing) the OC1x Register at the compare match between OCR1x and TCNT1 when the counter increments, and clearing (or setting) the OC1x Register at compare match between OCR1x and TCNT1 when the counter decrements. The PWM frequency for the output when using phase and frequency correct PWM can be calculated by the following equation:

$$f_{OCnxPFCPWM} = \frac{f_{clk_I/O}}{2 \cdot N \cdot TOP}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR1x Register represents special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR1x is set equal to BOTTOM the output will be continuously low and if set equal to

TOP the output will be set to high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

12.9 Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock (clk_{T1}) is therefore shown as a clock enable signal in the following figures. The figures include information on when interrupt flags are set, and when the OCR1x Register is updated with the OCR1x buffer value (only for modes utilizing double buffering). Figure 46 shows a timing diagram for the setting of OCF1x.

Figure 46. Timer/Counter Timing Diagram, Setting of OCF1x, no Prescaling

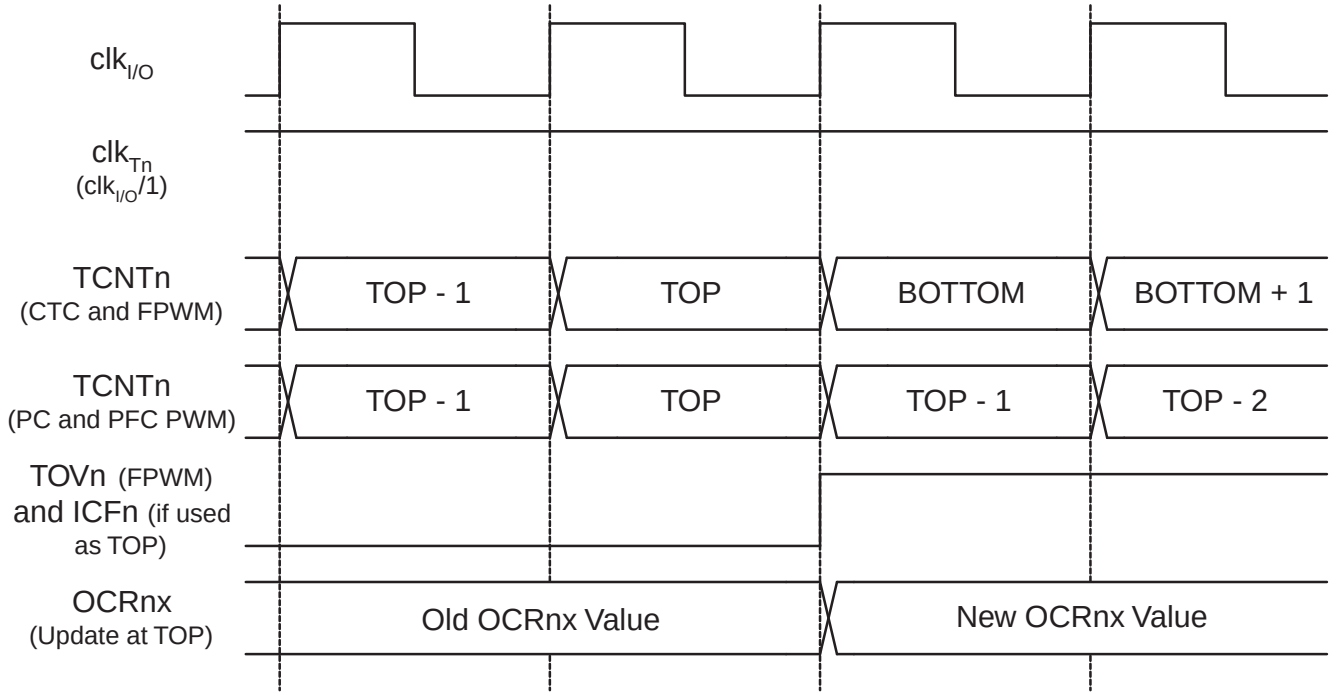


Figure 47 shows the same timing data, but with the prescaler enabled.

Figure 47. Timer/Counter Timing Diagram, Setting of OCF1x, with Prescaler ($f_{clk_I/O}$/8)

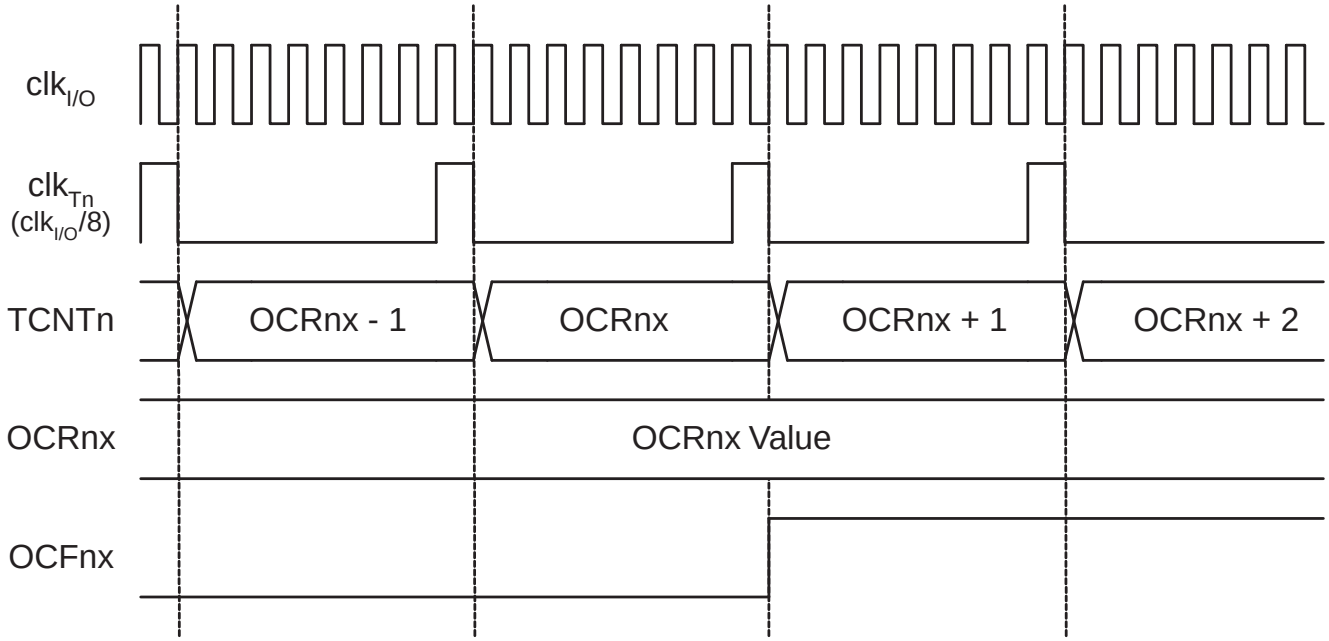


Figure 48 shows the count sequence close to TOP in various modes. When using phase and frequency correct PWM mode the OCR1x Register is updated at BOTTOM. The timing diagrams will be the same, but TOP should be replaced by BOTTOM, TOP-1 by BOTTOM+1 and so on. The same renaming applies for modes that set the TOV1 flag at BOTTOM.

Figure 48. Timer/Counter Timing Diagram, no Prescaling

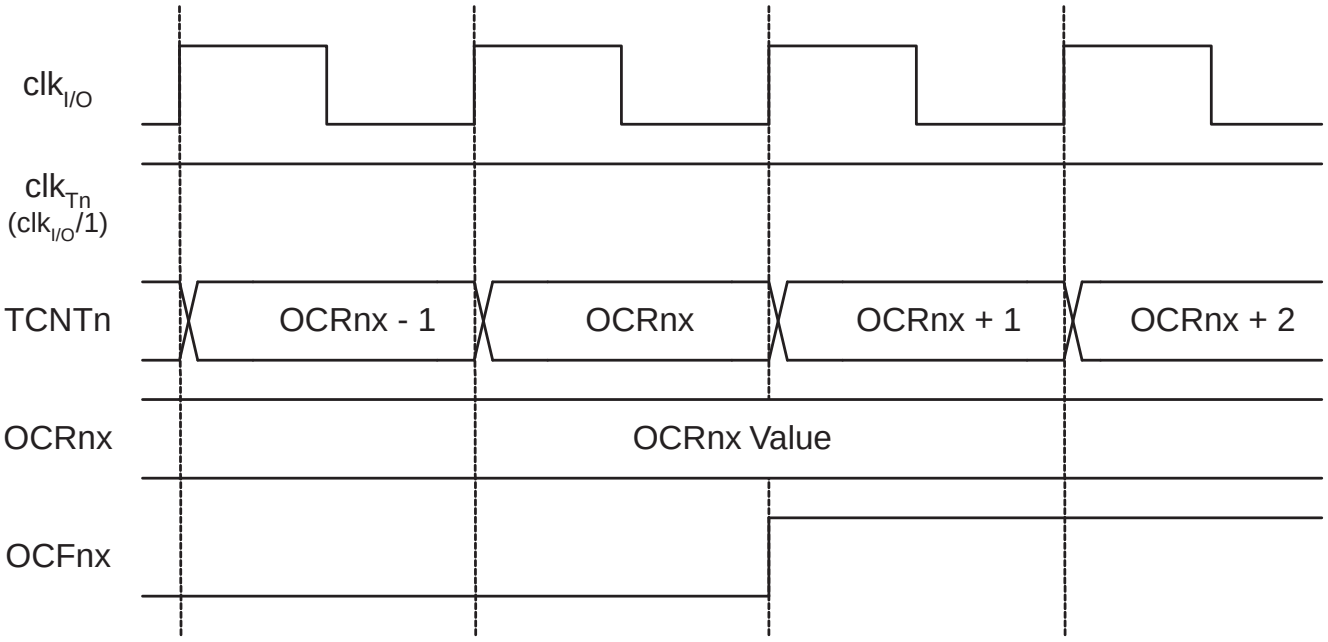
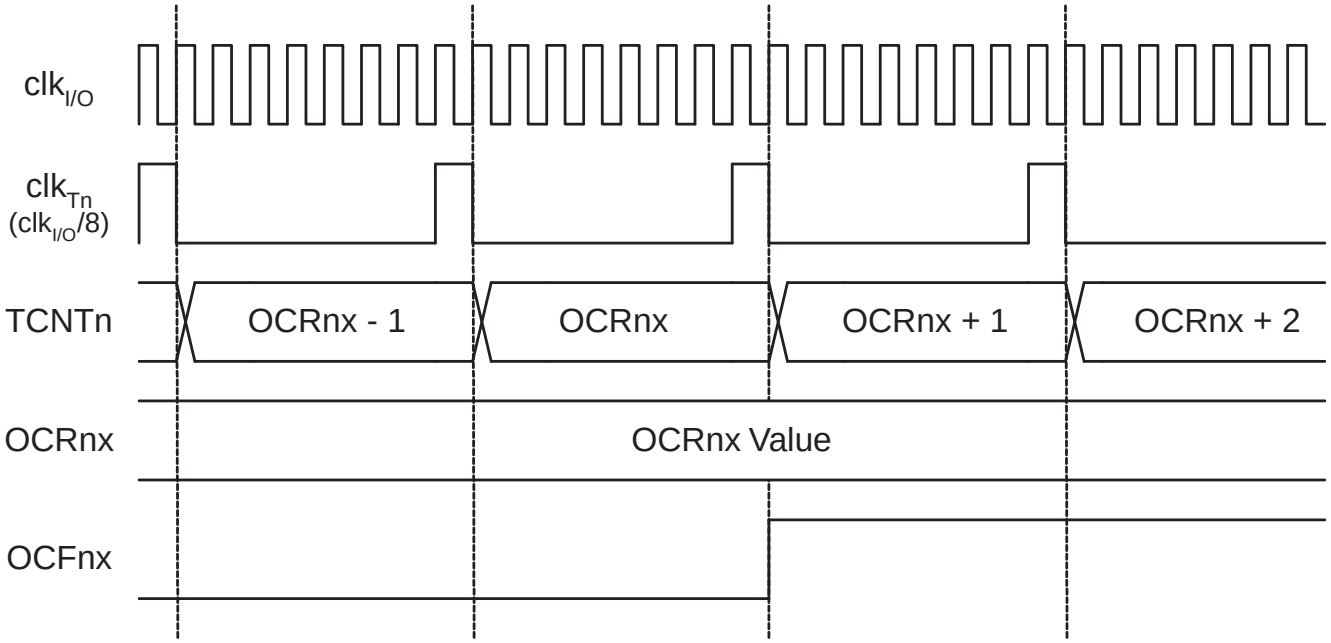


Figure 49 shows the same timing data, but with the prescaler enabled.

Figure 49. Timer/Counter Timing Diagram, with Prescaler ($f_{clk_I/O}$/8)



12.10 Accessing 16-bit Registers

The TCNT1, OCR1A/B, and ICR1 are 16-bit registers that can be accessed by the AVR CPU via the 8-bit data bus. The 16-bit register must be byte accessed using two read or write operations. Each 16-bit timer has a single 8-bit register for temporary storing of the high byte of the 16-bit access. The same temporary register is shared between all 16-bit registers within each 16-bit timer. Accessing the low byte triggers the 16-bit read or write operation. When the low byte of a 16-bit register is written by the CPU, the high byte stored in the temporary register, and the low byte written are both copied into the 16-bit register in the same clock cycle. When the low byte of a 16-bit 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.

Not all 16-bit accesses uses the temporary register for the high byte. Reading the OCR1A/B 16-bit registers does not involve using the temporary register.

To do a 16-bit write, the high byte must be written before the low byte. For a 16-bit read, the low byte must be read before the high byte.

The following code examples show how to access the 16-bit timer registers assuming that no interrupts updates the temporary register. The same principle can be used directly for accessing the OCR1A/B and ICR1 Registers. Note that when using “C”, the compiler handles the 16-bit access.

Assembly Code Example

```
...
; Set TCNT1 to 0x01FF
ldi          r17,0x01
ldi          r16,0xFF
out          TCNT1H,r17
out          TCNT1L,r16

; Read TCNT1 into r17:r16
in           r16,TCNT1L
in           r17,TCNT1H
...
```

C Code Example

```
unsigned int i;
...
/* Set TCNT1 to 0x01FF */
TCNT1 = 0x1FF;

/* Read TCNT1 into i */
i = TCNT1;
...
```

Note: See “Code Examples” on page 7.

The assembly code example returns the TCNT1 value in the r17:r16 register pair.

It is important to notice that accessing 16-bit registers are atomic operations. If an interrupt occurs between the two instructions accessing the 16-bit register, and the interrupt code updates the temporary register by accessing the same or any other of the 16-bit timer registers, then the result of the access outside the interrupt will be corrupted. Therefore, when both the main code and the interrupt code update the temporary register, the main code must disable the interrupts during the 16-bit access.

The following code examples show how to do an atomic read of the TCNT1 Register contents. Reading any of the OCR1A/B or ICR1 Registers can be done by using the same principle.

Assembly Code Example

```
TIM16_ReadTCNT1:
      ; Save global interrupt flag
      in           r18,SREG

      ; Disable interrupts
      cli

      ; Read TCNT1 into r17:r16
      in           r16,TCNT1L
      in           r17,TCNT1H

      ; Restore global interrupt flag
      out          SREG,r18
      ret
```

C Code Example

```
unsigned int TIM16_ReadTCNT1( void )
{
      unsigned char sreg;
      unsigned int i;

      /* Save global interrupt flag */
      sreg = SREG;

      /* Disable interrupts */
      _CLI();

      /* Read TCNT1 into i */
      i = TCNT1;

      /* Restore global interrupt flag */
      SREG = sreg;

      return i;
}
```

Note: See “Code Examples” on page 7.

The assembly code example returns the TCNT1 value in the r17:r16 register pair.

The following code examples show how to do an atomic write of the TCNT1 Register contents. Writing any of the OCR1A/B or ICR1 Registers can be done by using the same principle.

Assembly Code Example

```
TIM16_WriteTCNT1:
      ; Save global interrupt flag
      in          r18,SREG

      ; Disable interrupts
      cli

      ; Set TCNT1 to r17:r16
      out         TCNT1H,r17
      out         TCNT1L,r16

      ; Restore global interrupt flag
      out         SREG,r18

      ret
```

C Code Example

```
void TIM16_WriteTCNT1( unsigned int i )
{
      unsigned char sreg;
      unsigned int i;

      /* Save global interrupt flag */
      sreg = SREG;

      /* Disable interrupts */
      _CLI();

      /* Set TCNT1 to i */
      TCNT1 = i;

      /* Restore global interrupt flag */
      SREG = sreg;
}
```

Note: See “Code Examples” on page 7.

The assembly code example requires that the r17:r16 register pair contains the value to be written to TCNT1.

12.10.1 Reusing the Temporary High Byte Register

If writing to more than one 16-bit register where the high byte is the same for all registers written, then the high byte only needs to be written once. However, note that the same rule of atomic operation described previously also applies in this case.

12.11 Register Description

12.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	**COM1A1**	**COM1A0**	**COM1B1**	**COM1B0**	**–**	**–**	**WGM11**	**WGM10**	**TCCR1A**
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	

- **Bits 7:6 – COM1A[1:0] : Compare Output Mode for Channel A**
- **Bits 5:4 – COM1B[1:0] : Compare Output Mode for Channel B**

The COM1A[1:0] and COM1B[1:0] control the Output Compare pins (OC1A and OC1B respectively) behavior. If one or both of the COM1A[1:0] bits are written to one, the OC1A output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COM1B[1:0] bit are written to one, the OC1B output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to the OC1A or OC1B pin must be set in order to enable the output driver.

When the OC1A or OC1B is connected to the pin, the function of the COM1x[1:0] bits is dependent of the WGM1[3:0] bits setting.

Table 39 shows COM1x[1:0] bit functionality when WGM1[3:0] bits are set to a Normal or a CTC mode (non-PWM).

Table 39. Compare Output Mode, non-PWM

COM1A1 COM1B1	COM1A0 COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected
0	1	Toggle OC1A/OC1B on Compare Match
1	0	Clear OC1A/OC1B on Compare Match (Set output to low level)
1	1	Set OC1A/OC1B on Compare Match (Set output to high level).

Table 40 shows COM1x[1:0] bit functionality when WGM1[3:0] bits are set to fast PWM mode.

Table 40. Compare Output Mode, Fast PWM

COM1A1 COM1B1	COM1A0 COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected
0	1	WGM13=0: Normal port operation, OC1A/OC1B disconnected WGM13=1: Toggle OC1A on Compare Match, OC1B reserved
1	0	Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at BOTTOM (non-inverting mode)
1	1	Set OC1A/OC1B on Compare Match, clear OC1A/OC1B at BOTTOM (inverting mode)

Note: A special case occurs when OCR1A/OCR1B equals TOP and COM1A1/COM1B1 is set. In this case the compare match is ignored, but the set or clear is done at BOTTOM. See “Fast PWM Mode” on page 112 for more details.

Table 41 shows COM1x[1:0] bit functionality when WGM1[3:0] bits are set to phase correct or phase and frequency correct PWM mode.

Table 41. Compare Output Mode, Phase Correct and Phase & Frequency Correct PWM

COM1A1 COM1B1	COM1A0 COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected
0	1	WGM13=0: Normal port operation, OC1A/OC1B disconnected WGM13=1: Toggle OC1A on Compare Match, OC1B reserved
1	0	Clear OC1A/OC1B on Compare Match when up-counting Set OC1A/OC1B on Compare Match when downcounting
1	1	Set OC1A/OC1B on Compare Match when up-counting Clear OC1A/OC1B on Compare Match when downcounting

Note: A special case occurs when OCR1A/OCR1B equals TOP and COM1A1/COM1B1 is set. “Phase Correct PWM Mode” on page 114 for more details.

- **Bits 1:0 – WGM1[1:0]: Waveform Generation Mode**

Combined with the WGM1[3:2] bits found in the TCCR1B Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 42. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes. (“Modes of Operation” on page 111).

Table 42. Waveform Generation Modes

Mode	WGM1[3:0]	Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0000	Normal	0xFFFF	Immediate	MAX
1	0001	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0010	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0011	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0100	CTC (Clear Timer on Compare)	OCR1A	Immediate	MAX
5	0101	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0110	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0111	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1000	PWM, Phase & Freq. Correct	ICR1	BOTTOM	BOTTOM
9	1001	PWM, Phase & Freq. Correct	OCR1A	BOTTOM	BOTTOM
10	1010	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1011	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1100	CTC (Clear Timer on Compare)	ICR1	Immediate	MAX
13	1101	(Reserved)	–	–	–
14	1110	Fast PWM	ICR1	TOP	TOP
15	1111	Fast PWM	OCR1A	TOP	TOP

12.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ICNC1: Input Capture Noise Canceler**

Setting this bit (to one) activates the Input Capture Noise Canceler. When the noise canceler is activated, the input from the Input Capture pin (ICP1) is filtered. The filter function requires four successive equal valued samples of the ICP1 pin for changing its output. The Input Capture is therefore delayed by four Oscillator cycles when the noise canceler is enabled.

- **Bit 6 – ICES1: Input Capture Edge Select**

This bit selects which edge on the Input Capture pin (ICP1) that is used to trigger a capture event. When the ICES1 bit is written to zero, a falling (negative) edge is used as trigger, and when the ICES1 bit is written to one, a rising (positive) edge will trigger the capture.

When a capture is triggered according to the ICES1 setting, the counter value is copied into the Input Capture Register (ICR1). The event will also set the Input Capture Flag (ICF1), and this can be used to cause an Input Capture Interrupt, if this interrupt is enabled.

When the ICR1 is used as TOP value (see description of the WGM1[3:0] bits located in the TCCR1A and the TCCR1B Register), the ICP1 is disconnected and consequently the Input Capture function is disabled.

- **Bit 5 – Res: Reserved Bit**

This bit is reserved for future use. For ensuring compatibility with future devices, this bit must be written to zero when register is written.

- **Bits 4:3 – WGM1[3:2] : Waveform Generation Mode**

See TCCR1A Register description.

- **Bits 2:0 – CS1[2:0]: Clock Select**

The three Clock Select bits select the clock source to be used by the Timer/Counter, see Figure 46 on page 118 and Figure 47 on page 119.

Table 43. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter1, transitions on the T1 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

12.11.3 TCCR1C – Timer/Counter1 Control Register C

Bit	7	6	5	4	3	2	1	0	
(0x82)	FOC1A	FOC1B	–	–	–	–	–	–	TCCR1C
Read/Write	W	W	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC1A: Force Output Compare for Channel A**
- **Bit 6 – FOC1B: Force Output Compare for Channel B**

The FOC1A/FOC1B bits are only active when the WGM1[3:0] bits specifies a non-PWM mode. However, for ensuring compatibility with future devices, these bits must be set to zero when TCCR1A is written when operating in a PWM mode. When writing a logical one to the FOC1A/FOC1B bit, an immediate compare match is forced on the Waveform Generation unit. The OC1A/OC1B output is changed according to its COM1x[1:0] bits setting. Note that the FOC1A/FOC1B bits are implemented as strobes. Therefore it is the value present in the COM1x[1:0] bits that determine the effect of the forced compare.

A FOC1A/FOC1B strobe will not generate any interrupt nor will it clear the timer in Clear Timer on Compare match (CTC) mode using OCR1A as TOP.

The FOC1A/FOC1B bits are always read as zero.

- **Bits 5:0 – Res: Reserved Bit**

These bits are reserved for future use. To ensure compatibility with future devices, these bits must be set to zero when the register is written.

12.11.4 TOCPMSA1 and TOCPMSA0 – Timer/Counter Output Compare Pin Mux Selection Registers

Bit	7	6	5	4	3	2	1	0	
(0xE9)	TOCC7S1	TOCC7S0	TOCC6S1	TOCC6S0	TOCC5S1	TOCC5S0	TOCC4S1	TOCC4S0	TOCPMSA1
(0xE8)	TOCC3S1	TOCC3S0	TOCC2S1	TOCC2S0	TOCC1S1	TOCC1S0	TOCC0S1	TOCC0S0	TOCPMSA0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – TOCCnS1 and TOCCnS0: Timer/Counter Output Compare Channel Select**

TOCCnS1 and TOCCnS bits select which Timer/Counter compare output is routed to the corresponding TOCCn pin. The two timer/counters provide four possible compare outputs that can be routed to output pins, as shown in the table below.

Table 44. Selecting Timer/Counter Compare Output for TOCCn Pins

TOCCnS1	TOCCnS0	TOCCn Output (1)
0	0	OC0A
0	1	OC0B
1	0	OC1A
1	1	OC1B

Note: 1. See “Alternative Functions of Port C” on page 73.

12.11.5 TOCPMCOE – Timer/Counter Output Compare Pin Mux Channel Output Enable

Bit	7	6	5	4	3	2	1	0	
(0xE2)	TOCC7OE	TOCC6OE	TOCC5OE	TOCC4OE	TOCC3OE	TOCC2OE	TOCC1OE	TOCC0OE	TOCPMCOE
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – TOCCnOE: Timer/Counter Output Compare Channel Output Enable**

These bits enable the selected output compare channel on the corresponding TOCCn pin, regardless if the output compare mode is selected, or not.

12.11.6 TCNT1H and TCNT1L – Timer/Counter1

Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1[15:8]								TCNT1H
(0x84)	TCNT1[7:0]								TCNT1L
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 two Timer/Counter I/O locations (TCNT1H and TCNT1L, combined TCNT1) give direct access, both for read and for write operations, to the Timer/Counter unit 16-bit counter. To ensure that both the high and low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See “Accessing 16-bit Registers” on page 120.

Modifying the counter (TCNT1) while the counter is running introduces a risk of missing a compare match between TCNT1 and one of the OCR1x Registers.

Writing to the TCNT1 Register blocks (removes) the compare match on the following timer clock for all compare units.

12.11.7 OCR1AH and OCR1AL – Output Compare Register 1 A

Bit	7	6	5	4	3	2	1	0	
(0x89)	OCR1A[15:8]								OCR1AH
(0x88)	OCR1A[7:0]								OCR1AL
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	

12.11.8 OCR1BH and OCR1BL – Output Compare Register 1 B

Bit	7	6	5	4	3	2	1	0	
(0x8B)	OCR1B[15:8]								OCR1BH
(0x8A)	OCR1B[7:0]								OCR1BL
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 Output Compare Registers contain a 16-bit value that is continuously compared with the counter value (TCNT1). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC1x pin.

The Output Compare Registers are 16-bit in size. To ensure that both the high and low bytes are written simultaneously when the CPU writes to these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See “Accessing 16-bit Registers” on page 120.

12.11.9 ICR1H and ICR1L – Input Capture Register 1

Bit	7	6	5	4	3	2	1	0	
(0x87)	ICR1[15:8]								ICR1H
(0x86)	ICR1[7:0]								ICR1L
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 Input Capture is updated with the counter (TCNT1) value each time an event occurs on the ICP1 pin (or optionally on the Analog Comparator output for Timer/Counter1). The Input Capture can be used for defining the counter TOP value.

The Input Capture Register is 16-bit in size. To ensure that both the high and low bytes are read simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. “Accessing 16-bit Registers” on page 120.

12.11.10TIMSK1 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7, 6, 4, 3 – Res: Reserved Bit**

These bits are reserved for future use. To ensure compatibility with future devices, these bits must be set to zero when the register is written.

- **Bit 5 – ICIE1: Timer/Counter1, Input Capture Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Countern Input Capture interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 66.) is executed when the ICF1 Flag, located in TIFR, is set.

- **Bit 2 – OCIE1B: Timer/Counter1, Output Compare B Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare B Match interrupt is enabled. The corresponding Interrupt Vector (see “Interrupts” on page 48) is executed when the OCF1B flag, located in TIFR, is set.

- **Bit 1 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare A Match interrupt is enabled. The corresponding Interrupt Vector (see “Interrupts” on page 48) is executed when the OCF1A flag, located in TIFR, is set.

- **Bit 0 – TOIE1: Timer/Counter1, Overflow Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Overflow interrupt is enabled. The corresponding Interrupt Vector (see “Interrupts” on page 48) is executed when the TOV1 flag, located in TIFR, is set.

12.11.11TIFR1 – Timer/Counter Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7, 6, 4, 3 – Res: Reserved Bit**

These bits are reserved for future use. To ensure compatibility with future devices, these bits must be set to zero when the register is written.

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

This flag is set when a capture event occurs on the ICP1 pin. When the Input Capture Register (ICR1) is set by the WGM1[3:0] to be used as the TOP value, the ICF1 flag is set when the counter reaches the TOP value.

ICF1 is automatically cleared when the Input Capture Interrupt Vector is executed. Alternatively, ICF1 can be cleared by writing a logic one to its bit location.

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

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the Output Compare Register B (OCR1B).

Note that a Forced Output Compare (1B) strobe will not set the OCF1B flag.

OCF1B is automatically cleared when the Output Compare Match B Interrupt Vector is executed. Alternatively, OCF1B can be cleared by writing a logic one to its bit location.

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

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the Output Compare Register A (OCR1A).

Note that a Forced Output Compare (1A) strobe will not set the OCF1A flag.

OCF1A is automatically cleared when the Output Compare Match A Interrupt Vector is executed. Alternatively, OCF1A can be cleared by writing a logic one to its bit location.

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

The setting of this flag is dependent of the WGM1[3:0] bits setting. In Normal and CTC modes, the TOV1 flag is set when the timer overflows. See Table 42 on page 125 for the TOV1 flag behavior when using another WGM1[3:0] bit setting.

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

13. Timer/Counter Prescaler

Timer/Counter0 and Timer/Counter1 share the same prescaler module, but the Timer/Counters can have different prescaler settings. The description below applies to both Timer/Counters. Tn is used as a general name, n = 0, 1.

The Timer/Counter can be clocked directly by the system clock (by setting the CSn[2:0] = 1). This provides the fastest operation, with a maximum Timer/Counter clock frequency equal to system clock frequency ($f_{CLK_I/O}$). Alternatively, one of four taps from the prescaler can be used as a clock source. The prescaled clock has a frequency of either $f_{CLK_I/O}/8$, $f_{CLK_I/O}/64$, $f_{CLK_I/O}/256$, or $f_{CLK_I/O}/1024$.

13.1 Prescaler Reset

The prescaler is free running, i.e., operates independently of the Clock Select logic of the Timer/CounterCounter, and it is shared by the Timer/Counter Tn. Since the prescaler is not affected by the Timer/Counter's clock select, the state of the prescaler will have implications for situations where a prescaled clock is used. One example of prescaling artifacts occurs when the timer is enabled and clocked by the prescaler (CSn[2:0] = 2, 3, 4, or 5). The number of system clock cycles from when the timer is enabled to the first count occurs can be from 1 to N+1 system clock cycles, where N equals the prescaler divisor (8, 64, 256, or 1024).

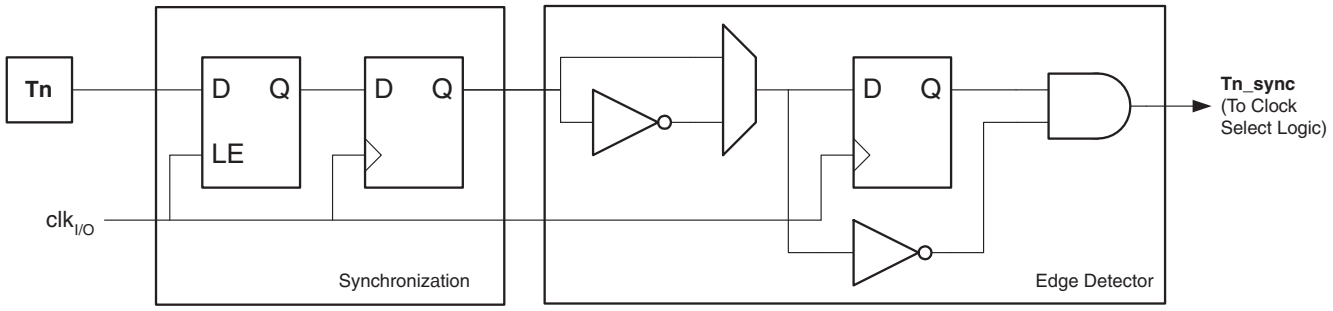
It is possible to use the Prescaler Reset for synchronizing the Timer/Counter to program execution.

13.2 External Clock Source

An external clock source applied to the Tn pin can be used as Timer/Counter clock (clk_{Tn}). The Tn pin is sampled once every system clock cycle by the pin synchronization logic. The synchronized (sampled) signal is then passed through the edge detector. Figure 50 shows a functional equivalent block diagram of the Tn synchronization and edge detector logic. The registers are clocked at the positive edge of the internal system clock ($clk_{I/O}$). The latch is transparent in the high period of the internal system clock.

The edge detector generates one clk_{T0} pulse for each positive (CSn[2:0] = 7) or negative (CSn[2:0] = 6) edge it detects.

Figure 50. T0 Pin Sampling



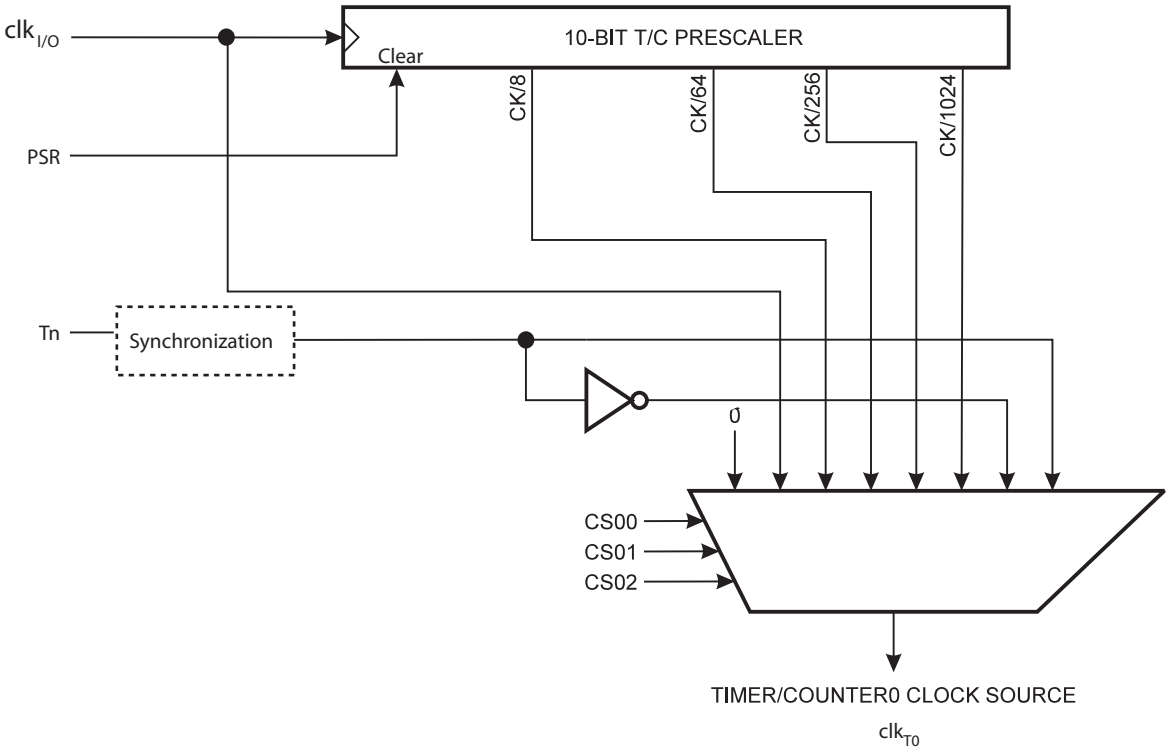
The synchronization and edge detector logic introduces a delay of 2.5 to 3.5 system clock cycles from an edge has been applied to the Tn pin to the counter is updated.

Enabling and disabling of the clock input must be done when Tn has been stable for at least one system clock cycle, otherwise it is a risk that a false Timer/Counter clock pulse is generated.

Each half period of the external clock applied must be longer than one system clock cycle to ensure correct sampling. The external clock must be guaranteed to have less than half the system clock frequency ($f_{ExtClk} < f_{clk_I/O}/2$) given a 50/50% duty cycle. Since the edge detector uses sampling, the maximum frequency of an external clock it can detect is half the sampling frequency (Nyquist sampling theorem). However, due to variation of the system clock frequency and duty cycle caused by oscillator source tolerances, it is recommended that maximum frequency of an external clock source is less than $f_{clk_I/O}/2.5$.

An external clock source can not be prescaled.

Figure 51. Prescaler for Timer/Counter0



Note: 1. The synchronization logic on the input pins (T0) is shown in Figure 50 on page 131.

13.3 Register Description

13.3.1 GTCCR – General Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	
0x23 (0x43)	**TSM**	–	–	–	–	–	–	**PSR**	**GTCCR**
Read/Write	R/W	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – TSM: Timer/Counter Synchronization Mode**

Writing the TSM bit to one activates the Timer/Counter Synchronization mode. In this mode, the value that is written to the PSR bit is kept, hence keeping the Prescaler Reset signal asserted. This ensures that the Timer/Counter is halted and can be configured without the risk of advancing during configuration.

When the TSM bit is written to zero, the PSR bit is cleared by hardware, and the Timer/Counter starts counting.

- **Bit 0 – PSR: Prescaler Reset**

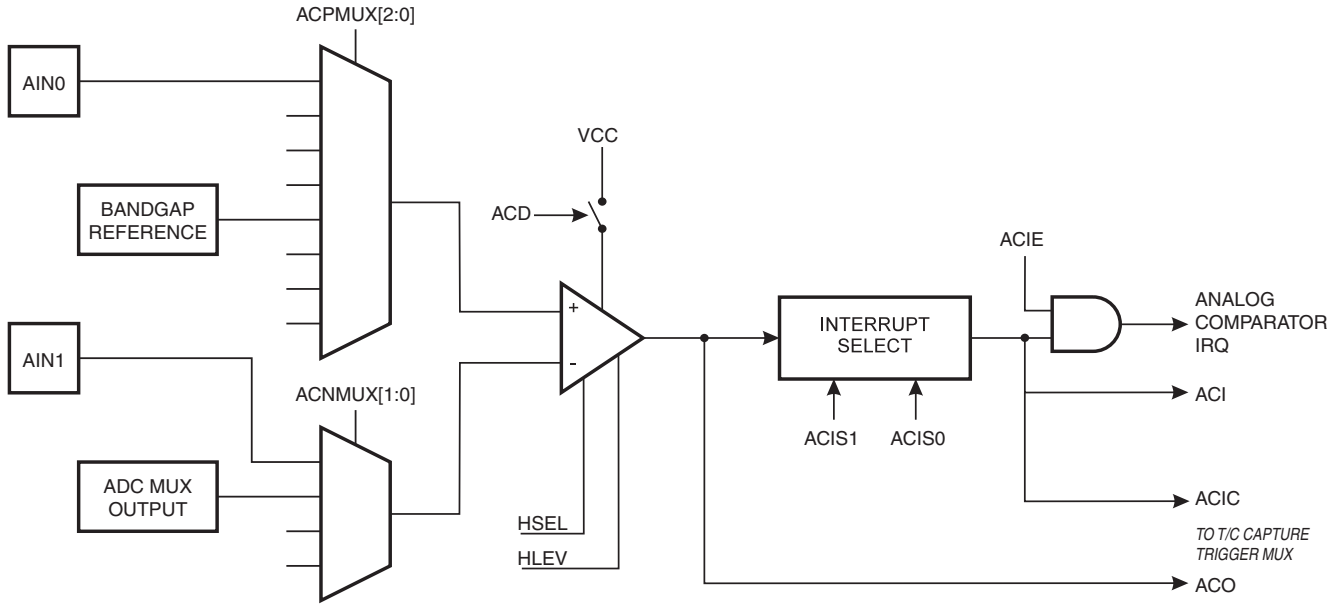
When this bit is one, the Timer/Counter prescaler will be Reset. This bit is normally cleared immediately by hardware, except if the TSM bit is set.

14. Analog Comparator

The analog comparator compares the input values on the positive pin (AIN0) and negative pin (AIN1). When the voltage on AIN0 is higher than the voltage on AIN1, the Analog Comparator Output, ACO, is set. The comparator can trigger a separate interrupt, exclusive to the analog comparator. The user can select Interrupt triggering on comparator output rise, fall or toggle.

A block diagram of the comparator and its surrounding logic is shown in Figure 52.

Figure 52. Analog Comparator Block Diagram



Notes: 1. See Table 51 on page 149.

For pin placements, see Figure 1 on page 2.

The following options are available for positive input signals to the analog comparator:

- AIN0 pin
- Bandgap reference voltage

The following options are available for negative input signals to the analog comparator:

- AIN1 pin
- ADC multiplexer output

The ADC Power Reduction bit must be disabled in order to use the ADC input multiplexer. This is done by clearing the PRADC bit in the Power Reduction Register. See "PRR – Power Reduction Register" on page 37 for more details.

14.1 Register Description

14.1.1 ACSRA – Analog Comparator Control and Status Register

Bit	7	6	5	4	3	2	1	0	
0x30 (0x50)	ACD	ACPMUX2	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSRA
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

- **Bit 7 – ACD: Analog Comparator Disable**

When this bit is written logic one, the power to the analog comparator is switched off. This bit can be set at any time to turn off the analog comparator. This will reduce power consumption in Active and Idle mode.

When changing this bit, the analog comparator Interrupt must be disabled (see ACIE bit). Otherwise, an interrupt can occur when the bit is changed.

- **Bit 6 – ACPMUX2: Analog Comparator Positive Input Multiplexer**

Together with ACPMUX1 and ACPMUX0, these bits select the source for the positive input of the analog converter. See “ACSRB – Analog Comparator Control and Status Register B” on page 135.

- **Bit 5 – ACO: Analog Comparator Output**

The output of the analog comparator is synchronized and then directly connected to this bit. The synchronization introduces a delay of 1 - 2 clock cycles.

- **Bit 4 – ACI: Analog Comparator Interrupt Flag**

This bit is set by hardware when a comparator output event triggers the interrupt mode defined by ACIS1 and ACIS0. The analog comparator interrupt routine is executed if the ACIE bit is set and the I-bit in SREG is set. ACI is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ACI is cleared by writing a logic one to the flag.

- **Bit 3 – ACIE: Analog Comparator Interrupt Enable**

When this bit is set and the I-bit in the Status Register is set, the analog comparator interrupt is activated. When this bit is cleared the interrupt is disabled.

- **Bit 2 – ACIC: Analog Comparator Input Capture Enable**

When this bit is set the input capture function of Timer/Counter1 can be triggered by the analog comparator. The comparator output is then directly connected to the input capture front-end logic, making the comparator utilize the noise canceler and edge select features of the Timer/Counter1 input capture interrupt. To make the comparator trigger the Timer/Counter1 Input Capture interrupt the ICIE1 bit must be set (see “TIMSK1 – Timer/Counter Interrupt Mask Register” on page 129).

When this bit is cleared, no connection between the analog comparator and the input capture function exists.

- **Bits 1:0 – ACIS[1:0]: Analog Comparator Interrupt Mode Select**

These bits determine which comparator events that trigger the analog comparator interrupt. The different settings are shown in Table 45.

Table 45. ACIS1/ACIS0 Settings

ACIS1	ACIS0	Interrupt Mode
0	0	Comparator Interrupt on Output Toggle.
0	1	Reserved
1	0	Comparator Interrupt on Falling Output Edge.
1	1	Comparator Interrupt on Rising Output Edge.

When changing these bits, the analog comparator interrupt must be disabled. Otherwise, an interrupt can occur when the bits are changed.

14.1.2 ACSRB – Analog Comparator Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
0x2F (0x4F)	**HSEL**	**HLEV**	**–**	**–**	**ACNMUX1**	**ACNMUX0**	**ACPMUX1**	**ACPMUX0**	**ACSRB**
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 – HSEL: Hysteresis Select**

When this bit is written logic one, the hysteresis of the analog comparator is enabled. The level of hysteresis is selected by the HLEV bit.

- **Bit 6 – HLEV: Hysteresis Level**

When enabled via the HSEL bit, the level of hysteresis can be set using the HLEV bit, as shown in Table 46.

Table 46. Selecting Level of Analog Comparator Hysteresis

HSEL	HLEV	Hysteresis of Analog Comparator
0	X	Not enabled
1	0	20 mV
	1	50 mV

- **Bit 4 – Res: Reserved Bit**

This bit is reserved and will always read as zero.

- **Bits 3:2 – ACNMUX[1:0]: Analog Comparator Negative Input Multiplexer**

These bits select the source for the negative input of the analog comparator, as shown in Table 47, below.

Table 47. Source Selection for Analog Comparator Negative Input

ACNMUX1	ACNMUX0	Analog Comparator Negative Input
0	0	AIN1 pin
0	1	Output of ADC multiplexer
1	0	Reserved
1	1	

- **Bits 1:0 – ACPMUX[1:0]: Analog Comparator Positive Input Multiplexer**

Together with ACPMUX2, these bits select the source for the positive input of the analog comparator, as shown in Table 48, below.

Table 48. Source Selection for Analog Comparator Positive Input

ACPMUX2	ACPMUX1	ACPMUX0	Analog Comparator Positive Input
0	0	0	AIN0 pin
0	0	1	Reserved
0	1	0	Reserved
0	1	1	Reserved
1	0	0	Internal bandgap reference voltage
1	0	1	Reserved
1	1	0	Reserved
1	1	1	Reserved

14.1.3 DIDR0 – Digital Input Disable Register 0

Bit	7	6	5	4	3	2	1	0	
(0x7E)	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	DIDR0
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 2 – ADC2D: ADC2/AIN1 Digital input buffer disable**

When used as an analog input but not required as a digital input the power consumption of the digital input buffer can be reduced by writing this bit to logic one. When this bit is set, the digital input buffer on the AIN1 pin is disabled and the corresponding pin register bit (PA2) will always read zero.

- **Bits 1 – ADC1D: ADC1/AIN0 Digital input buffer disable**

When used as an analog input but not required as a digital input the power consumption of the digital input buffer can be reduced by writing this bit to logic one. When this bit is set, the digital input buffer on the AIN0 pin is disabled and the corresponding pin register bit (PA1) will always read zero.

15. Analog to Digital Converter

15.1 Features

- 10-bit Resolution
- 1 LSB Integral Non-linearity
- ± 2 LSB Absolute Accuracy
- 15µs Conversion Time
- 15 kSPS at Maximum Resolution
- 28 Multiplexed Single Ended Input Channels
- Temperature Sensor Input Channel
- Optional Left Adjustment for ADC Result Readout
- 0 - V_{CC} ADC Input Voltage Range
- 1.1V ADC Reference Voltage
- Free Running or Single Conversion Mode
- ADC Start Conversion by Auto Triggering on Interrupt Sources
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceler

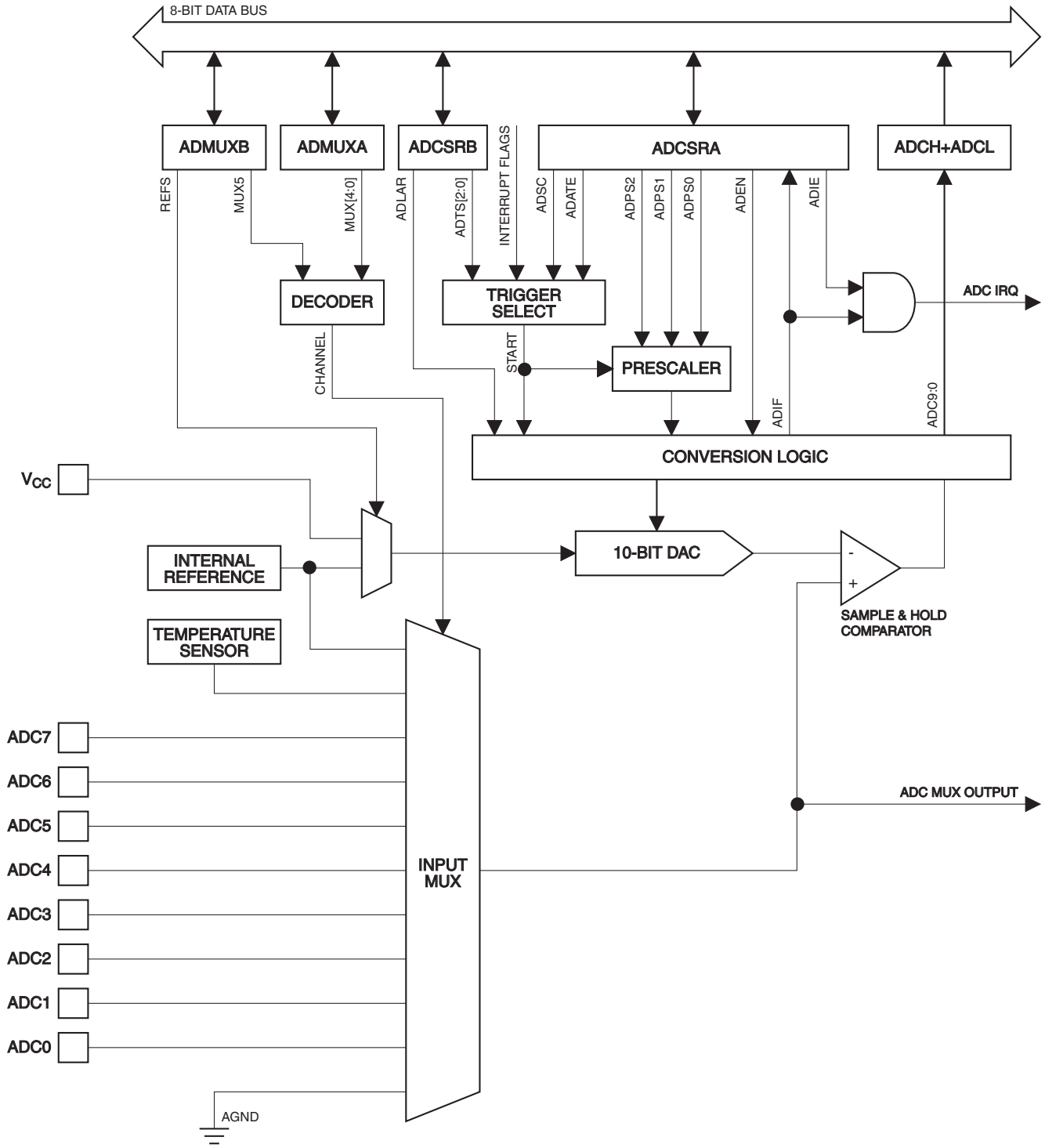
15.2 Overview

ATtiny828 features a 10-bit, successive approximation Analog-to-Digital Converter (ADC). The ADC is wired to a 32-channel analog multiplexer, which allows the ADC to measure the voltage at 28 single-ended input pins, or from four internal, single-ended voltage channels coming from the internal temperature sensor, internal voltage reference, analog ground, or supply voltage. Voltage inputs are referred to 0V (GND).

The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A block diagram of the ADC is shown in Figure 53 on page 138.

Internal reference voltage of nominally 1.1V is provided on-chip. Alternatively, V_{CC} can be used as reference voltage for single ended channels.

Figure 53. Analog to Digital Converter Block Schematic



15.3 Operation

In order to be able to use the ADC the Power Reduction bit, PRADC, in the Power Reduction Register must be disabled. This is done by clearing the PRADC bit. See "PRR – Power Reduction Register" on page 37 for more details.

The ADC is enabled by setting the ADC Enable bit, ADEN in ADCSRA. Voltage reference and input channel selections will not go into effect until ADEN is set. The ADC does not consume power when ADEN is cleared, so it is recommended to switch off the ADC before entering power saving sleep modes.

The ADC converts an analog input voltage to a 10-bit digital value using successive approximation. The minimum value represents GND and the maximum value represents the reference voltage. The ADC voltage reference is selected by writing the REFS bit. Alternatives are the V_{CC} supply pin and the internal 1.1V voltage reference.

The analog input channel is selected by writing to the MUX bits. Any of the ADC input pins can be selected as single ended inputs to the ADC.

The ADC generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADLAR bit in ADCSRB.

If the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH, only. Otherwise, ADCL must be read first, then ADCH, to ensure that the content of the data registers belongs to the same conversion. Once ADCL is read, ADC access to data registers is blocked. This means that if ADCL has been read, and a conversion completes before ADCH is read, neither register is updated and the result from the conversion is lost. When ADCH is read, ADC access to the ADCH and ADCL Registers is re-enabled.

The ADC has its own interrupt which can be triggered when a conversion completes. When ADC access to the data registers is prohibited between reading of ADCH and ADCL, the interrupt will trigger even if the result is lost.

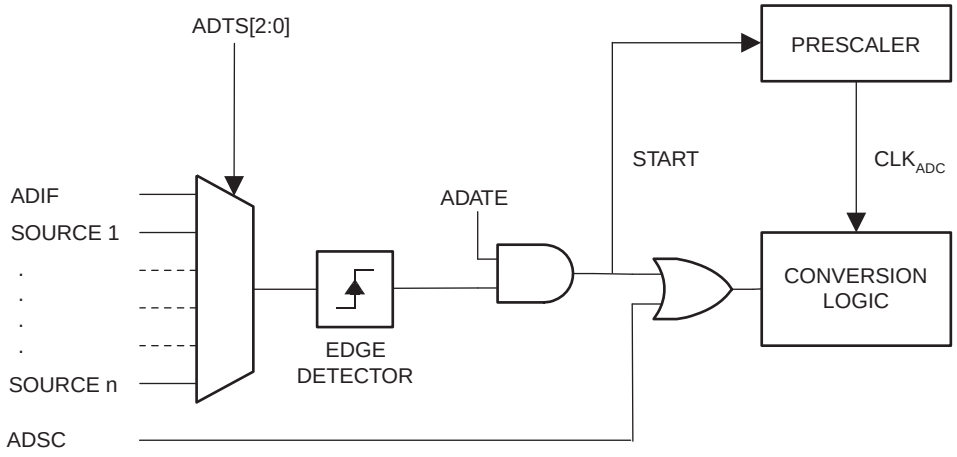
15.4 Starting a Conversion

Make sure the ADC is powered by clearing the ADC Power Reduction bit, PRADC, in the Power Reduction Register, PRR (see “PRR – Power Reduction Register” on page 37).

A single conversion is started by writing a logical one to the ADC Start Conversion bit, ADSC. This bit stays high as long as the conversion is in progress and will be cleared by hardware when the conversion is completed. If a different data channel is selected while a conversion is in progress, the ADC will finish the current conversion before performing the channel change.

Alternatively, a conversion can be triggered automatically by various sources. Auto Triggering is enabled by setting the ADC Auto Trigger Enable bit, ADATE. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in ADCSRB (see description of the ADTS bits for a list of the trigger sources). When a positive edge occurs on the selected trigger signal, the ADC prescaler is reset and a conversion is started. This provides a method of starting conversions at fixed intervals. If the trigger signal still is set when the conversion completes, a new conversion will not be started. If another positive edge occurs on the trigger signal during conversion, the edge will be ignored. Note that an Interrupt Flag will be set even if the specific interrupt is disabled or the Global Interrupt Enable bit in SREG is cleared. A conversion can thus be triggered without causing an interrupt. However, the Interrupt Flag must be cleared in order to trigger a new conversion at the next interrupt event.

Figure 54. ADC Auto Trigger Logic



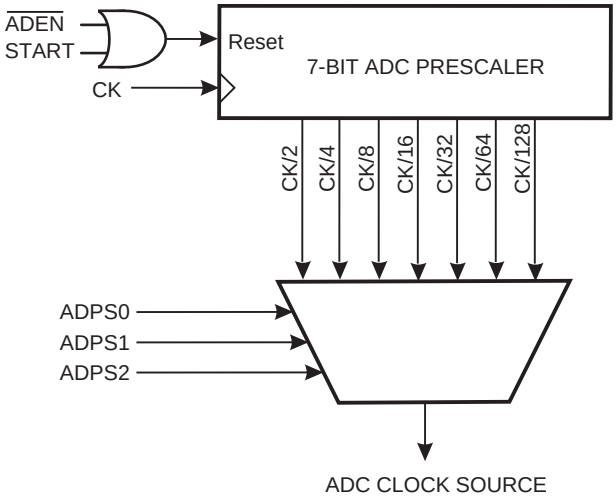
Using the ADC Interrupt Flag as a trigger source makes the ADC start a new conversion as soon as the ongoing conversion has finished. The ADC then operates in Free Running mode, constantly sampling and updating the ADC Data Register. The first conversion must be started by writing a logical one to the ADSC bit in ADCSRA. In this mode the ADC will perform successive conversions independently of whether the ADC Interrupt Flag, ADIF is cleared or not.

If Auto Triggering is enabled, single conversions can be started by writing ADSC in ADCSRA to one. ADSC can also be used to determine if a conversion is in progress. The ADSC bit will be read as one during a conversion, independently of how the conversion was started.

15.5 Prescaling and Conversion Timing

By default, the successive approximation circuitry requires an input clock frequency between 50 kHz and 200 kHz to get maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200 kHz to get a higher sample rate. It is not recommended to use a higher input clock frequency than 1 MHz.

Figure 55. ADC Prescaler

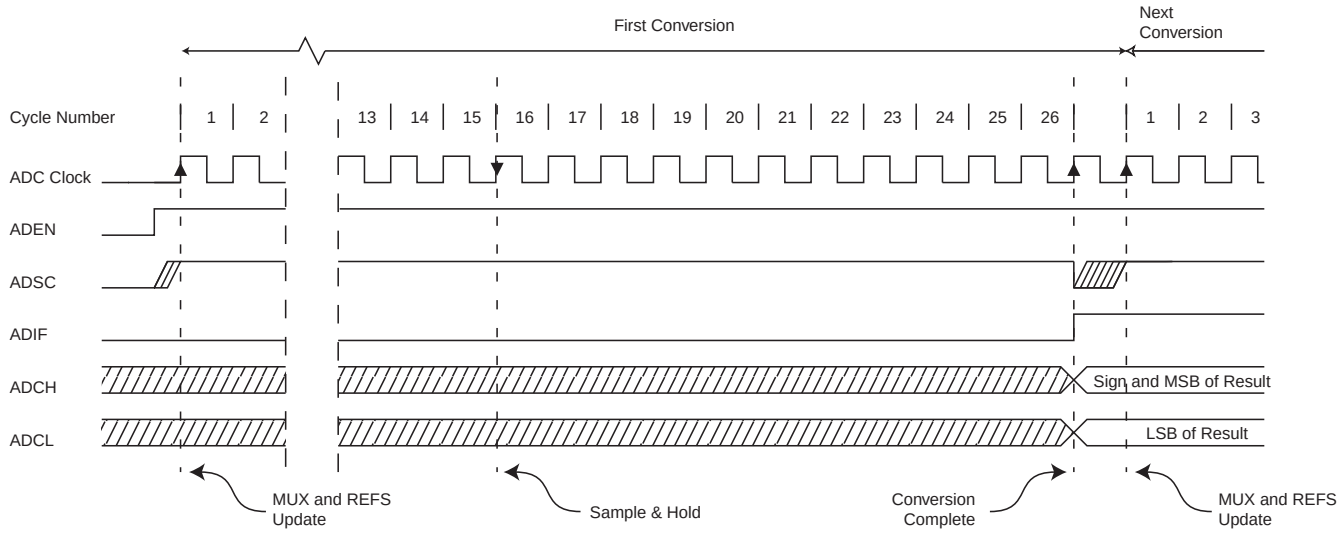


The ADC module contains a prescaler, as illustrated in Figure 55 on page 140, which generates an acceptable ADC clock frequency from any CPU frequency above 100 kHz. The prescaling is set by the ADPS bits in ADCSRA. The prescaler starts counting from the moment the ADC is switched on by setting the ADEN bit in ADCSRA. The prescaler keeps running for as long as the ADEN bit is set, and is continuously reset when ADEN is low.

When initiating a single ended conversion by setting the ADSC bit in ADCSRA, the conversion starts at the following rising edge of the ADC clock cycle.

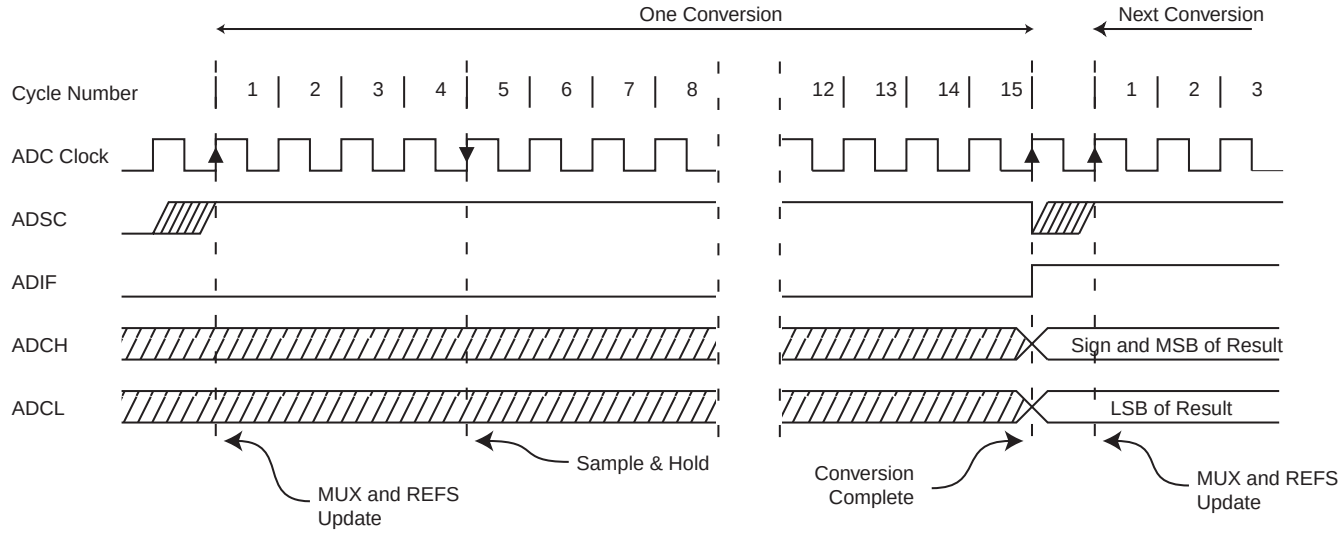
A normal conversion takes 15 ADC clock cycles, as summarised in Table 49 on page 143. The first conversion after the ADC is switched on (ADEN in ADCSRA is set) takes 26 ADC clock cycles in order to initialize the analog circuitry, as shown in Figure 56 below.

Figure 56. ADC Timing Diagram, First Conversion (Single Conversion Mode)



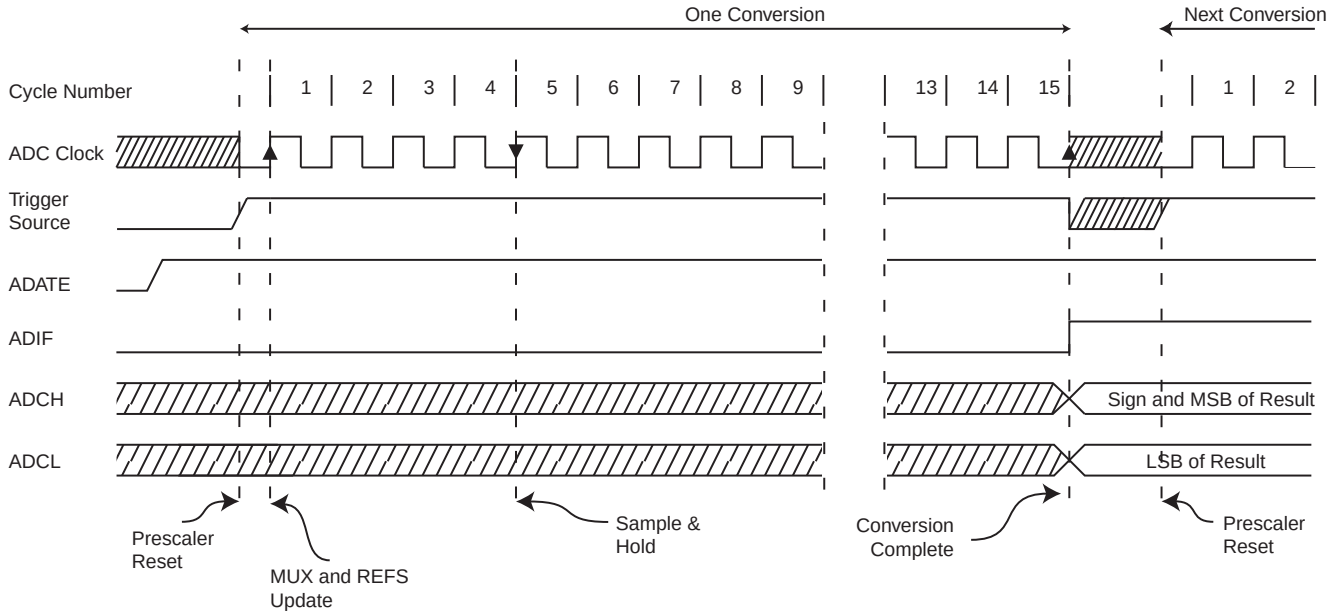
The actual sample-and-hold takes place 4 ADC clock cycles after the start of a normal conversion and 15 ADC clock cycles after the start of a first conversion. See Figure 57. When a conversion is complete, the result is written to the ADC Data Registers, and ADIF is set. In Single Conversion mode, ADSC is cleared simultaneously. The software may then set ADSC again, and a new conversion will be initiated on the first rising ADC clock edge.

Figure 57. ADC Timing Diagram, Single Conversion



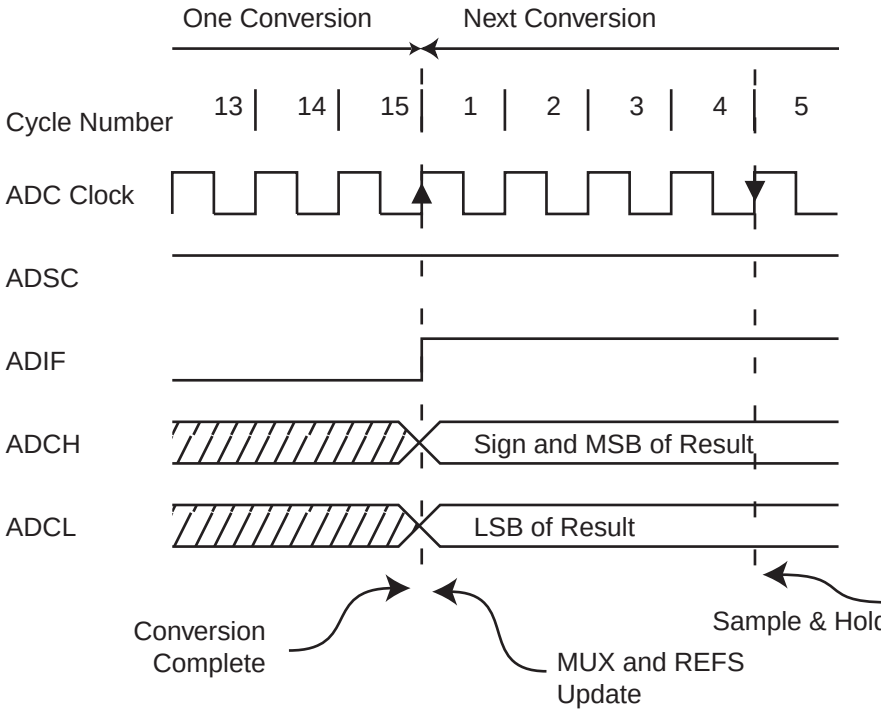
When Auto Triggering is used, the prescaler is reset when the trigger event occurs, as shown in Figure 58 below. This assures a fixed delay from the trigger event to the start of conversion. In this mode, the sample-and-hold takes place 4.5 ADC clock cycles after the rising edge on the trigger source signal. Two additional CPU clock cycles are used for synchronization logic.

Figure 58. ADC Timing Diagram, Auto Triggered Conversion



In Free Running mode, a new conversion will be started immediately after the conversion completes, while ADSC remains high. See Figure 59.

Figure 59. ADC Timing Diagram, Free Running Conversion



For a summary of conversion times, see Table 49 on page 143.

Table 49. ADC Conversion Time

Condition	Sample & Hold (Cycles from Start of Conversion)	Conversion Time (Cycles)
First conversion	15	26
Normal conversions	4	15
Auto Triggered conversions	4.5	15.5
Free Running conversion	4	15

15.6 Changing Channel or Reference Selection

The MUX and REFS bits are single buffered through a temporary register to which the CPU has random access. This ensures that the channels and reference selection only takes place at a safe point during the conversion. The channel and reference selection is continuously updated until a conversion is started. Once the conversion starts, the channel and reference selection is locked to ensure a sufficient sampling time for the ADC. Continuous updating resumes in the last ADC clock cycle before the conversion completes (ADIF in ADCSRA is set). Note that the conversion starts on the following rising ADC clock edge after ADSC is written. The user is thus advised not to write new channel or reference selection values until one ADC clock cycle after ADSC is written.

If Auto Triggering is used, the exact time of the triggering event can be indeterministic. Special care must be taken when updating the ADMUXA and ADMUXB registers, in order to control which conversion will be affected by the new settings.

If both ADATE and ADEN are written to one, an interrupt event can occur at any time. If the ADCSRA register is changed in this period, the user cannot tell if the next conversion is based on the old or the new settings. ADCSRA can be safely updated under the following conditions:

- When ADATE or ADEN is cleared.
- During conversion: at least one ADC clock cycle after the trigger event.
- After a conversion: before the Interrupt flag used as trigger source is cleared.

When updating in one of these conditions, the new settings will affect the next ADC conversion.

15.6.1 ADC Input Channels

When changing channel selections, the user should observe the following guidelines to ensure that the correct channel is selected:

- In Single Conversion mode, always select the channel before starting the conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the conversion to complete before changing the channel selection.
- In Free Running mode, always select the channel before starting the first conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the first conversion to complete, and then change the channel selection. Since the next conversion has already started automatically, the next result will reflect the previous channel selection. Subsequent conversions will reflect the new channel selection.

15.6.2 ADC Voltage Reference

The ADC reference voltage (V_{REF}) indicates the conversion range for the ADC. Single ended channels that exceed V_{REF} will result in codes close to 0x3FF. V_{REF} can be selected as either V_{CC}, or internal 1.1V reference. The internal 1.1V reference is generated from the internal bandgap reference (V_{BG}) through an internal amplifier.

The first ADC conversion result after switching reference voltage source may be inaccurate, and the user is advised to discard this result.

15.7 ADC Noise Canceler

The ADC features a noise canceler that enables conversion during sleep mode. This reduces noise induced from the CPU core and other I/O peripherals. The noise canceler can be used with ADC Noise Reduction and Idle mode. To make use of this feature, the following procedure should be used:

- Make sure that the ADC is enabled and is not busy converting. Single Conversion mode must be selected and the ADC conversion complete interrupt must be enabled.
- Enter ADC Noise Reduction mode (or Idle mode). The ADC will start a conversion once the CPU has been halted.
- If no other interrupts occur before the ADC conversion completes, the ADC interrupt will wake up the CPU and execute the ADC Conversion Complete interrupt routine. If another interrupt wakes up the CPU before the ADC conversion is complete, that interrupt will be executed, and an ADC Conversion Complete interrupt request will be generated when the ADC conversion completes. The CPU will remain in active mode until a new sleep command is executed.

Note that the ADC will not automatically be turned off when entering other sleep modes than Idle mode and ADC Noise Reduction mode. The user is advised to write zero to ADEN before entering such sleep modes to avoid excessive power consumption.

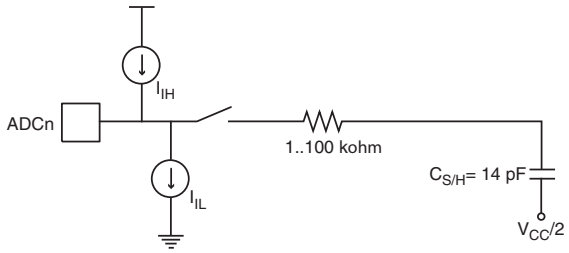
15.8 Analog Input Circuitry

The analog input circuitry for single ended channels is illustrated in Figure 60. An analog source applied to ADCn is subjected to the pin capacitance and input leakage of that pin, regardless of whether that channel is selected as input for the ADC. When the channel is selected, the source must drive the S/H capacitor through the series resistance (combined resistance in the input path).

The ADC is optimized for analog signals with an output impedance of approximately 10kΩ or less. If such a source is used, the sampling time will be negligible. If a source with higher impedance is used, the sampling time will depend on how long time the source needs to charge the S/H capacitor, which can vary widely. The user is recommended to only use low impedance sources with slowly varying signals, since this minimizes the required charge transfer to the S/H capacitor.

In order to avoid distortion from unpredictable signal convolution, signal components higher than the Nyquist frequency ($f_{ADC}/2$) should not be present. The user is advised to remove high frequency components with a low-pass filter before applying the signals as inputs to the ADC.

Figure 60. Analog Input Circuitry



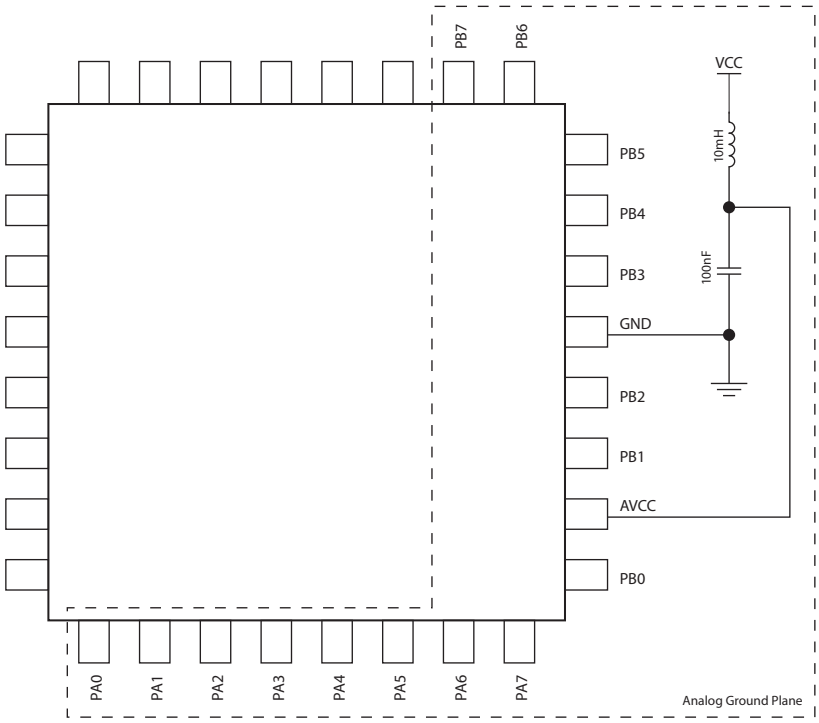
Note: The capacitor in the figure depicts the total capacitance, including the sample/hold capacitor and any stray or parasitic capacitance inside the device. The value given is worst case.

15.9 Noise Canceling Techniques

Digital circuitry inside and outside the device generates EMI which might affect the accuracy of analog measurements. When conversion accuracy is critical, the noise level can be reduced by applying the following techniques:

- Keep analog signal paths as short as possible.
- Make sure analog tracks run over the analog ground plane.
- Keep analog tracks well away from high-speed switching digital tracks.
- If any port pin is used as a digital output, it mustn't switch while a conversion is in progress.
- Place bypass capacitors as close to V_{CC} and GND pins as possible.
- The analog supply voltage pin (AV_{CC}) should be connected to the digital supply voltage pin (V_{CC}) via an LC network as shown in Figure 61.

Figure 61. ADC Power Connections



Where high ADC accuracy is required it is recommended to use ADC Noise Reduction Mode, as described in Section 15.7 on page 144. This is especially the case when system clock frequency is above 1 MHz, or when the ADC is used for reading the internal temperature sensor, as described in Section 15.12 on page 148. A good system design with properly placed, external bypass capacitors does reduce the need for using ADC Noise Reduction Mode

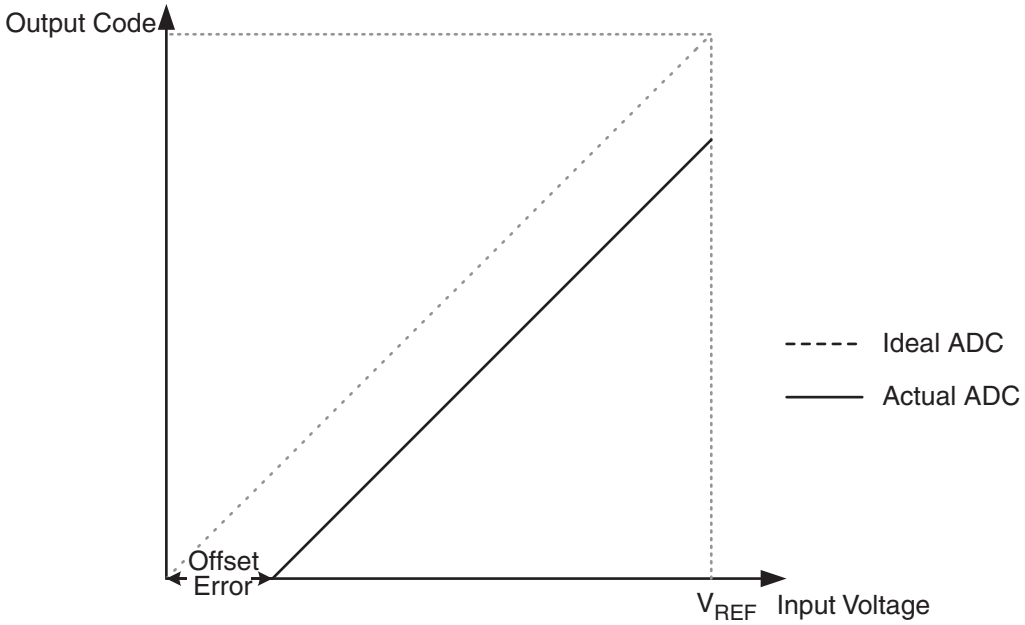
15.10 ADC Accuracy Definitions

An n-bit single-ended ADC converts a voltage linearly between GND and V_{REF} in 2^n steps (LSBs). The lowest code is read as 0, and the highest code is read as 2^n-1.

Several parameters describe the deviation from the ideal behavior, as follows:

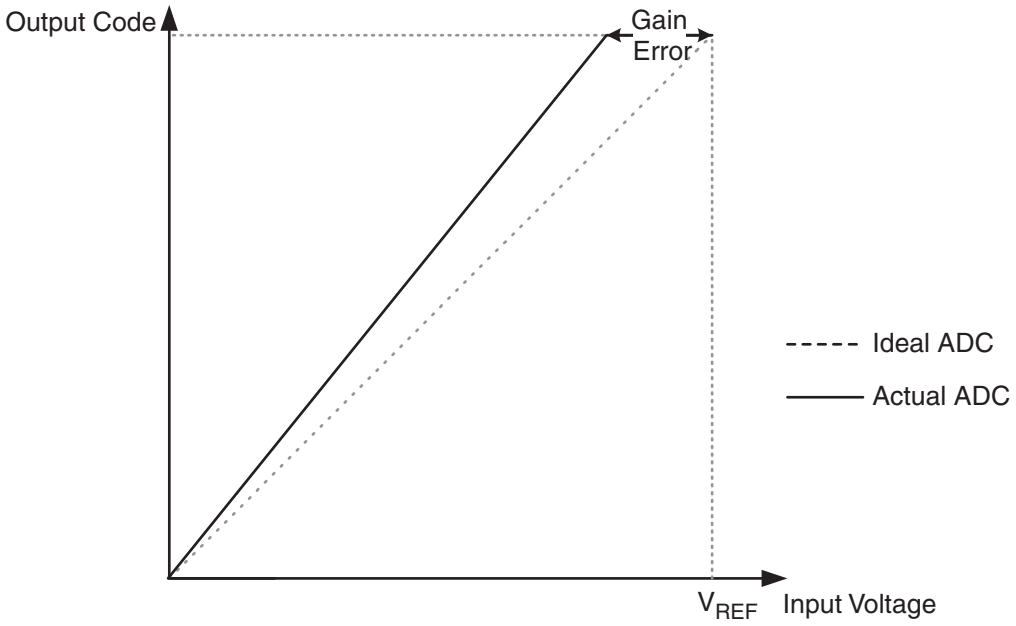
- Offset: The deviation of the first transition (0x000 to 0x001) compared to the ideal transition (at 0.5 LSB). Ideal value: 0 LSB.

Figure 62. Offset Error



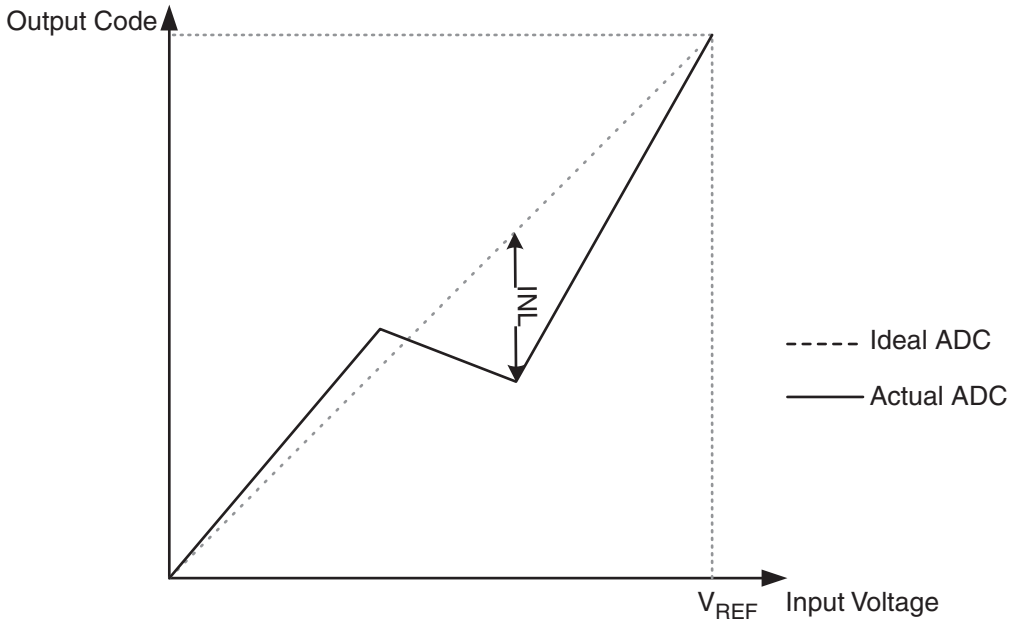
- Gain Error: After adjusting for offset, the Gain Error is found as the deviation of the last transition (0x3FE to 0x3FF) compared to the ideal transition (at 1.5 LSB below maximum). Ideal value: 0 LSB

Figure 63. Gain Error



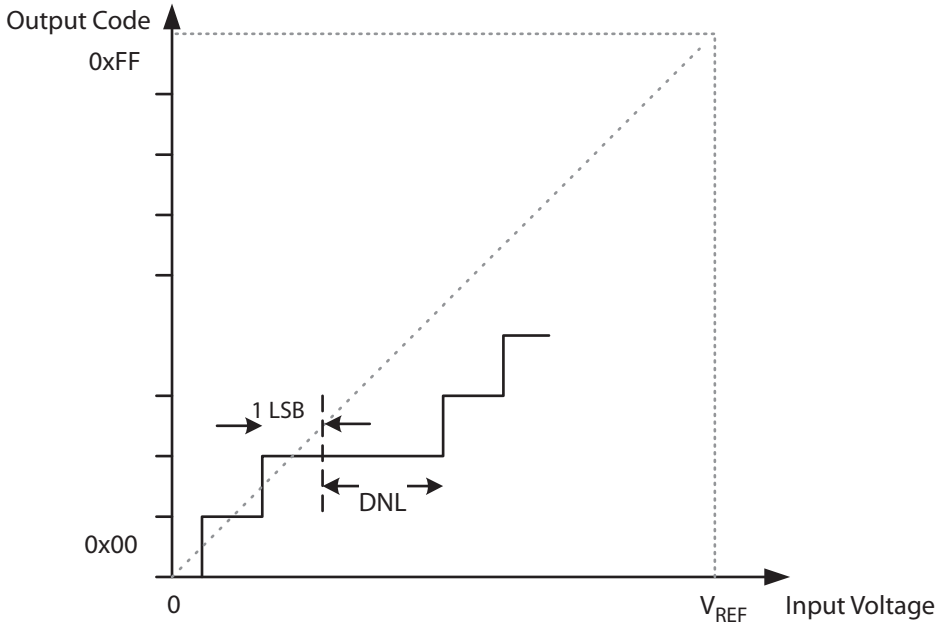
- Integral Non-linearity (INL): After adjusting for offset and gain error, the INL is the maximum deviation of an actual transition compared to an ideal transition for any code. Ideal value: 0 LSB.

Figure 64. Integral Non-linearity (INL)



- Differential Non-linearity (DNL): The maximum deviation of the actual code width (the interval between two adjacent transitions) from the ideal code width (1 LSB). Ideal value: 0 LSB.

Figure 65. Differential Non-linearity (DNL)



- Quantization Error: Due to the quantization of the input voltage into a finite number of codes, a range of input voltages (1 LSB wide) will code to the same value. Always ± 0.5 LSB.

- Absolute Accuracy: The maximum deviation of an actual (unadjusted) transition compared to an ideal transition for any code. This is the compound effect of offset, gain error, differential error, non-linearity, and quantization error. Ideal value: ± 0.5 LSB.

15.11 ADC Conversion Result

After the conversion is complete (ADIF is high), the conversion result can be found in the ADC Data Registers (ADCL, ADCH). The result is, as follows:

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

where V_{IN} is the voltage on the selected input pin and V_{REF} the selected voltage reference (see Table 52 on page 150). 0x000 represents analog ground, and 0x3FF represents the selected reference voltage minus one LSB. The result is presented in one-sided form, from 0x3FF to 0x000.

15.12 Temperature Measurement

The temperature measurement is based on an on-chip temperature sensor that is coupled to a single ended ADC channel. The temperature sensor is enabled by writing MUX bits. The internal 1.1V reference must also be selected for the ADC reference source in the temperature sensor measurement. When the temperature sensor is enabled, the ADC converter can be used in single conversion mode to measure the voltage over the temperature sensor.

The measured voltage has a linear relationship to the temperature as described in Table 50 The sensitivity is approximately 1 LSB / °C and the accuracy depends on the method of user calibration. Typically, the measurement accuracy after a single temperature calibration is ±10°C, assuming calibration at room temperature. Better accuracies are achieved by using two temperature points for calibration.

Table 50. Temperature vs. Sensor Output Voltage (Typical Case)

Temperature	-40°C	+25°C	+85°C
ADC	235 LSB	300 LSB	360 LSB

The values described in Table 50 are typical values. However, due to process variation the temperature sensor output voltage varies from one chip to another. To be capable of achieving more accurate results the temperature measurement can be calibrated in the application software. The sofware calibration can be done using the formula:

$$T = k * [(ADCH << 8) \mid ADCL] + T_{OS}$$

where ADCH and ADCL are the ADC data registers, k is the fixed slope coefficient and T_{OS} is the temperature sensor offset. Typically, k is very close to 1.0 and in single-point calibration the coefficient may be omitted. Where higher accuracy is required the slope coefficient should be evaluated based on measurements at two temperatures.

15.13 Register Description

15.13.1 ADMUXA – ADC Multiplexer Selection Register A

Bit	7	6	5	4	3	2	1	0	
(0x7C)	–	–	–	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUXA
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 4:0 – MUX[4:0] : Analog Channel and Gain Selection Bits**

These bits together with MUX5 in ADMUXB select which analog input is connected to the ADC. See Table 51.

Table 51. Single-Ended Input channel Selections

MUX[5:0]	Single Ended Input	Pin
000000	ADC0	PA0
000001	ADC1	PA1
000010	ADC2	PA2
000011	ADC3	PA3
000100	ADC4	PA4
000101	ADC5	PA5
000110	ADC6	PA6
000111	ADC7	PA7
001000	ADC8	PB0
001001	ADC9	PB1
001010	ADC10	PB2
001011	ADC11	PB3
001100	ADC12	PB4
001101	ADC13	PB5
001110	ADC14	PB6
001111	ADC15	PB7
010000	ADC16	PC0
010001	ADC17	PC1
010010	ADC18	PC2
010011	ADC19	PC3
010100	ADC20	PC4
010101	ADC21	PC5
010110	ADC22	PC6
010111	ADC23	PC7

MUX[5:0]	Single Ended Input	Pin
011000	ADC24	PD0
011001	ADC25	PD1
011010	ADC26	PD2
011011	ADC27	PD3
011100	Ground	GND
011101	Internal 1.1V reference (1)	(internal)
011110	Temperature sensor (2)	(internal)
011111	Supply voltage	VCC
100000 – 111111	Reserved	(not connected)

Notes: 1. After switching to internal voltage reference the ADC requires a settling time of 1ms before measurements are stable. Conversions starting before this may not be reliable. The ADC must be enabled during the settling time.

2. See “Temperature Measurement” on page 148.

If these bits are changed during a conversion, the change will not go into effect until the conversion is complete (ADIF in ADCSRA is set).

15.13.2 ADMUXB – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7D)	–	–	REFS	–	–	–	–	MUX5	ADMUXB
Read/Write	R	R	R/W	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7, 6, 2, 1 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 5 – REFS: Reference Selection Bit**

This bits selects the voltage reference for the ADC, as shown in Table 52.

Table 52. Voltage Reference Selections for ADC

REFS	Voltage Reference Selection
0	V_{CC} used as analog reference
1	Internal 1.1V voltage reference

If this bit is changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSR is set).

- **Bit 0 – MUX5: Analog Channel and Gain Selection Bit**

This bit together with MUX[4:0] in ADMUXA select which analog input is connected to the ADC. See Table 51 on page 149.

15.13.3 ADCL and ADCH – ADC Data Register

15.13.3.1ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

15.13.3.2ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

When an ADC conversion is complete, the result is found in these two registers.

When ADCL is read, the ADC Data Register is not updated until ADCH is read. Consequently, if the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH.

The ADLAR bit and the MUX bits affect the way the result is read from the registers. If ADLAR is set, the result is left adjusted. If ADLAR is cleared (default), the result is right adjusted.

- **ADC[9:0]: ADC Conversion Result**

These bits represent the result from the conversion, as detailed in “ADC Conversion Result” on page 148.

15.13.4 ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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 – ADEN: ADC Enable**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- **Bit 6 – ADSC: ADC Start Conversion**

In Single Conversion mode, write this bit to one to start each conversion. In Free Running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 26 ADC clock cycles instead of the normal 15. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

- **Bit 5 – ADATE: ADC Auto Trigger Enable**

When this bit is written to one, Auto Triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in ADCSRB.

- **Bit 4 – ADIF: ADC Interrupt Flag**

This bit is set when an ADC conversion completes and the data registers are updated. The ADC Conversion Complete Interrupt is executed if the ADIE bit and the I-bit in SREG are set. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag.

Beware that if doing a Read-Modify-Write on ADCSRA, a pending interrupt can be disabled.

- **Bit 3 – ADIE: ADC Interrupt Enable**

When this bit is written to one and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

- **Bits 2:0 – ADPS[2:0]: ADC Prescaler Select Bits**

These bits determine the division factor between the system clock frequency and the input clock to the ADC.

Table 53. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

15.13.5 ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	–	–	–	ADLAR	ADTS2	ADTS1	ADTS0	ADCSRB
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 5 – Res: Reserved Bit**

Thisis a reserved bit. For compatibility with future devices always write this bit to zero.

- **Bit 3 – ADLAR: ADC Left Adjust Result**

The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC Data Register immediately, regardless of any ongoing conversions. For a comple the description of this bit, see “ADCL and ADCH – ADC Data Register” on page 151.

- **Bits 2:0 – ADTS[2:0] : ADC Auto Trigger Source**

If ADATE in ADCSRA is written to one, the value of these bits selects which source will trigger an ADC conversion. If ADATE is cleared, the ADTS[2:0] settings will have no effect. A conversion will be triggered by the rising edge of the selected Interrupt Flag. Note that switching from a trigger source that is cleared to a trigger source that is set, will generate a positive edge on the trigger signal. If ADEN in ADCSRA is set, this will start a conversion. Switching to Free Running mode (ADTS[2:0]=0) will not cause a trigger event, even if the ADC Interrupt Flag is set**.**

Table 54. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match A
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

15.13.6 DIDR0 – Digital Input Disable Register 0

Bit	7	6	5	4	3	2	1	0	
(0x7E)	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	DIDR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – ADC7D:ADC0D : ADC[7:0] Digital Input Disable**

When an analog signal is applied to ADCn and the digital input of the pin is not needed, the ADCnD bit should be set to reduce power consumption. Setting ADCnD disables the digital input buffer on the corresponding pin (ADCn). When ADCnD is set the corresponding bit in the PINxn register will always read zero.

15.13.7 DIDR1 – Digital Input Disable Register 1

Bit	7	6	5	4	3	2	1	0	
(0x7F)	ADC15D	ADC14D	ADC13D	ADC12D	ADC11D	ADC10D	ADC9D	ADC8D	DIDR1
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – ADC15D:ADC8D : ADC[15:8] Digital Input Disable**

When an analog signal is applied to ADCn and the digital input of the pin is not needed, the ADCnD bit should be set to reduce power consumption. Setting ADCnD disables the digital input buffer on the corresponding pin (ADCn). When ADCnD is set the corresponding bit in the PINxn register will always read zero.

15.13.8 DIDR2 – Digital Input Disable Register 2

Bit	7	6	5	4	3	2	1	0	
(0xDE)	ADC23D	ADC22D	ADC21D	ADC20D	ADC19D	ADC18D	ADC17D	ADC16D	DIDR2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – ADC23D:ADC16D : ADC[23:16] Digital Input Disable**

When an analog signal is applied to ADCn and the digital input of the pin is not needed, the ADCnD bit should be set to reduce power consumption. Setting ADCnD disables the digital input buffer on the corresponding pin (ADCn). When ADCnD is set the corresponding bit in the PINxn register will always read zero.

15.13.9 DIDR3 – Digital Input Disable Register 3

Bit	7	6	5	4	3	2	1	0	
(0xDF)	–	–	–	–	ADC27D	ADC26D	ADC25D	ADC24D	DIDR3
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 27:24 – ADC27D:ADC24D : ADC[27:24] Digital Input Disable**

When an analog signal is applied to ADCn and the digital input of the pin is not needed, the ADCnD bit should be set to reduce power consumption. Setting ADCnD disables the digital input buffer on the corresponding pin (ADCn). When ADCnD is set the corresponding bit in the PINxn register will always read zero.

16. SPI – Serial Peripheral Interface

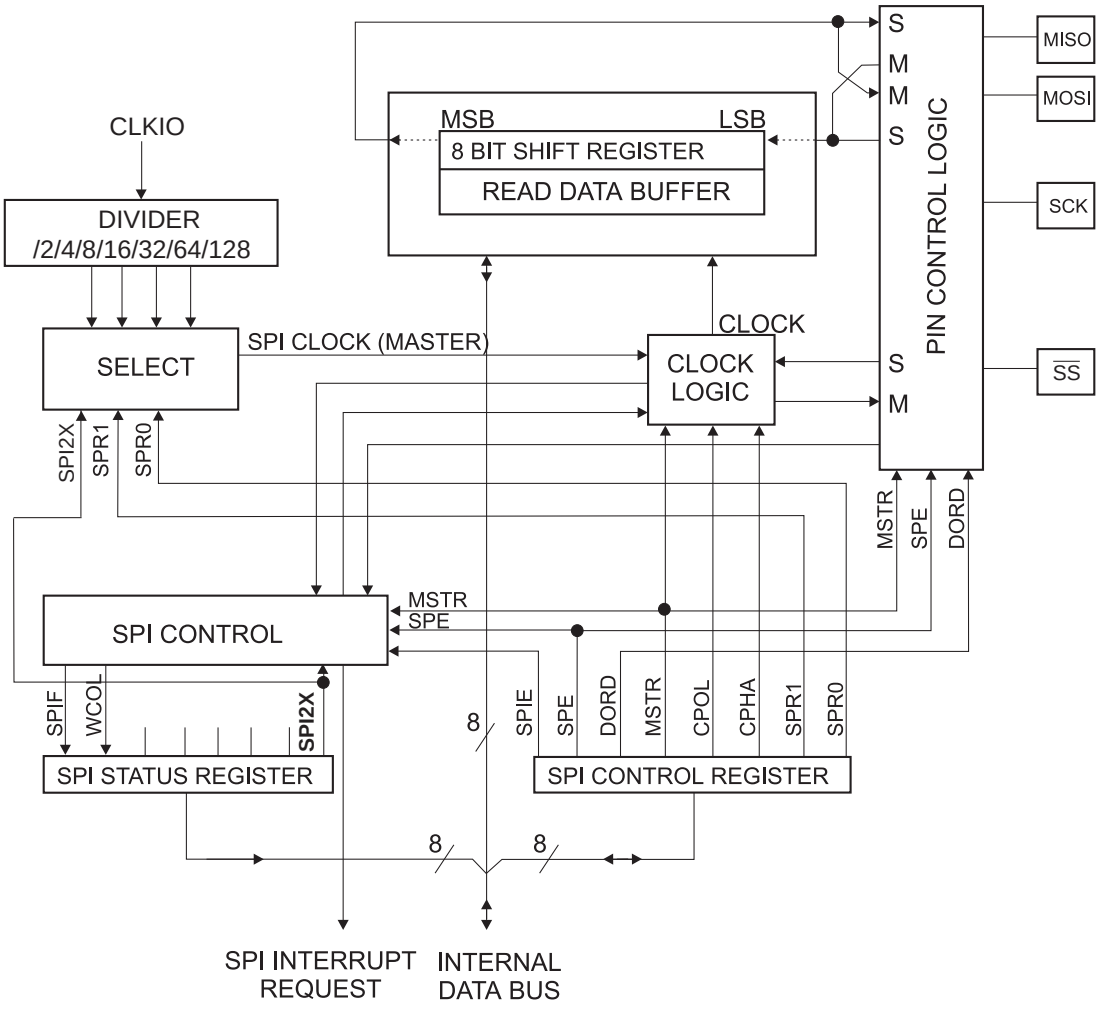
16.1 Features

- Full-duplex, Three-wire Synchronous Data Transfer
- Master or Slave Operation
- LSB First or MSB First Data Transfer
- Seven Programmable Bit Rates
- End of Transmission Interrupt Flag
- Write Collision Flag Protection
- Wake-up from Idle Mode
- Double Speed (CK/2) Master SPI Mode

16.2 Overview

The Serial Peripheral Interface (SPI) allows high-speed synchronous data transfer between ATtiny828 and peripheral devices, or between several AVR devices. The SPI module is illustrated in Figure 66.

Figure 66. SPI Block Diagram



Note: For SPI pin placement, see Figure 1 on page 2, and Table 55 on page 157.

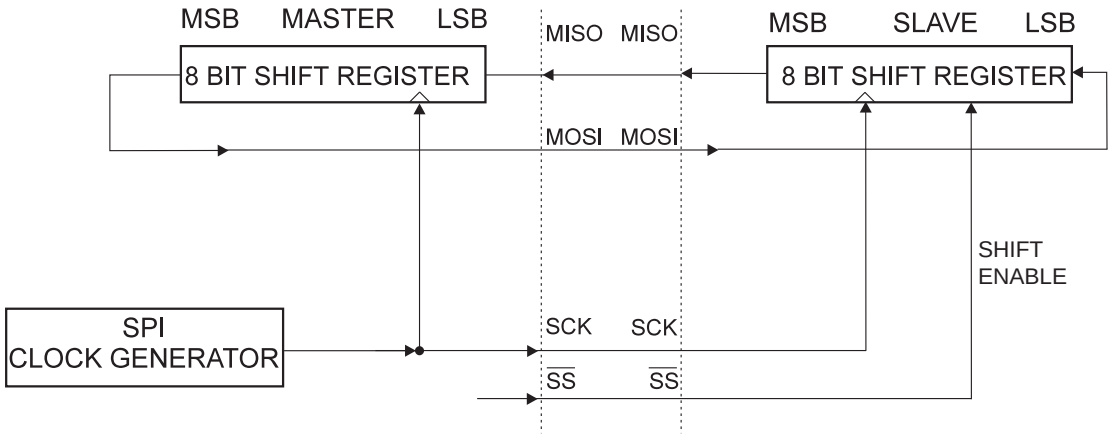
To enable the SPI module, the PRSPI bit in the Power Reduction Register must be written to zero. See “PRR – Power Reduction Register” on page 37.

The interconnection between Master and Slave CPUs with SPI is shown in Figure 67 on page 156. The system consists of two shift Registers, and a Master clock generator. The SPI Master initiates the communication cycle when pulling low the Slave Select $\overline{SS}$ pin of the desired Slave. Master and Slave prepare the data to be sent in their respective shift Registers, and the Master generates the required clock pulses on the SCK line to interchange data. Data is always shifted from Master to Slave on the Master Out – Slave In, MOSI, line, and from Slave to Master on the Master In – Slave Out, MISO, line. After each data packet, the Master will synchronize the Slave by pulling high the Slave Select, $\overline{SS}$, line.

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

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

Figure 67. SPI Master-Slave Interconnection



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

In SPI Slave mode, the control logic will sample the incoming signal of the SCK pin. To ensure correct sampling of the clock signal, the minimum low and high periods should be:

- Low periods: Longer than 2 CPU clock cycles.
- High periods: Longer than 2 CPU clock cycles.

When the SPI is enabled, the data direction of the MOSI, MISO, SCK, and $\overline{SS}$ pins is overridden according to Table 55 on page 157. For more details on automatic port overrides, refer to “Alternative Port Functions” on page 63.

Table 55. SPI Pin Overrides

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

Note: See “Alternative Port Functions” on page 63 for a detailed description of how to define the direction of the user defined SPI pins.

The following code examples show how to initialize the SPI as a Master and how to perform a simple transmission. DDR_SPI in the examples must be replaced by the actual Data Direction Register controlling the SPI pins. DD_MOSI, DD_MISO and DD_SCK must be replaced by the actual data direction bits for these pins. E.g. if MOSI is placed on pin PB5, replace DD_MOSI with DDB5 and DDR_SPI with DDRB.

Assembly Code Example

```
SPI_MasterInit:
      ; Set MOSI and SCK output, all others input
      ldi               r17,(1<<DD_MOSI)|(1<<DD_SCK)
      out               DDR_SPI,r17
      ; Enable SPI, Master, set clock rate fck/16
      ldi               r17,(1<<SPE)|(1<<MSTR)|(1<<SPR0)
      out               SPCR,r17
      ret

SPI_MasterTransmit:
      ; Start transmission of data (r16)
      out               SPDR,r16

Wait_Transmit:
      ; Wait for transmission complete
      in                r16, SPSR
      sbrs        r16, SPIF
      rjmp              Wait_Transmit
      ret
```

C Code Example

```
void SPI_MasterInit(void)
{
      /* Set MOSI and SCK output, all others input */
      DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);
      /* Enable SPI, Master, set clock rate fck/16 */
      SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}

void SPI_MasterTransmit(char cData)
{
      /* Start transmission */
      SPDR = cData;
      /* Wait for transmission complete */
      while(!(SPSR & (1<<SPIF)))
            ;
}
```

Note: See "Code Examples" on page 7.

The following code examples show how to initialize the SPI as a Slave and how to perform a simple reception.

Assembly Code Example

```
SPI_SlaveInit:
      ; Set MISO output, all others input
      ldi               r17,(1<<DD_MISO)
      out               DDR_SPI,r17
      ; Enable SPI
      ldi               r17,(1<<SPE)
      out               SPCR,r17
      ret

SPI_SlaveReceive:
      ; Wait for reception complete
      in                r16, SPSR
      sbrs              r16, SPIF
      rjmp              SPI_SlaveReceive
      ; Read received data and return
      in                r16,SPDR
      ret
```

C Code Example

```
void SPI_SlaveInit(void)
{
      /* Set MISO output, all others input */
      DDR_SPI = (1<<DD_MISO);
      /* Enable SPI */
      SPCR = (1<<SPE);
}

char SPI_SlaveReceive(void)
{
      /* Wait for reception complete */
      while(!(SPSR & (1<<SPIF)))
            ;
      /* Return Data Register */
      return SPDR;
}
```

Note: See "Code Examples" on page 7.

16.3 $\overline{SS}$ Pin Functionality

16.3.1 Slave Mode

When the SPI is configured as a Slave, the Slave Select ($\overline{SS}$) pin is always input. When $\overline{SS}$ is held low, the SPI is activated, and MISO becomes an output if configured so by the user. All other pins are inputs. When $\overline{SS}$ is driven high, all pins are inputs, and the SPI is passive, which means that it will not receive incoming data. Note that the SPI logic will be reset once the $\overline{SS}$ pin is driven high.

The $\overline{SS}$ pin is useful for packet/byte synchronization to keep the slave bit counter synchronous with the master clock generator. When the $\overline{SS}$ pin is driven high, the SPI slave will immediately reset the send and receive logic, and drop any partially received data in the Shift Register.

16.3.2 Master Mode

When the SPI is configured as a Master (MSTR in SPCR is set), the user can determine the direction of the $\overline{SS}$ pin.

If $\overline{SS}$ is configured as an output, the pin is a general output pin which does not affect the SPI system. Typically, the pin will be driving the $\overline{SS}$ pin of the SPI Slave.

If $\overline{SS}$ is configured as an input, it must be held high to ensure Master SPI operation. If the $\overline{SS}$ pin is driven low by peripheral circuitry when the SPI is configured as a Master with the $\overline{SS}$ pin defined as an input, the SPI system interprets this as another master selecting the SPI as a slave and starting to send data to it. To avoid bus contention, the SPI system takes the following actions:

1. The MSTR bit in SPCR is cleared and the SPI system becomes a Slave. As a result of the SPI becoming a Slave, the MOSI and SCK pins become inputs.
2. The SPIF Flag in SPSR is set, and if the SPI interrupt is enabled, and the I-bit in SREG is set, the interrupt routine will be executed.

Thus, when interrupt-driven SPI transmission is used in Master mode, and there exists a possibility that $\overline{SS}$ is driven low, the interrupt should always check that the MSTR bit is still set. If the MSTR bit has been cleared by a slave select, it must be set by the user to re-enable SPI Master mode.

16.4 Data Modes

There are four combinations of SCK phase and polarity with respect to serial data, which are determined by control bits CPHA and CPOL. The SPI data transfer formats are shown in Figure 68 on page 160 and Figure 69 on page 160.

Figure 68. SPI Transfer Format with CPHA = 0

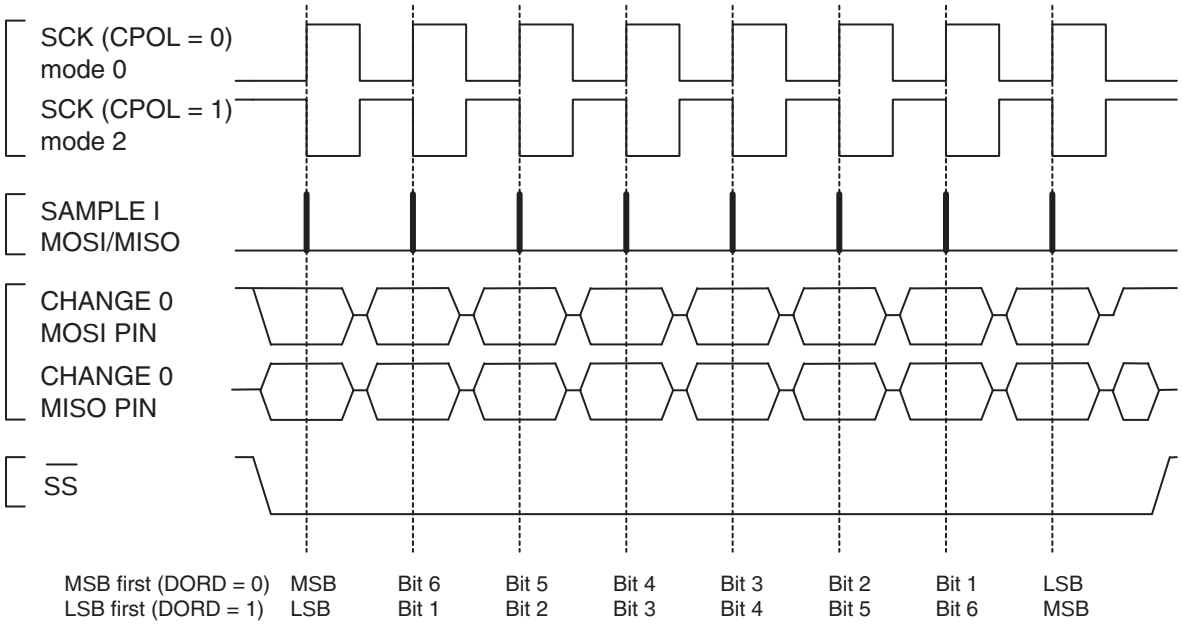
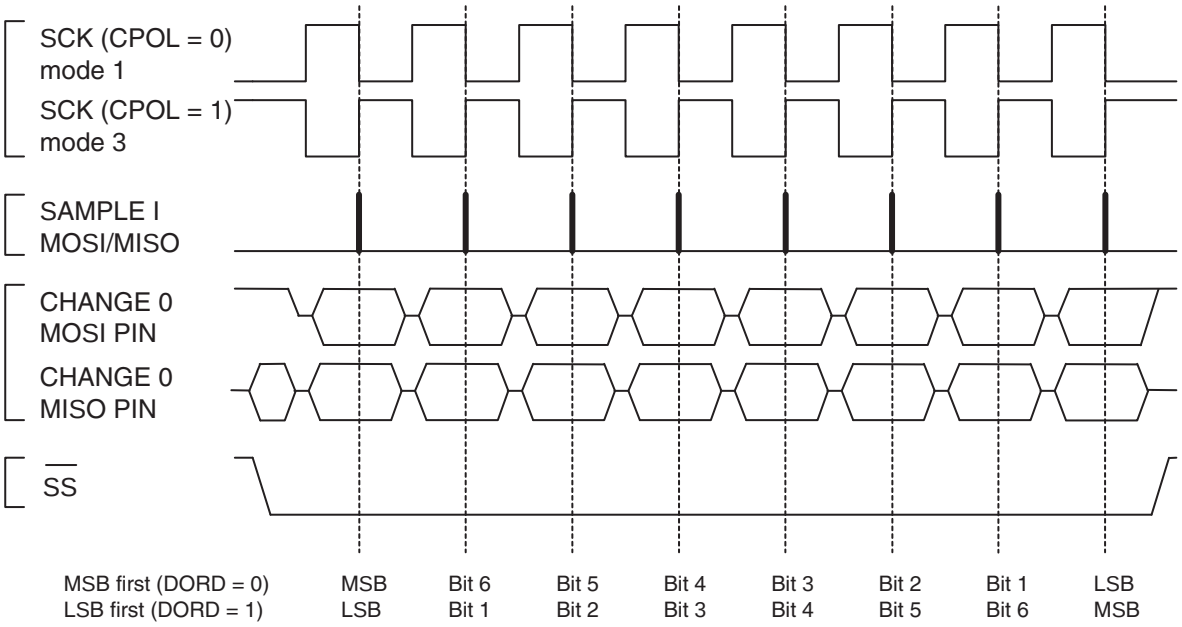


Figure 69. SPI Transfer Format with CPHA = 1



Data bits are shifted out and latched in on opposite edges of the SCK signal, ensuring sufficient time for data signals to stabilize. This is shown in Table 56, which is a summary of Table 57 on page 161 and Table 58 on page 162.

Table 56. SPI Modes

SPI Mode	Conditions	Leading Edge	Trailing eDge
0	CPOL=0, CPHA=0	Sample (Rising)	Setup (Falling)
1	CPOL=0, CPHA=1	Setup (Rising)	Sample (Falling)
2	CPOL=1, CPHA=0	Sample (Falling)	Setup (Rising)
3	CPOL=1, CPHA=1	Setup (Falling)	Sample (Rising)

16.5 Register Description

16.5.1 SPCR – SPI Control Register

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
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 – SPIE: SPI Interrupt Enable**

When this bit is set, the SPI interrupt is enabled. Provided the Global Interrupt Enable bit in SREG is set, the SPI interrupt service routine will be executed when the SPIF bit in SPSR is set.

- **Bit 6 – SPE: SPI Enable**

When this bit is set, the SPI is enabled. This bit must be set to enable any SPI operations.

- **Bit 5 – DORD: Data Order**

When this bit is set, the LSB of the data word is transmitted first.

When this bit is cleared, the MSB of the data word is transmitted first.

- **Bit 4 – MSTR: Master/Slave Select**

This bit selects Master SPI mode when written to one, and Slave SPI mode when written logic zero. If $\overline{SS}$ is configured as an input and is driven low while MSTR is set, MSTR will be cleared, and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI Master mode.

- **Bit 3 – CPOL: Clock Polarity**

When this bit is set, SCK is high when idle. When this bit is cleared, SCK is low when idle. Refer to Figure 68 and Figure 69 for an example. The CPOL functionality is summarized below:

Table 57. CPOL Functionality

CPOL	Leading Edge	Trailing Edge
0	Rising	Falling
1	Falling	Rising

- **Bit 2 – CPHA: Clock Phase**

The settings of the Clock Phase bit (CPHA) determine if data is sampled on the leading (first) or trailing (last) edge of SCK. Refer to Figure 68 and Figure 69 for an example. The CPOL functionality is summarized below:

Table 58. CPHA Functionality

CPHA	Leading Edge	Trailing Edge
0	Sample	Setup
1	Setup	Sample

- **Bits 1:0 – SPR[1:0]: SPI Clock Rate Select 1 and 0**

These two bits control the SCK rate of the device configured as a Master. SPR1 and SPR0 have no effect on the Slave. The relationship between SCK and the I/O clock frequency $f_{clk_I/O}$ is shown in the following table:

Table 59. Relationship Between SCK and the I/O Clock Frequency

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{clk_I/O}/4$
0	0	1	$f_{clk_I/O}/16$
0	1	0	$f_{clk_I/O}/64$
0	1	1	$f_{clk_I/O}/128$
1	0	0	$f_{clk_I/O}/2$
1	0	1	$f_{clk_I/O}/8$
1	1	0	$f_{clk_I/O}/32$
1	1	1	$f_{clk_I/O}/64$

16.5.2 SPSR – SPI Status Register

Bit	7	6	5	4	3	2	1	0	
0x2D (0x4D)	**SPIF**	**WCOL**	–	–	–	–	–	**SPI2X**	**SPSR**
Read/Write	R/W	R/W	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIF: SPI Interrupt Flag**

This bit is set when a serial transfer is complete. An interrupt is generated if SPIE in SPCR is set and global interrupts are enabled. If $\overline{SS}$ is an input and is driven low when the SPI is in Master mode, this will also set the SPIF Flag. SPIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, the SPIF bit is cleared by first reading the SPI Status Register with SPIF set, then accessing the SPI Data Register (SPDR).

- **Bit 6 – WCOL: Write COLlision Flag**

This bit is set if the SPI Data Register (SPDR) is written during a data transfer. The WCOL bit (and the SPIF bit) are cleared by first reading the SPI Status Register with WCOL set, and then accessing the SPI Data Register.

- **Bits 5:1 – Res: Reserved Bits**

These bits are reserved and will always read as zero.

- **Bit 0 – SPI2X: Double SPI Speed Bit**

When this bit is set the SPI speed (SCK Frequency) will be doubled when the SPI is in Master mode (see Table 59 on page 162). This means that the minimum SCK period will be two I/O clock periods. When the SPI is configured as Slave, the SPI is only guaranteed to work at $f_{clk_I/O}/4$ or lower.

16.5.3 SPDR – SPI Data Register

Bit	7	6	5	4	3	2	1	0	
0x2E (0x4E)	**MSB**							**LSB**	**SPDR**
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	Undefined

The SPI Data Register is a read/write register used for data transfer between the Register File and the SPI Shift Register. Writing to the register initiates data transmission. Reading the register causes the Shift Register Receive buffer to be read.

17. USART

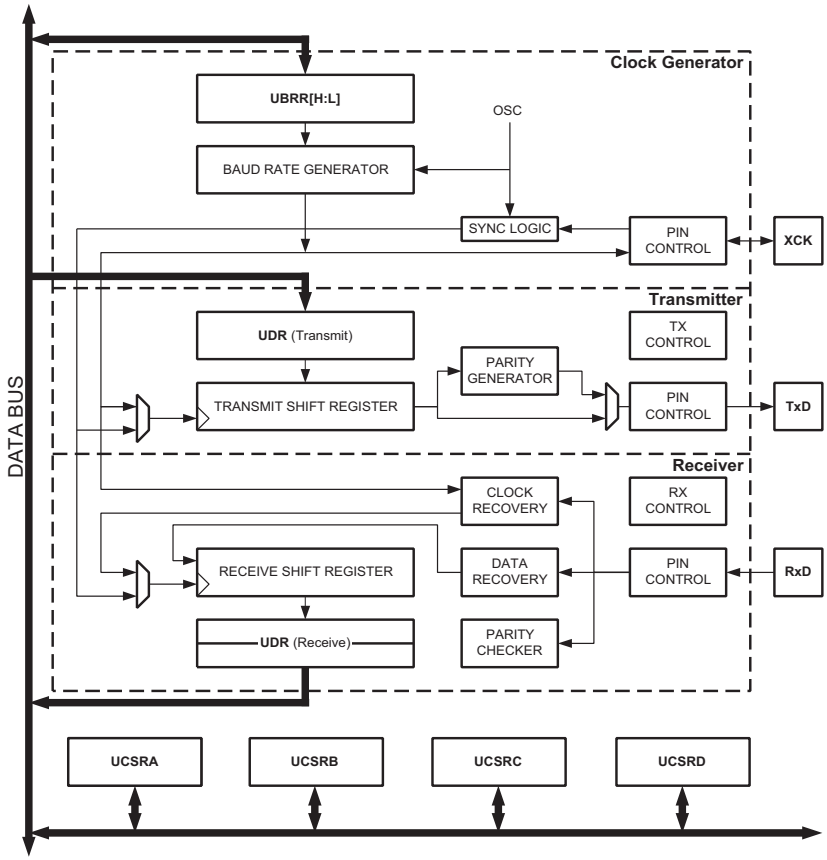
17.1 Features

- Full Duplex Operation (Independent Serial Receive and Transmit Registers)
- Asynchronous or Synchronous Operation
- Master or Slave Clocked Synchronous Operation
- High Resolution Baud Rate Generator
- Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits
- Odd or Even Parity Generation and Parity Check Supported by Hardware
- Data OverRun Detection
- Framing Error Detection
- Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter
- Three Separate Interrupts on TX Complete, TX Data Register Empty and RX Complete
- Multi-processor Communication Mode
- Double Speed Asynchronous Communication Mode
- Start Frame Detection

17.2 Overview

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device. A simplified block diagram of the USART transmitter is shown in Figure 70. CPU accessible I/O registers and I/O pins are shown in bold.

Figure 70. USART Block Diagram



For USART pin placement, see Figure 1 on page 2 and “Alternative Port Functions” on page 63.

The dashed boxes in the block diagram of Figure 70 illustrate the three main parts of the USART, as follows(listed from the top):

- Clock generator
- Transmitter
- Receiver

The clock generation logic consists of synchronization logic (for external clock input in synchronous slave operation), and the baud rate generator. The transfer clock pin (XCK) is only used in synchronous transfer mode.

The transmitter consists of a single write buffer, a serial shift register, a parity generator and control logic for handling different serial frame formats. The write buffer allows a continuous transfer of data without delay between frames.

The receiver is the most complex part of the USART module due to its clock and data recovery units. The recovery units are used for asynchronous data reception. In addition to the recovery units, the receiver includes a parity checker, control logic, a ahift register and a two level receive buffer (UDR). The receiver supports the same frame formats as the transmitter, and can detect the following errors:

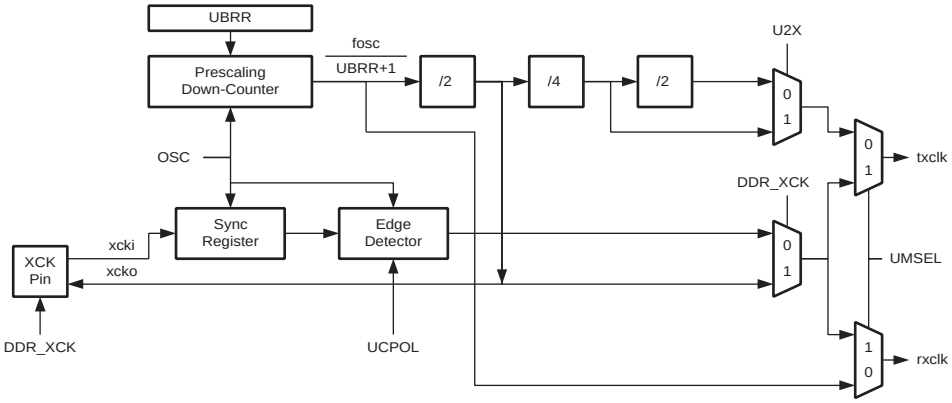
- Frame Error
- Data Overrun Error
- Parity Errors.

In order for the USART to be operative the USART power reducion bit must be disabled. See “PRR – Power Reduction Register” on page 37.

17.3 Clock Generation

The clock generation logic creates the base clock for the transmitter and receiver. A block diagram of the clock generation logic is shown in Figure 71.

Figure 71. Clock Generation Logic, Block Diagram



Signal description for Figure 71:

txclk Transmitter clock (Internal Signal)
rxclk Receiver base clock (Internal Signal)
xcki Input from XCK pin (internal Signal). Used for synchronous slave operation
xcko Clock output to XCK pin (Internal Signal). Used for synchronous master operation
f_{osc} XTAL pin frequency (System Clock)

The USART supports four modes of clock operation, as follows:

- Normal asynchronous mode
- Double speed asynchronous mode
- Master synchronous mode
- Slave synchronous mode

The UMSEL bit (see “UCSRC – USART Control and Status Register C” on page 186) selects between asynchronous and synchronous operation. In asynchronous mode, the speed is controlled by the U2X bit (see “UCSRA – USART Control and Status Register A” on page 184).

In synchronous mode (UMSEL = 1), the direction bit of the XCK pin (DDR_XCK) in the Data Direction Register where the XCK pin is located (DDRx) controls whether the clock source is internal (master mode), or external (slave mode). The XCK pin is active in synchronous mode, only.

17.3.1 Internal Clock Generation – The Baud Rate Generator

Internal clock generation is used in asynchronous and synchronous master modes of operation. The description in this section refers to Figure 71 on page 165.

The USART Baud Rate Register (UBRR) and the down-counter connected to it function as a programmable prescaler, or baud rate generator. The down-counter, running at system clock (f_{osc}) is loaded with the UBRR value each time the counter has counted down to zero, or when UBRR0L is written.

A clock is generated each time the counter reaches zero. This is the baud rate generator clock output and has a frequency of f_{osc}/(UBRR+1). Depending on the mode of operation the transmitter divides the baud rate generator clock output by 2, 8 or 16. The baud rate generator output is used directly by the receiver’s clock and data recovery units. However, the recovery units use a state machine that uses 2, 8 or 16 states, depending on mode set by UMSEL, U2X and DDR_XCK bits.

Table 60 contains equations for calculating the baud rate (in bits per second) and for calculating the UBRR value for each mode of operation using an internally generated clock source.

Table 60. Equations for Calculating Baud Rate Register Setting

Operating Mode	Baud Rate[1]	UBRR Value
Asynchronous Normal mode (U2Xn = 0)	$BAUD = \frac{f_{OSC}}{16 \times (\boldsymbol{UBRR} + 1)}$	$\boldsymbol{UBRR} = \frac{f_{OSC}}{16 \times BAUD} - 1$
Asynchronous Double Speed mode (U2Xn = 1)	$BAUD = \frac{f_{OSC}}{8 \times (\boldsymbol{UBRR} + 1)}$	$\boldsymbol{UBRR} = \frac{f_{OSC}}{8 \times BAUD} - 1$
Synchronous Master mode	$BAUD = \frac{f_{OSC}}{2 \times (\boldsymbol{UBRR} + 1)}$	$\boldsymbol{UBRR} = \frac{f_{OSC}}{2 \times BAUD} - 1$

Note: 1. Baud rate is defined as the transfer rate in bits per second (bps)

Signal description for Table 60:

BAUD	Baud rate (in bits per second, bps)
f_{osc}	System Oscillator clock frequency
UBRR	Contents of the UBRRH and UBRRL Registers, (0-4095)

Some examples of UBRR values for selected system clock frequencies are shown in Table 63 on page 181.

17.3.2 Double Speed Operation (U2X)

The transfer rate can be doubled by setting the U2X bit (see “UCSRA – USART MSPIM Control and Status Register A” on page 196). Setting this bit only has effect in asynchronous mode of operation. In synchronous mode of operation this bit should be cleared.

Setting this bit will reduce the divisor of the baud rate divider from 16 to 8, effectively doubling the transfer rate for asynchronous communication. Note, however, that in this case the receiver will use half the number of samples, only. In double speed mode, the number of data and clock recovery sampels are reduced from 16 to 8, and therefore a more accurate baud rate setting and system clock are required.

There are no downsides for the transmitter.

17.3.3 External Clock

External clocking is used in synchronous slave modes of operation. To minimize the chance of meta-stability, the external clock input from the XCK pin is sampled by a synchronization register. The output from the synchronization register then passes through an edge detector before it is used by the transmitter and receiver. This process introduces a delay of two CPU clocks, and therefore the maximum external clock frequency is limited by the following equation:

$$f_{XCK} < \frac{f_{OSC}}{4}$$

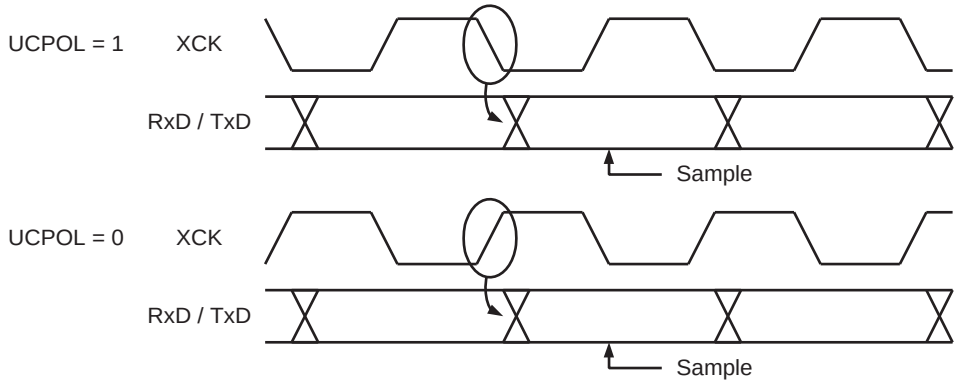
Note that f_{osc} depends on the stability of the system clock source. It is therefore recommended to add some margin to avoid possible data loss due to frequency variations.

17.3.4 Synchronous Clock Operation

In synchronous mode (UMSEL = 1), the XCK is used as either clock input (slave mode) or clock output (master mode). The dependency between clock edges and data sampling or data change is the same. The basic principle is that data input (on RxD) is sampled on the opposite XCK clock edge when data output (TxDn) is changed.

Which XCK clock edge is used for data sampling and which is used for data change can be changed with the UCPOL bit (see “UCSRC – USART MSPIM Control and Status Register C” on page 197).

Figure 72. Synchronous Mode XCK Timing.



As shown in Figure 72, when UCPOL is set, the data is changed at falling XCK edge and sampled at rising XCK edge. When UCPOL is cleared, the data is changed at rising XCK edge and sampled at falling XCK edge.

17.4 Frame Formats

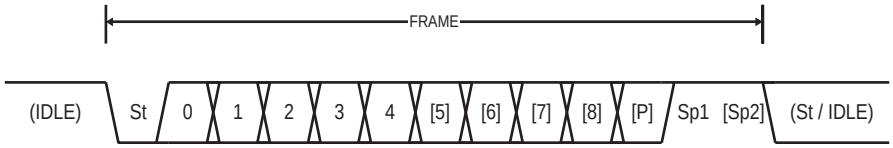
A serial frame is defined to be one character of data bits with synchronization bits (start and stop bits), and optionally a parity bit for error checking. The USART accepts all 30 combinations of the following as valid frame formats:

- Start bit: 1
- Data bits: 5, 6, 7, 8, or 9
- Parity bit: no, even, or odd parity
- Stop bits: 1, or 2

A frame begins with the start bit followed by the least significant data bit. Then follows the other data bits, the last one being the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame has been transmitted it can be directly followed by a new frame, or the communication line can be set to an idle (high) state.

Figure 73 illustrates the possible combinations of the frame formats. Bits inside brackets are optional.

Figure 73. Frame Formats



Signal description for Figure 73:

St	Start bit (always low)
(n)	Data bits (0 to 4/5/6/7/8)
P	Parity bit, if enabled (odd or even)
Sp	Stop bit (always high)
IDLE	No transfers on the communication line (RxDn or TxDn). (high)

The frame format used by the USART is set by the UCSZ, UPM and USBS bits (see "UCSRB – USART Control and Status Register B" on page 185 and "UCSRC – USART Control and Status Register C" on page 186), as follows:

- The USART Character SiZe bits (UCSZ) select the number of data bits in the frame
- The USART Parity Mode bits (UPM) choose the type of parity bit
- The selection between one or two stop bits is done by the USART Stop Bit Select bit (USBS). The receiver ignores the second stop bit. An FE (Frame Error) will therefore only be detected in the cases where the first stop bit is zero.

The receiver and transmitter use the same setting. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the receiver and transmitter.

17.4.1 Parity Bit Calculation

The parity bit is calculated by doing an exclusive-or of all the data bits. If odd parity is used, the result of the exclusive or is inverted. The relation between the parity bit and data bits is as follows:

$$P_{EVEN} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{ODD} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

... where:

P_{EVEN}	Parity bit using even parity
P^{ODD}	Parity bit using odd parity
d_n	Data bit n of the character

If used, the parity bit is located between the last data bit and the first stop bit of a serial frame.

17.5 USART Initialization

The USART has to be initialized before any communication can take place. The initialization process normally consists of setting the baud rate, setting frame format and, depending on the method of use, enabling the transmitter or the receiver. For interrupt driven USART operation, the global interrupt flag should be cleared and the USART interrupts should be disabled.

Before re-initializating baud rate or frame format, it should be checked that there are no ongoing transmissions during the period the registers are changed. The TXC flag can be used to check that the transmitter has completed all transfers, and the RXC flag can be used to check that there are no unread data in the receive buffer. Note that, if used, the TXC flag must be cleared before each transmission (before UDR is written).

The following simple USART initialization code examples show one assembly and one C function that are equal in functionality. The examples assume asynchronous operation using polling (no interrupts enabled) and a fixed frame format. The baud rate is given as a function parameter. For the assembly code, the baud rate parameter is assumed to be stored in registers R17:R16.

Assembly Code Example[(1)]

```
USART_Init:
      ; Set baud rate
      out               UBRRH, r17
      out               UBRRL, r16

      ; Enable receiver and transmitter
      ldi               r16, (1<<RXEN)|(1<<TXEN)
      out               UCSRB,r16

      ; Set frame format: 8 data bits, 2 stop bits
      ldi               r16, (1<<USBS)|(3<<UCSZ0)
      out               UCSRC,r16
      ret
```

C Code Example[1]

```
void USART_Init( unsigned int baud )
{
	/* Set baud rate */
	UBRRH = (unsigned char)(baud>>8);
	UBRRL = (unsigned char)baud;

	/* Enable receiver and transmitter */
	UCSRB = (1<<RXEN)|(1<<TXEN);

	/* Set frame format: 8 data bits, 2 stop bits */
	UCSRC = (1<<USBS)|(3<<UCSZ0);
}
```

Note: 1. See “Code Examples” on page 7.

More advanced initialization routines can be made that include frame format as parameters, disable interrupts and so on. However, many applications use a fixed setting of the baud and control registers, and for these types of applications the initialization code can be placed directly in the main routine, or be combined with initialization code for other I/O modules.

17.6 Data Transmission – The USART Transmitter

The USART transmitter is enabled by setting the Transmit Enable bit (TXEN) (see “UCSRB – USART Control and Status Register B” on page 185). When the transmitter is enabled, the normal port operation of the TxDn pin is overridden by the USART and given the function as the transmitter’s serial output. The baud rate, mode of operation and frame format must be set up once before doing any transmissions. If synchronous operation is used, the clock on the XCK pin will be overridden and used as transmission clock.

17.6.1 Sending Frames with 5 to 8 Data Bits

A data transmission is initiated by loading the transmit buffer with the data to be transmitted. The CPU can load the transmit buffer by writing to the UDR register. The buffered data in the transmit buffer will be moved to the shift register when the it is ready to send a new frame. The shift register is loaded with new data 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 new data, it will transfer one complete frame at the rate given by the Baud Rate Register, the U2X bit or by XCK, depending on the mode of operation.

The following code examples show a simple USART transmit function based on polling of the Data Register Empty flag (UDRE). When using frames with less than eight bits, the most significant bits written to UDR are ignored. The USART has to be initialized before the function can be used. For the assembly code, the data to be sent is assumed to be stored in register R16

Assembly Code Example[1]

```
USART_Transmit:
	; Wait for empty transmit buffer
	sbis		UCSRA,UDRE
	rjmp		USART_Transmit

	; Put data (r16) into buffer, sends the data
	out		UDR,r16
	ret
```

C Code Example[1]

```
void USART_Transmit( unsigned char data )
{
      /* Wait for empty transmit buffer */
      while ( !( UCSRA & (1<<UDRE)) )
                        ;

      /* Put data into buffer, sends the data */
      UDR = data;
}
```

Note: 1. See “Code Examples” on page 7.

The function simply waits for the transmit buffer to be empty by checking the UDRE flag, before loading it with new data to be transmitted. If the Data Register Empty interrupt is utilized, the interrupt routine writes the data into the buffer.

17.6.2 Sending Frames with 9 Data Bit

If 9-bit characters are used (UCSZ = 7), the ninth bit must be written to the TXB8 bit in UCSRB before the low byte of the character is written to UDR. The following code examples show a transmit function that handles 9-bit characters. For the assembly code, the data to be sent is assumed to be stored in registers R17:R16.

Assembly Code Example[1][2]

```
USART_Transmit:
      ; Wait for empty transmit buffer
      sbis              UCSRA,UDRE
      rjmp              USART_Transmit

      ; Copy 9th bit from r17 to TXB8
      cbi               UCSRB,TXB8
      sbrc              r17,0
      sbi               UCSRB,TXB8

      ; Put LSB data (r16) into buffer, sends the data
      out               UDR,r16
      ret
```

C Code Example[1][2]

```
void USART_Transmit( unsigned int data )
{
      /* Wait for empty transmit buffer */
      while ( !( UCSRA & (1<<UDRE)) )
                        ;

      /* Copy 9th bit to TXB8 */
      UCSRB &= ~(1<<TXB8);
      if ( data & 0x0100 )
            UCSRB |= (1<<TXB8);

      /* Put data into buffer, sends the data */
      UDR = data;
}
```

Notes: 1. These transmit functions are written to be general functions. They can be optimized if the contents of the UCSRB is static. For example, only the TXB8 bit of UCSRB is used after initialization.

2. See “Code Examples” on page 7.

The ninth bit can be used for indicating an address frame when using multi processor communication mode or for other protocol handling as for example synchronization.

17.6.3 Transmitter Flags and Interrupts

The USART transmitter has two flags that indicate its state: USART Data Register Empty (UDRE) and Transmit Complete (TXC). Both flags can be used for generating interrupts.

The Data Register Empty flag (UDRE) indicates whether the transmit buffer is ready to receive new data. This bit is set when the transmit buffer is empty, and cleared when the transmit buffer contains data to be transmitted that has not yet been moved into the shift register. For compatibility with future devices, always write this bit to zero when writing UCSRA.

When the Data Register Empty Interrupt Enable bit (UDRIE) is set, the USART Data Register Empty Interrupt will be executed as long as UDRE is set (and provided that global interrupts are enabled). UDRE is cleared by writing UDR. When interrupt-driven data transmission is used, the Data Register Empty interrupt routine must either write new data to UDR in order to clear UDRE or disable the Data Register Empty interrupt, otherwise a new interrupt will occur once the interrupt routine terminates.

The Transmit Complete flag (TXC) is set when the entire frame in the transmit shift register has been shifted out and there are no new data currently present in the transmit buffer. The TXC flag is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its location. The TXC flag is useful in half-duplex communication interfaces (like the RS-485 standard), where a transmitting application must enter receive mode and free the communication bus immediately after completing the transmission.

When the Transmit Compete Interrupt Enable bit (TXCIE) is set, the USART Transmit Complete Interrupt will be executed when the TXC flag becomes set (and provided that global interrupts are enabled). When the transmit complete interrupt is used, the interrupt handling routine does not have to clear the TXC flag, since this is done automatically when the interrupt is executed.

17.6.4 Parity Generator

The parity generator calculates the parity bit for the serial frame data. When parity bit is enabled (UPM1 = 1), the transmitter control logic inserts the parity bit between the last data bit and the first stop bit of the frame that is sent.

17.6.5 Disabling the Transmitter

Clearing TXEN will disable the transmitter but the change will not become effective before any ongoing and pending transmissions are completed, i.e. not before the transmit shift register and transmit buffer register are cleared of data to be transmitted. When disabled, the transmitter will no longer override the TxD pin.

17.7 Data Reception – The USART Receiver

The USART receiver is enabled by writing the Receive Enable bit (RXEN) (see “UCSRB – USART Control and Status Register B” on page 185). When the receiver is enabled, the normal operation of the RxD pin is overridden by the USART and given the function as the receiver’s serial input. The baud rate, mode of operation and frame format must be set up once before any serial reception can be done. If synchronous operation is used, the clock on the XCK pin will be used as transfer clock.

17.7.1 Receiving Frames with 5 to 8 Data Bits

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 then 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, i.e., a complete serial frame is present in the receive shift register, the contents of it will be moved into the receive buffer. The receive buffer can then be read by reading UDR.

The following code example shows a simple USART receive function based on polling of the Receive Complete flag (RXC). When using frames with less than eight bits the most significant bits of the data read from the UDR will be masked to zero. The USART has to be initialized before the function can be used.

Assembly Code Example[1]

```
USART_Receive:
      ; Wait for data to be received
      sbis              UCSRA, RXC
      rjmp              USART_Receive

      ; Get and return received data from buffer
      in                r16, UDR
      ret
```

C Code Example[1]

```
unsigned char USART_Receive( void )
{
      /* Wait for data to be received */
      while ( !(UCSRA & (1<<RXC)) )
                        ;

      /* Get and return received data from buffer */
      return UDR;
}
```

Note: 1. See “Code Examples” on page 7.

The function simply waits for data to be present in the receive buffer by checking the RXC flag, before reading the buffer and returning the value.

17.7.2 Receiving Frames with 9 Data Bits

If 9-bit characters are used (UCSZ = 7) the ninth bit must be read from the RXB8 bit before reading the low bits from UDR. This rule applies to the FE, DOR and UPE status flags, as well. Status bits must be read before data from UDR, since reading UDR will change the state of the receive buffer FIFO and, consequently, state of TXB8, FE, DOR and UPE bits.

The following code example shows a simple USART receive function that handles both nine bit characters and the status bits.

Assembly Code Example[(1)]

```
USART_Receive:
      ; Wait for data to be received
      sbis              UCSRA, RXC
      rjmp              USART_Receive

      ; Get status and 9th bit, then data from buffer
      in                r18, UCSRA
      in                r17, UCSRB
      in                r16, UDR

      ; If error, return -1
      andi              r18,(1<<FE)|(1<<DOR)|(1<<UPE)
      breq              USART_ReceiveNoError
      ldi               r17, HIGH(-1)
      ldi               r16, LOW(-1)

USART_ReceiveNoError:
      ; Filter the 9th bit, then return
      lsr               r17
      andi              r17, 0x01
      ret
```

C Code Example[1]

```
unsigned int USART_Receive( void )
{
      unsigned char status, resh, resl;

      /* Wait for data to be received */
      while ( !(UCSRA & (1<<RXC)) )
                        ;

      /* Get status and 9th bit, then data from buffer */
      status = UCSRA;
      resh = UCSRB;
      resl = UDR;

      /* If error, return -1 */
      if ( status & (1<<FE)|(1<<DOR)|(1<<UPE) )
            return -1;

      /* Filter the 9th bit, then return */
      resh = (resh >> 1) & 0x01;
      return ((resh << 8) | resl);
}
```

Note: 1. See “Code Examples” on page 7.

The receive function example reads all the I/O registers into the register file before any computation is done. This gives an optimal receive buffer utilization since the buffer location read will be free to accept new data as early as possible.

17.7.3 Receive Complete Flag and Interrupt

The USART receiver has one flag that indicates the receiver state.

The Receive Complete flag (RXC) indicates if there are unread data present in the receive buffer. This flag is set when unread data exist in the receive buffer, and cleared when the receive buffer is empty (i.e., it does not contain any unread data). If the receiver is disabled (RXEN = 0), the receive buffer will be flushed and, consequently, the RXC bit will become zero.

When the Receive Complete Interrupt Enable (RXCIE) is set, the USART Receive Complete interrupt will be executed as long as the RXC flag is set (and provided that global interrupts are enabled). When interrupt-driven data reception is used, the receive complete routine must read the received data from UDR in order to clear the RXC flag, otherwise a new interrupt will occur once the interrupt routine terminates.

17.7.4 Receiver Error Flags

The USART receiver has three error flags: Frame Error (FE), Data OverRun Error (DOR) and Parity Error (UPE). All error flags are located in the receive buffer together with the frame for which they indicate the error status, and they can be accessed via UCSRA. Due to the buffering of error flags, they must be read before the receive buffer (UDR), since reading UDR changes the buffer.

Error flags can not be changed by software, however, for upward compatibility of future USART implementations all flags must be cleared when UCSRA is written . None of the error flags can generate an interrupt.

- The Frame Error flag (FE) indicates the state of the first stop bit of the next readable frame stored in the receive buffer. The flag is zero when the stop bit was correctly read (as one), and the flag is one when the stop bit was incorrect (zero). This flag can be used for detecting out-of-sync conditions,for detecting break conditions and for

protocol handling. The flag is not affected by the USBS bit, since the receiver ignores all stop bits, except the first. For compatibility with future devices, this bit must always be cleared when writing UCSRA.

- The Data OverRun flag (DOR) indicates data loss due to a receiver buffer full condition. A data overrun situation occurs when the receive buffer is full (two characters), there is a new character waiting in the receive shift register, and a new start bit is detected. If the flag is set there was one or more serial frames lost between the frame last and the next frame read from UDR. For compatibility with future devices, this bit must always be cleared when writing to UCSRA. The flag is cleared when the frame received was successfully moved from the shift register to the receive buffer.
- The Parity Error flag (UPE) indicates that the next frame in the receive buffer had a parity error. If parity check is not enabled the flag will always be zero. For compatibility with future devices, this bit must always be cleared when writing UCSRA. For more details, see “Parity Bit Calculation” on page 168 and “Parity Checker” on page 176.

17.7.5 Parity Checker

The parity checker is active when the high USART Parity Mode bit (UPM1) is set. The type of parity check to be performed (odd or even) is selected by the UPM0 bit. When enabled, the parity checker calculates the parity of the data bits in incoming frames and compares the result with the parity bit from the serial frame. The result of the check is stored in the receive buffer together with the received data and stop bits. The Parity Error flag (UPE) can then be read by software to check if the frame had a parity error.

If parity checking is enabled (UPM = 1), the UPE bit is set if the next character that can be read from the receive buffer had a parity error when received. This bit is valid until the receive buffer (UDR) is read.

17.7.6 Disabling the Receiver

Unlike the transmitter, the receiver is disabled immediately and any data from ongoing receptions will be lost. When disabled (RXEN = 0), the receiver will no longer override the normal function of the RxD port pin and the FIFO buffer is flushed, with any remaining data in the buffer lost.

17.7.7 Flushing the Receive Buffer

The receiver buffer FIFO will be flushed when the receiver is disabled, i.e., the buffer will be emptied of its contents. Unread data will be lost. To flush the buffer during normal operation, due to for instance an error condition, read the UDR until the RXC flag is cleared. The following code example shows how to flush the receive buffer.

Assembly Code Example[1]

```
USART_Flush:
      sbis              UCSRA, RXC
      ret
      in                r16, UDR
      rjmp              USART_Flush
```

C Code Example[1]

```
void USART_Flush( void )
{
      unsigned char dummy;

      while ( UCSRA & (1<<RXC) ) dummy = UDR;
}
```

Note: 1. See “Code Examples” on page 7.

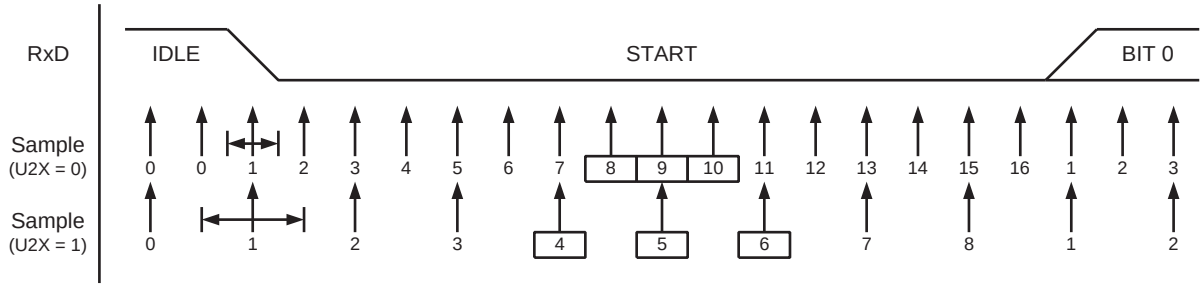
17.8 Asynchronous Data Reception

The USART includes a clock recovery and a data recovery unit for handling asynchronous data reception. The clock recovery logic is used for synchronizing the internally generated baud rate clock to the incoming asynchronous serial frames at the RxD pin. The data recovery logic 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.

17.8.1 Asynchronous Clock Recovery

The clock recovery logic synchronizes the internal clock to the incoming serial frames. Figure 74 illustrates the sampling process of the start bit of an incoming frame. In normal mode the sample rate is 16 times the baud rate, in double speed mode eight times. 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 (U2X = 1). Samples denoted zero are samples done when the RxD line is idle (i.e., no communication activity).

Figure 74. Start Bit Sampling

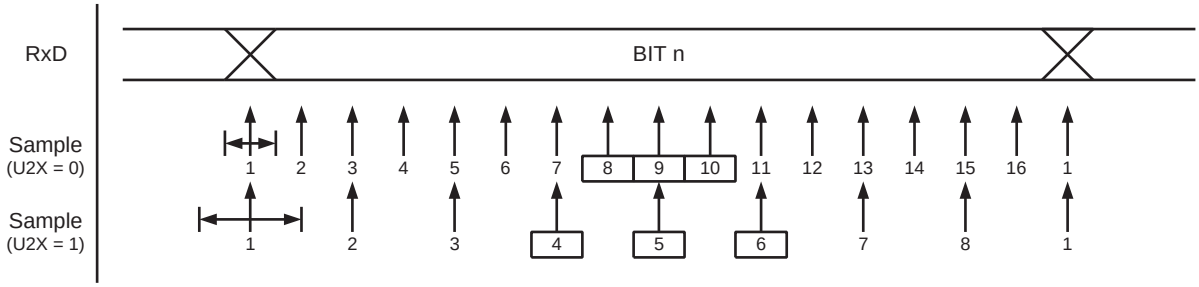


When the clock recovery logic detects a high (idle) to low (start) transition on the RxD line, the start bit detection sequence is initiated. In Figure 74, samples are indicated with numbers inside boxes and sample number 1 denotes the first zero-sample. The clock recovery logic then uses samples 8, 9, and 10 (in normal mode), or samples 4, 5, and 6 (in double speed mode), to decide if a valid start bit is received. If two or more of these three samples have logical high levels (the majority wins), the start bit is rejected as a noise spike and the receiver starts looking for the next high to low-transition. If, however, a valid start bit is detected, the clock recovery logic is synchronized and the data recovery can begin. The synchronization process is repeated for each start bit.

17.8.2 Asynchronous Data Recovery

When the receiver clock is synchronized to the start bit, the data recovery can begin. The data recovery unit uses a state machine that has 16 states for each bit in normal mode and eight states for each bit in double speed mode. Figure 75 shows the sampling of the data bits and the parity bit. Each of the samples is given a number that is equal to the state of the recovery unit.

Figure 75. Sampling of Data and Parity Bit



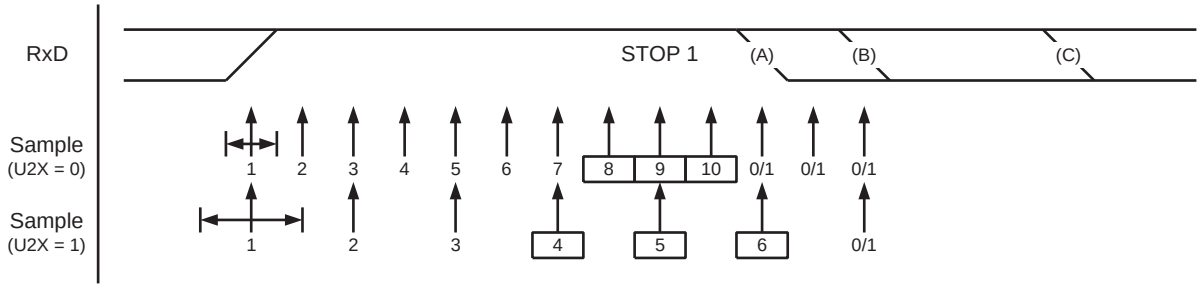
The decision of the logic level of the received bit is taken by doing a majority voting of the logic value to the three samples in the center of the received bit. In the figure, the center samples are emphasized by having the sample number inside

boxes. The majority voting process is done as follows: If two or all three samples have high levels, the received bit is registered to be a logic one. If two, or all three samples have low levels, the received bit is registered to be a logic zero. This majority voting process acts as a low pass filter for the incoming signal on the RxD pin. The recovery process is then repeated until a complete frame is received. Including the first stop bit.

Note that the receiver only uses the first stop bit of a frame.

Figure 76 shows the sampling of the stop bit and the earliest possible beginning of the start bit of the next frame.

Figure 76. Stop Bit Sampling and Next Start Bit Sampling



The stop bit is subject to the same majority voting as the other bits in the frame. If the stop bit is registered to have a logic low value, the Frame Error flag (FE) will be set.

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. In normal speed mode, the first low level sample can be at point marked (A) in Figure 76. In double speed mode the first low level must be delayed to (B). Point (C) marks the full length of a stop bit.

The early start bit detection influences the operational range of the receiver.

17.8.3 Asynchronous Operational Range

The operational range of the receiver depends on the mismatch between the received bit rate and the internally generated baud rate. If the transmitter is sending frames at too fast or too slow bit rates, or the internally generated baud rate of the receiver does not have a similar base frequency (see Table 61 on page 179), 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.

$$R_{slow} = \frac{(D+1)S}{S-1+D\cdot S+S_F} \qquad R_{fast} = \frac{(D+2)S}{(D+1)S+S_M}$$

... where:

- **D** Sum of character size and parity size (D = 5 to 10 bit)
- **S** Samples per bit, 16 for normal speed mode, or 8 for double speed mode.
- **S_F** First sample number used for majority voting, 8 (normal speed), or 4 (double)
- **S_M** Middle sample number for majority voting, 9 (normal speed), or 5 (double speed)
- **R_{slow}** The ratio of the slowest incoming data rate that can be accepted with respect to the receiver baud rate.
- **R_{fast}** The ratio of the fastest incoming data rate that can be accepted with respect to the receiver baud rate.

Table 61 on page 179 and Table 62 on page 179 list the maximum receiver baud rate error that can be tolerated. Note that normal speed mode has higher toleration of baud rate variations.

Table 61. Recommended Maximum Receiver Baud Rate Error in 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 / -6.8	± 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 62. Recommended Maximum Receiver Baud Rate Error in 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 of the maximum receiver baud rate error are made under the assumption that the receiver and transmitter divide the maximum total error equally.

There are two possible sources for the receivers baud rate error:

- The system clock of the receiver will always have some minor instability over the supply voltage range and the temperature range
- The second source for error is more controllable. The baud rate generator can not always do an exact division of the system frequency to get the baud rate wanted. In this case an UBRR value that gives an acceptable low error should be used, if possible

17.8.4 Start Frame Detection

The USART start frame detector can wake up the MCU when it detects a start bit. See Table 9 on page 34.

When a high-to-low transition is detected on RxDn, the internal 8 MHz oscillator is powered up and the USART clock is enabled. After start-up the rest of the data frame can be received, provided that the baud rate is slow enough in relation to the internal 8 MHz oscillator start-up time. Start-up time of the internal 8 MHz oscillator varies with supply voltage and temperature.

The USART start frame detection works both in asynchronous and synchronous modes. It is enabled by writing the Start Frame Detection Enable bit (SFDE) in “UCSRD – USART Control and Status Register D” . If the USART Start Interrupt Enable (RXSIE) bit is set, the USART Receive Start Interrupt is generated immediately when start is detected.

When using the feature without start interrupt, the start detection logic activates the internal 8 MHz oscillator and the USART clock while the frame is being received, only. Other clocks remain stopped until the Receive Complete Interrupt wakes up the MCU.

For more details, see “UCSRD – USART Control and Status Register D” on page 188.

17.9 Multi-processor Communication Mode

Setting the Multi-processor Communication Mode bit (MPCM) enables a filtering function of incoming frames received by the USART receiver. Frames that do not contain address information will be ignored and not put into the receive buffer. In a system with multiple MCUs that communicate via the same serial bus this effectively reduces the number of incoming frames that has to be handled by the CPU. The transmitter is unaffected by the MPCM bit, but has to be used differently when it is a part of a system utilizing the multi-processor communication mode.

If the receiver is set up to receive frames that contain 5 to 8 data bits, then the first stop bit indicates if the frame contains data or address information. If the receiver is set up for frames with nine data bits, then the ninth bit (RXB8) is used for identifying address and data frames. When the frame type bit (the first stop or the ninth bit) is one, the frame contains an address. When the frame type bit is zero the frame is a data frame.

The multi-processor communication mode enables several slave MCUs to receive data from a master MCU. This is done by first decoding an address frame to find out which MCU has been addressed. If a particular slave MCU has been addressed, it will receive the following data frames as normal, while the other slave MCUs will ignore the received frames until another address frame is received.

17.9.1 Using MPCM

For an MCU to act as a master MCU, it can use a 9-bit character frame format (UCSZ = 7). The ninth bit (TXB8) must be set when an address frame is transmitted (TXB8 = 1), and cleared when a data frame is transmitted (TXB = 0) . In this case, the slave MCUs must be set to use a 9-bit character frame format.

The following procedure should be used to exchange data in multi-processor communication mode:

1. All slave MCUs are set to multi-processor communication mode (MPCM = 1)
2. The master MCU sends an address frame, and all slaves receive and read this frame. In the slave MCUs, the RXC flag is set as normal
3. Each slave MCU reads UDR and determines if it has been selected. If so, it clears the MPCM bit. Else, it waits for the next address byte and keeps the MPCM setting
4. The addressed MCU will receive all data frames until a new address frame is received. The other slave MCUs, which still have the MPCM bit set, will ignore the data frames
5. When the last data frame is received by the addressed MCU it sets the MPCM bit and waits for a new address frame from master. The process then repeats from step 2

It is possible to use any of the 5- to 8-bit character frame formats, but impractical since the receiver must change between using n and n+1 character frame formats. This makes full-duplex operation difficult since the transmitter and receiver use the same character size setting. If 5- to 8-bit character frames are used, the transmitter must be set to use two stop bits (USBS = 1), since the first stop bit is used for indicating the frame type.

Do not use Read-Modify-Write instructions (SBI and CBI) to set or clear the MPCM bit. The MPCM bit shares the same I/O location as the TXC flag and this might accidentally be cleared when using SBI or CBI instructions.

17.10 Examples of Baud Rate Setting

Commonly used baud rates for asynchronous operation can be generated by using the UBRR settings in Table 63 to Table 66. UBRR values which yield an actual baud rate differing less than 0.5% from the target baud rate, are shown in bold. Higher error ratings are acceptable, but the receiver will have less noise resistance when the error ratings are high,

especially for large serial frames (see “Asynchronous Operational Range” on page 178). The error values are calculated using the following equation:

$$\text{Error}[\%] = \left(\frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}} - 1\right) \bullet 100\%$$

Table 63. Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f_{osc} = 1.0000MHz				f_{osc} = 1.8432MHz				f_{osc} = 2.0000MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max. (1)	62.5 kbps		125 kbps		115.2 kbps		230.4 kbps		125 kbps		250 kbps	

1. UBRR = 0, Error = 0.0%

Table 64. Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f_{osc} = 3.6864MHz				f_{osc} = 4.0000MHz				f_{osc} = 7.3728MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	95	0.0%	191	0.0%	103	0.2%	207	0.2%	191	0.0%	383	0.0%
4800	47	0.0%	95	0.0%	51	0.2%	103	0.2%	95	0.0%	191	0.0%
9600	23	0.0%	47	0.0%	25	0.2%	51	0.2%	47	0.0%	95	0.0%
14.4k	15	0.0%	31	0.0%	16	2.1%	34	-0.8%	31	0.0%	63	0.0%

Baud Rate (bps)	f_{osc} = 3.6864MHz				f_{osc} = 4.0000MHz				f_{osc} = 7.3728MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
19.2k	11	0.0%	23	0.0%	12	0.2%	25	0.2%	23	0.0%	47	0.0%
28.8k	7	0.0%	15	0.0%	8	-3.5%	16	2.1%	15	0.0%	31	0.0%
38.4k	5	0.0%	11	0.0%	6	-7.0%	12	0.2%	11	0.0%	23	0.0%
57.6k	3	0.0%	7	0.0%	3	8.5%	8	-3.5%	7	0.0%	15	0.0%
76.8k	2	0.0%	5	0.0%	2	8.5%	6	-7.0%	5	0.0%	11	0.0%
115.2k	1	0.0%	3	0.0%	1	8.5%	3	8.5%	3	0.0%	7	0.0%
230.4k	0	0.0%	1	0.0%	0	8.5%	1	8.5%	1	0.0%	3	0.0%
250k	0	-7.8%	1	-7.8%	0	0.0%	1	0.0%	1	-7.8%	3	-7.8%
0.5M	–	–	0	-7.8%	–	–	0	0.0%	0	-7.8%	1	-7.8%
1M	–	–	–	–	–	–	–	–	–	–	0	-7.8%
Max. (1)	230.4 kbps		460.8 kbps		250 kbps		0.5 Mbps		460.8 kbps		921.6 kbps	

1. UBRR = 0, Error = 0.0%

Table 65. Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f_{osc} = 8.0000MHz				f_{osc} = 11.0592MHz				f_{osc} = 14.7456MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%

Baud Rate (bps)	f_{osc} = 8.0000MHz				f_{osc} = 11.0592MHz				f_{osc} = 14.7456MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
0.5M	0	0.0%	1	0.0%	–	–	2	-7.8%	1	-7.8%	3	-7.8%
1M	–	–	0	0.0%	–	–	–	–	0	-7.8%	1	-7.8%
Max. (1)	0.5 Mbps		1 Mbps		691.2 kbps		1.3824 Mbps		921.6 kbps		1.8432 Mbps	

1. UBRR = 0, Error = 0.0%

Table 66. Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f_{osc} = 16.0000MHz				f_{osc} = 18.4320MHz				f_{osc} = 20.0000MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	–	–	4	-7.8%	–	–	4	0.0%
1M	0	0.0%	1	0.0%	–	–	–	–	–	–	–	–
Max. (1)	1 Mbps		2 Mbps		1.152 Mbps		2.304 Mbps		1.25 Mbps		2.5 Mbps	

1. UBRR = 0, Error = 0.0%

17.11 Register Description

17.11.1 UDR – USART I/O Data Register

Bit	7	6	5	4	3	2	1	0	
(0xC6)	RXB[7:0]								UDR (Read)
(0xC6)	TXB[7:0]								UDR (Write)
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 and USART receive data buffer registers share the same I/O address, referred to as USART Data Register, or UDR. Data written to UDR goes to the Transmit Data Buffer register (TXB). Reading UDR returns the contents of the Receive Data Buffer register (RXB).

For 5-, 6-, 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 only be written when the UDRE flag is set. Data written to UDR when the UDRE flag is not set will be ignored. When the transmitter is enabled and data is written to the transmit buffer, the transmitter will load the data into the transmit shift register when it is empty. The data is then serially transmitted on the TxD pin.

The receive buffer consists of a two level FIFO. The FIFO will change its state whenever the receive buffer is accessed. Due to this behavior of the receive buffer, Read-Modify-Write instructions (SBI and CBI) should not be used to access this location. Care should also be taken when using bit test instructions (SBIC and SBIS), since these also change the state of the FIFO.

17.11.2 UCSRA – USART Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0xC0)	RXC	TXC	UDRE	FE	DOR	UPE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 7 – RXC: USART Receive Complete**

This flag is set when there is unread data in the receive buffer, and cleared when the receive buffer is empty (i.e., does not contain any unread data). If the receiver is disabled, the receive buffer will be flushed and consequently the RXC flag will become zero. The flag can be used to generate a Receive Complete interrupt (see RXCIE bit).

- **Bit 6 – TXC: USART Transmit Complete**

This flag is set when the entire frame in the transmit shift register has been shifted out and there is no new data currently present in the transmit buffer (UDR). The TXC flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The flag can generate a Transmit Complete interrupt (see TXCIE bit).

- **Bit 5 – UDRE: USART Data Register Empty**

The UDRE flag indicates the transmit buffer (UDR) is ready to receive new data. If UDRE is one, the buffer is empty, and therefore ready to be written. The UDRE flag can generate a Data Register Empty interrupt (see UDRIE bit).

The UDRE flag is set after a reset to indicate that the transmitter is ready.

- **Bit 4 – FE: Frame Error**

This bit is set if the next character in the receive buffer had a frame error when received (i.e. when the first stop bit of the next character in the receive buffer is zero). This bit is valid until the receive buffer (UDR) is read. The FE bit is zero when the stop bit of received data is one.

Always set this bit to zero when writing the register.

- **Bit 3 – DOR: Data OverRun**

This bit is set if a Data OverRun condition is detected. A data overrun occurs when the receive buffer is full (two characters), there is a new character waiting in the receive shift register, and a new start bit is detected. This bit is valid until the receive buffer (UDR) is read.

Always set this bit to zero when writing the register.

- **Bit 2 – UPE: USART Parity Error**

This bit is set if the next character in the receive buffer had a parity error when received and the parity checking was enabled at that point (UPM1 = 1). This bit is valid until the receive buffer (UDR) is read.

Always set this bit to zero when writing the register.

- **Bit 1 – U2X: Double the USART Transmission Speed**

This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation.

Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8, effectively doubling the transfer rate for asynchronous communication.

- **Bit 0 – MPCM: Multi-processor Communication Mode**

This bit enables the Multi-processor Communication Mode. When the bit is written to one, all the incoming frames received by the USART receiver that do not contain address information will be ignored. The transmitter is unaffected by the MPCM bit. For more detailed information, see “Multi-processor Communication Mode” on page 180.

17.11.3 UCSRB – USART Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
(0xC1)	**RXCIE**	**TXCIE**	**UDRIE**	**RXEN**	**TXEN**	**UCSZ2**	**RXB8**	**TXB8**	**UCSRB**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – RXCIE: RX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the RXC flag. A USART Receive Complete interrupt will be generated only if the RXCIE bit, the Global Interrupt Flag, and the RXC bits are set.

- **Bit 6 – TXCIE: TX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the TXC flag. A USART Transmit Complete interrupt will be generated only if the TXCIE bit, the Global Interrupt Flag, and the TXC bit are set.

- **Bit 5 – UDRIE: USART Data Register Empty Interrupt Enable**

Writing this bit to one enables interrupt on the UDRE flag. A Data Register Empty interrupt will be generated only if the UDRIE bit, the Global Interrupt Flag, and the TXC bit are set.

- **Bit 4 – RXEN: Receiver Enable**

Writing this bit to one enables the USART Receiver. When enabled, the receiver will override normal port operation for the RxD pin. Disabling the receiver will flush the receive buffer, invalidating FE, DOR, and UPE Flags.

- **Bit 3 – TXEN: Transmitter Enable**

Writing this bit to one enables the USART Transmitter. When enabled, the transmitter will override normal port operation for the TxD pin. Disabling the transmitter (writing TXENn 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 – UCSZ2: Character Size**

The UCSZ2 bit combined with the UCSZ[1:0] bits sets the number of data bits (Character SiZe) in a frame the receiver and transmitter use.

- **Bit 1 – RXB8: Receive Data Bit 8**

RXB8 is the ninth data bit of the received character when operating with serial frames with nine data bits. It must be read before reading the low bits from UDR.

- **Bit 0 – TXB8: Transmit Data Bit 8**

TXB8 is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. It must be written before writing the low bits to UDR.

17.11.4 UCSRC – USART Control and Status Register C

Bit	7	6	5	4	3	2	1	0	
(0xC2)	**UMSEL1**	**UMSEL0**	**UPM1**	**UPM0**	**USBS**	**UCSZ1**	**UCSZ0**	**UCPOL**	**UCSRC**
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

- **Bits 7:6 – UMSEL[1:0]: USART Mode Select**

These bits select the mode of operation of the USART, as shown in Table 67.

Table 67. USART Mode of Operation

UMSEL1	UMSEL0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	Reserved
1	1	Master SPI (MSPIM)[1]

Note: 1. For full description of the Master SPI Mode (MSPIM) Operation, see “USART in SPI Mode” on page 190.

- **Bits 5:4 – UPM1:0: Parity Mode**

These bits enable and set type of parity generation and check. If 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 UPM setting. If a mismatch is detected, the UPE flag is set.

Table 68. Parity Mode Selection

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

- **Bit 3 – USBS: Stop Bit Select**

This bit selects the number of stop bits to be inserted by the transmitter. The receiver ignores this setting.

Table 69. Stop Bit Selection

USBS	Stop Bit(s)
0	1-bit
1	2-bit

- **Bits 2:1 – UCSZ[1:0]: Character Size**

Together with the UCSZ2 bit, the UCSZ[1:0] bits sets the number of data bits (Character Size) in a frame the receiver and transmitter use. See Table 70.

Table 70. Character Size Settings

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

- **Bit 0 – UCPOL: Clock Polarity**

This bit is used for synchronous mode only. Write this bit to zero when asynchronous mode is used. The UCPOL bit sets the relationship between data output change and data input sample, and the synchronous clock (XCK).

Table 71. Clock Polarity Settings

UCPOL	Transmitted Data Changed (Output of TxD Pin)	Received Data Sampled (Input on RxD Pin)
0	Rising XCK Edge	Falling XCK Edge
1	Falling XCK Edge	Rising XCK Edge

17.11.5 UCSRD – USART Control and Status Register D

Bit	7	6	5	4	3	2	1	0	
(0xC3)	RXSIE	RXS	SFDE	–	–	–	–	–	UCSRD
Read/Write	R/W	R/W	R	R	R	R	R	R	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 7 – RXSIE: USART RX Start Interrupt Enable**

Writing this bit to one enables the interrupt on the RXS flag. In sleep modes this bit enables start frame detector that can wake up the MCU when a start condition is detected on the RxD line. The USART RX Start Interrupt is generated only, if the RXSIE bit, the Global Interrupt Enable flag, and RXS are set.

- **Bit 6 – RXS: USART RX Start**

The RXS flag is set when a start condition is detected on the RxD line. If the RXSIE bit and the Global Interrupt Enable flag are set, an RX Start Interrupt will be generated when this flag is set. The flag can only be cleared by writing a logical one to the RXS bit location.

If the start frame detector is enabled (RXSIE = 1) and the Global Interrupt Enable Flag is set, the RX Start Interrupt will wake up the MCU from all sleep modes.

- **Bit 5 – SFDE: Start Frame Detection Enable**

Writing this bit to one enables the USART Start Frame Detection mode. The start frame detector is able to wake up the MCU from sleep mode when a start condition, i.e. a high (IDLE) to low (START) transition, is detected on the RxD line.

Table 72. USART Start Frame Detection modes

SFDE	RXSIE	RXCIE	Description
0	X	X	Start frame detector disabled
1	0	0	Reserved
1	0	1	Start frame detector enabled. RXC flag wakes up MCU from all sleep modes
1	1	0	Start frame detector enabled. RXS flag wakes up MCU from all sleep modes
1	1	1	Start frame detector enabled. Both RXC and RXS wake up the MCU from all sleep modes

For more information, see “Start Frame Detection” on page 179.

- **Bits 4:0 – Res: Reserved Bits**

These bits are reserved and will always read zero.

17.11.6 UBRRL and UBRRH – USART Baud Rate Registers

Initial Value	0	0	0	0	0	0	0	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Bit	15	14	13	12	11	10	9	8	
(0xC5)	–	–	–	–	UBRR[11:8]				UBRRH
(0xC4)	UBRR[7:0]								UBRRL
Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 15:12 – Res: Reserved Bits**

These bits are reserved for future use. For compatibility with future devices, these bits must be cleared when UBRRH is written.

- **Bits 11:0 – UBRR[11:0]: USART Baud Rate Register**

This is a 12-bit register which contains the USART baud rate. UBRRH contains the four most significant bits, and UBRRL contains the eight least significant bits of the USART baud rate. Ongoing transmissions by the transmitter and receiver will be corrupted if the baud rate is changed. Writing UBRRL will trigger an immediate update of the baud rate prescaler.

18. USART in SPI Mode

18.1 Features

- Full Duplex, Three-wire Synchronous Data Transfer
- Master Operation
- Supports all four SPI Modes of Operation (Mode 0, 1, 2, and 3)
- LSB First or MSB First Data Transfer (Configurable Data Order)
- Queued Operation (Double Buffered)
- High Resolution Baud Rate Generator
- High Speed Operation (fXCKmax = fCK/2)
- Flexible Interrupt Generation

18.2 Overview

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) can be set to a master SPI compliant mode of operation.

Setting both UMSEL[1:0] bits to one enables the USART in MSPIM logic. In this mode of operation the SPI master control logic takes direct control over the USART resources. These resources include the transmitter and receiver shift register and buffers, and the baud rate generator. The parity generator and checker, the data and clock recovery logic, and the RX and TX control logic is disabled. The USART RX and TX control logic is replaced by a common SPI transfer control logic. However, the pin control logic and interrupt generation logic is identical in both modes of operation.

The I/O register locations are the same in both modes. However, some of the functionality of the control registers changes when using MSPIM.

18.3 Clock Generation

The clock generation logic generates the base clock for the transmitter and receiver. For USART MSPIM mode of operation only internal clock generation (i.e. master operation) is supported. Therefore, for the USART in MSPIM to operate correctly, the Data Direction Register (DDRx) where the XCK pin is located must be configured to set the pin as output (DDR_XCK = 1) . Preferably the DDR_XCK should be set up before the USART in MSPIM is enabled (i.e. before TXEN and RXEN bits are set).

The internal clock generation used in MSPIM mode is identical to the USART synchronous master mode. The baud rate or UBRR setting can therefore be calculated using the same equations, see Table 73:

Table 73. Equations for Calculating Baud Rate Register Setting

Operating Mode	Calculating Baud Rate[1]	Calculating UBRR Value
Synchronous Master mode	$BAUD = \frac{f_{OSC}}{2(\boldsymbol{UBRR}+1)}$	$\boldsymbol{UBRR} = \frac{f_{OSC}}{2BAUD} - 1$

Note: 1. The baud rate is defined as the transfer rate in bits per second (bps)

BAUD	Baud rate (in bits per second, bps)
f_{osc}	System oscillator clock frequency
UBRRn	Contents of UBRRH and UBRRL, (0-4095)

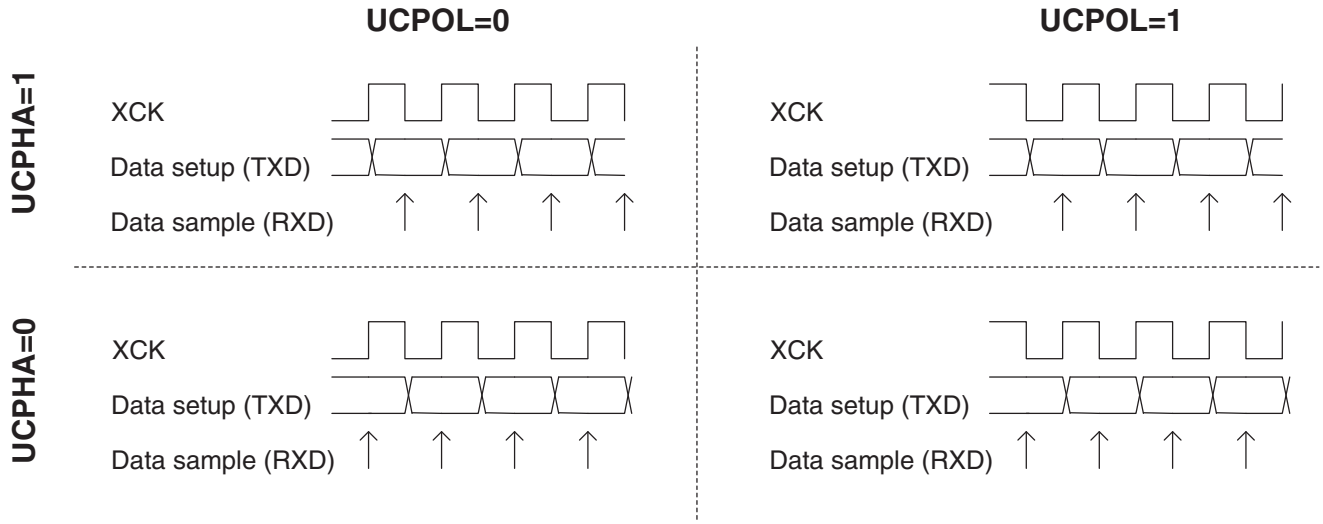
18.4 SPI Data Modes and Timing

There are four combinations of XCK (SCK) phase and polarity with respect to serial data, which are determined by control bits UCPHA and UCPOL. The data transfer timing diagrams are shown in Figure 77. Data bits are shifted out and latched in on opposite edges of the XCK signal, ensuring sufficient time for data signals to stabilize. The UCPOL and UCPHA functionality is summarized in Table 74. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the receiver and transmitter.

Table 74. UCPOLn and UCPHAn Functionality

UCPOL	UCPHA	SPI Mode	Leading Edge	Trailing Edge
0	0	0	Sample (Rising)	Setup (Falling)
0	1	1	Setup (Rising)	Sample (Falling)
1	0	2	Sample (Falling)	Setup (Rising)
1	1	3	Setup (Falling)	Sample (Rising)

Figure 77. UCPHA and UCPOL data transfer timing diagrams.



18.5 Frame Formats

A serial frame for the MSPIM is defined to be one character of 8 data bits. The USART in MSPIM mode has two valid frame formats:

- 8-bit data with MSB first
- 8-bit data with LSB first

A frame starts with the least or most significant data bit. Then follows the next data bits, up to a total of eight, ending with the most or least significant bit, accordingly. When a complete frame is transmitted, a new frame can directly follow it, or the communication line can be set to an idle (high) state.

The UDORD bit sets the frame format used by the USART in MSPIM mode. The receiver and transmitter use the same setting. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the receiver and transmitter.

16-bit data transfer can be achieved by writing two data bytes to UDR. A USART Transmit Complete interrupt will then signal that the 16-bit value has been shifted out.

18.5.1 USART MSPIM Initialization

The USART in MSPIM mode has to be initialized before any communication can take place. The initialization process normally consists of setting the baud rate, setting master mode of operation, setting frame format and enabling the transmitter and the receiver. Only the transmitter can operate independently. For interrupt driven USART operation, the Global Interrupt Flag should be cleared (and thus interrupts globally disabled) when doing the initialization.

Note: To ensure immediate initialization of the XCK output the baud-rate register (UBRR) must be zero at the time the transmitter is enabled. Contrary to the normal mode USART operation the UBRR must then be written to the desired value after the transmitter is enabled, but before the first transmission is started. Setting UBRR to zero before enabling the transmitter is not necessary if the initialization is done immediately after a reset since UBRR is reset to zero.

Before doing a re-initialization with changed baud rate, data mode, or frame format, be sure that there is no ongoing transmissions during the period the registers are changed. The TXC flag can be used to check that the transmitter has completed all transfers, and the RXC flag can be used to check that there are no unread data in the receive buffer. Note that the TXC flag must be cleared before each transmission (before UDR is written), if it is used for this purpose.

The following simple USART initialization code examples show one assembly and one C function that are equal in functionality. The examples assume polling (no interrupts enabled). The baud rate is given as a function parameter. For the assembly code, the baud rate parameter is assumed to be stored in registers R17:R16.

Assembly Code Example[1]

```
USART_Init:
      clr r18
      out UBRRH,r18
      out UBRRL,r18

      ; Setting the XCK port pin as output, enables master mode.
      sbi XCK_DDR, XCK

      ; Set MSPI mode of operation and SPI data mode 0.
      ldi r18, (1<<UMSEL1)|(1<<UMSEL0)|(0<<UCPHA)|(0<<UCPOL)
      out UCSRC,r18

      ; Enable receiver and transmitter.
      ldi r18, (1<<RXEN)|(1<<TXEN)
      out UCSRB,r18

      ; Set baud rate.
      ; IMPORTANT: Baud Rate must be set after the transmitter is enabled!
      out UBRRH, r17
      out UBRRL, r18
      ret
```

C Code Example[1]

```
void USART_Init( unsigned int baud )
{
      UBRR = 0;

      /* Setting the XCK port pin as output, enables master mode. */
      XCK_DDR |= (1<<XCK);

      /* Set MSPI mode of operation and SPI data mode 0. */
      UCSRC = (1<<UMSEL1)|(1<<UMSEL0)|(0<<UCPHA)|(0<<UCPOL);

      /* Enable receiver and transmitter. */
      UCSRB = (1<<RXEN)|(1<<TXEN);

      /* Set baud rate. */
      /* IMPORTANT: Baud Rate must be set after transmitter is enabled */
      UBRR = baud;
}
```

Note: 1. See “Code Examples” on page 7.

18.6 Data Transfer

Using the USART in MSPI mode requires the transmitter to be enabled, i.e. the TXEN bit to be set. When the transmitter is enabled, the normal port operation of the TxD pin is overridden and given the function as the transmitter's serial output. Enabling the receiver is optional and is done by setting the RXEN bit. When the receiver is enabled, the normal pin operation of the RxD pin is overridden and given the function as the receiver's serial input. The XCK will in both cases be used as the transfer clock.

After initialization the USART is ready for doing data transfers. A data transfer is initiated by writing to UDR. This is the case for both sending and receiving data since the transmitter controls the transfer clock. The data written to UDR is moved from the transmit buffer to the shift register when the shift register is ready to send a new frame.

Note: To keep the input buffer in sync with the number of data bytes transmitted, UDR must be read once for each byte transmitted. The input buffer operation is identical to normal USART mode, i.e. if an overflow occurs the character last received will be lost, not the first data in the buffer. This means that if four bytes are transferred, byte 1 first, then byte 2, 3, and 4, and the UDR is not read before all transfers are completed, then byte 3 to be received will be lost, and not byte 1.

The following code examples show a simple USART in MSPIM mode transfer function based on polling of the Data Register Empty flag (UDRE) and the Receive Complete flag (RXC). The USART has to be initialized before the function can be used. For the assembly code, the data to be sent is assumed to be stored in register R16 and the data received will be available in the same register (R16) after the function returns.

The function simply waits for the transmit buffer to be empty by checking the UDRE flag, before loading it with new data to be transmitted. The function then waits for data to be present in the receive buffer by checking the RXC flag, before reading the buffer and returning the value..

Assembly Code Example[1]

```
USART_MSPIM_Transfer:
      ; Wait for empty transmit buffer
      sbis UCSRA, UDRE
      rjmp USART_MSPIM_Transfer

      ; Put data (r16) into buffer, sends the data
      out UDR,r16

USART_MSPIM_Wait_RXC:
      ; Wait for data to be received
      sbis UCSRA, RXC
      rjmp USART_MSPIM_Wait_RXC

      ; Get and return received data from buffer
      in r16, UDR
      ret
```

C Code Example[1]

```
unsigned char USART_Receive( void )
{
      /* Wait for empty transmit buffer */
      while ( !( UCSRA & (1<<UDRE)) );

      /* Put data into buffer, sends the data */
      UDR = data;

      /* Wait for data to be received */
      while ( !(UCSRA & (1<<RXC)) );

      /* Get and return received data from buffer */
      return UDR;
}
```

Note: 1. See “Code Examples” on page 7.

18.6.1 Transmitter and Receiver Flags and Interrupts

The RXC, TXC, and UDRE flags and corresponding interrupts in USART in MSPIM mode are identical in function to the normal USART operation. However, the receiver error status flags (FE, DOR, and PE) are not in use and always read zero.

18.6.2 Disabling the Transmitter or Receiver

The disabling of the transmitter or receiver in USART in MSPIM mode is identical in function to the normal USART operation.

18.7 AVR USART MSPIM vs. AVR SPI

The USART in MSPIM mode is fully compatible with the AVR SPI regarding:

- Master mode timing diagram
- The UCPOL bit functionality is identical to the SPI CPOL bit
- The UCPHA bit functionality is identical to the SPI CPHA bit
- The UDORD bit functionality is identical to the SPI DORD bit

However, since the USART in MSPIM mode reuses the USART resources, the use of the USART in MSPIM mode is somewhat different compared to the SPI. In addition to differences of the control register bits, and that only master operation is supported by the USART in MSPIM mode, the following features differ between the two modules:

- The USART in MSPIM mode includes (double) buffering of the transmitter. The SPI has no buffer.
- The USART in MSPIM mode receiver includes an additional buffer level.
- The SPI WCOL (Write Collision) bit is not included in USART in MSPIM mode.
- The SPI double speed mode (SPI2X) bit is not included. However, the same effect is achieved by setting UBRR accordingly.
- Interrupt timing is not compatible.
- Pin control differs due to the master only operation of the USART in MSPIM mode.

A comparison of the USART in MSPIM mode and the SPI pins is shown in Table 75.

Table 75. Comparison of USART in MSPIM mode and SPI pins

USART_MSPIM	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 MSPIM

18.8 Register Description

The following section describes the registers used for SPI operation using the USART.

18.8.1 UDR – USART MSPIM I/O Data Register

The function and bit description of the USART data register (UDR) in MSPI mode is identical to normal USART operation. See “UDR – USART I/O Data Register” on page 184.

18.8.2 UCSRA – USART MSPIM Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0xC0)	RXC	TXC	UDRE	–	–	–	–	–	UCSRA
Read/Write	R/W	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	1	1	0	

- **Bit 7 – RXC: USART Receive Complete**

This flag bit 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). If the receiver is disabled, the receive buffer will be flushed and consequently the RXC bit will become zero. The RXC flag can be used to generate a Receive Complete interrupt (see RXCIE bit).

- **Bit 6 – TXC: USART Transmit Complete**

This flag bit is set when the entire frame in the transmit shift register has been shifted out and there are no new data currently present in the transmit buffer (UDR). The TXC flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXC flag can generate a Transmit Complete interrupt (see TXCIE bit).

- **Bit 5 – UDRE: USART Data Register Empty**

The UDRE flag indicates if the transmit buffer (UDR) is ready to receive new data. If UDRE is one, the buffer is empty, and therefore ready to be written. The UDRE flag can generate a Data Register Empty interrupt (see UDRIE bit). UDRE is set after a reset to indicate that the transmitter is ready.

- **Bits 4:0 – Reserved Bits in MSPI mode**

When in MSPI mode, these bits are reserved for future use. For compatibility with future devices, these bits must be written to zero when UCSRA is written.

18.8.3 UCSRB – USART MSPIM Control and Status Register n B

Bit	7	6	5	4	3	2	1	0	
(0xC1)	RXCIE	TXCIE	UDRIE	RXEN	TXEN	–	–	–	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R	R	R	
Initial Value	0	0	0	0	0	1	1	0	

- **Bit 7 – RXCIE: RX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the RXC flag. A USART Receive Complete interrupt will be generated only if the RXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXC bit in UCSRA is set.

- **Bit 6 – TXCIE: TX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the TXC flag. A USART Transmit Complete interrupt will be generated only if the TXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXC bit in UCSRA is set.

- **Bit 5 – UDRIE: USART Data Register Empty Interrupt Enable**

Writing this bit to one enables interrupt on the UDRE flag. A Data Register Empty interrupt will be generated only if the UDRIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDRE bit in UCSRA is set.

- **Bit 4 – RXEN: Receiver Enable**

Writing this bit to one enables the USART Receiver in MSPIM mode. The receiver will override normal port operation for the RxD pin when enabled. Disabling the receiver will flush the receive buffer. Only enabling the receiver in MSPI mode (i.e. setting RXEN=1 and TXEN=0) has no meaning since it is the transmitter that controls the transfer clock and since only master mode is supported.

- **Bit 3 – TXEN: Transmitter Enable**

Writing this bit to one enables the USART Transmitter. The transmitter will override normal port operation for the TxD pin when enabled. The disabling of 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.

- **Bits 2:0 – Reserved Bits in MSPI mode**

When in MSPI mode, these bits are reserved for future use. For compatibility with future devices, these bits must be written to zero when UCSRB is written.

18.8.4 UCSRC – USART MSPIM Control and Status Register C

Bit	7	6	5	4	3	2	1	0	
(0xC2)	UMSEL1	UMSEL0	-	-	-	UDORD	UCPHA	UCPOL	UCSRC
Read/Write	R/W	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

- **Bits 7:6 – UMSEL[1:0]: USART Mode Select**

These bits select the mode of operation of the USART as shown in Table 76. See “UCSRC – USART Control and Status Register C” on page 186 for full description of the normal USART operation. The MSPIM is enabled when both UMSEL bits are set to one. The UDORD, UCPHA, and UCPOL can be set in the same write operation where the MSPIM is enabled.

Table 76. UMSEL Bits Settings

UMSEL1	UMSEL0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)

- **Bits 5:3 – Reserved Bits in MSPI mode**

When in MSPI mode, these bits are reserved for future use. For compatibility with future devices, these bits must be written to zero when UCSRC is written.

- **Bit 2 – UDORD: Data Order**

When set to one the LSB of the data word is transmitted first. When set to zero the MSB of the data word is transmitted first. See “Frame Formats” on page 191 for details.

- **Bit 1 – UCPHA: Clock Phase**

The UCPHA bit setting determine if data is sampled on the leasing edge (first) or tailing (last) edge of XCK. See “SPI Data Modes and Timing” on page 191 for details.

- **Bit 0 – UCPOL: Clock Polarity**

The UCPOL bit sets the polarity of the XCK clock. The combination of the UCPOL and UCPHA bit settings determine the timing of the data transfer. See “SPI Data Modes and Timing” on page 191 for details.

18.8.5 UBRRL and UBRRH – USART MSPIM Baud Rate Registers

The function and bit description of the baud rate registers in MSPI mode is identical to normal USART operation. See “UBRRL and UBRRH – USART Baud Rate Registers” on page 189.

19. I²C Compatible, Two-Wire Slave Interface

19.1 Features

- I²C compatible
- SMBus compatible (with reservations)
- 100kHz and 400kHz support at low system clock frequencies
- Slew-Rate Limited Output Drivers
- Input Filter provides noise suppression
- 7-bit, and General Call Address Recognition in Hardware
- Address mask register for address masking or dual address match
- 10-bit addressing supported
- Optional Software Address Recognition Provides Unlimited Number of Slave Addresses
- Operates in all sleep modes, including Power Down
- Slave Arbitration allows support for SMBus Address Resolve Protocol (ARP)

19.2 Overview

The Two Wire Interface (TWI) is a bi-directional, bus communication interface, which uses only two wires. The TWI is I²C compatible and, with reservations, SMBus compatible (see “Compatibility with SMBus” on page 205).

A device connected to the bus must act as a master or slave.The master initiates a data transaction by addressing a slave on the bus, and telling whether it wants to transmit or receive data. One bus can have several masters, and an arbitration process handles priority if two or more masters try to transmit at the same time.

The TWI module in ATtiny828 implements slave functionality, only. Lost arbitration, errors, collisions and clock holds on the bus are detected in hardware and indicated in separate status flags.

Both 7-bit and general address call recognition is implemented in hardware. 10-bit addressing is also supported. A dedicated address mask register can act as a second address match register or as a mask register for the slave address to match on a range of addresses. The slave logic continues to operate in all sleep modes, including Power down. This enables the slave to wake up from sleep on TWI address match. It is possible to disable the address matching and let this be handled in software instead. This allows the slave to detect and respond to several addresses. Smart Mode can be enabled to auto trigger operations and reduce software complexity.

The TWI module includes bus state logic that collects information to detect START and STOP conditions, bus collision and bus errors. The bus state logic continues to operate in all sleep modes including Power down.

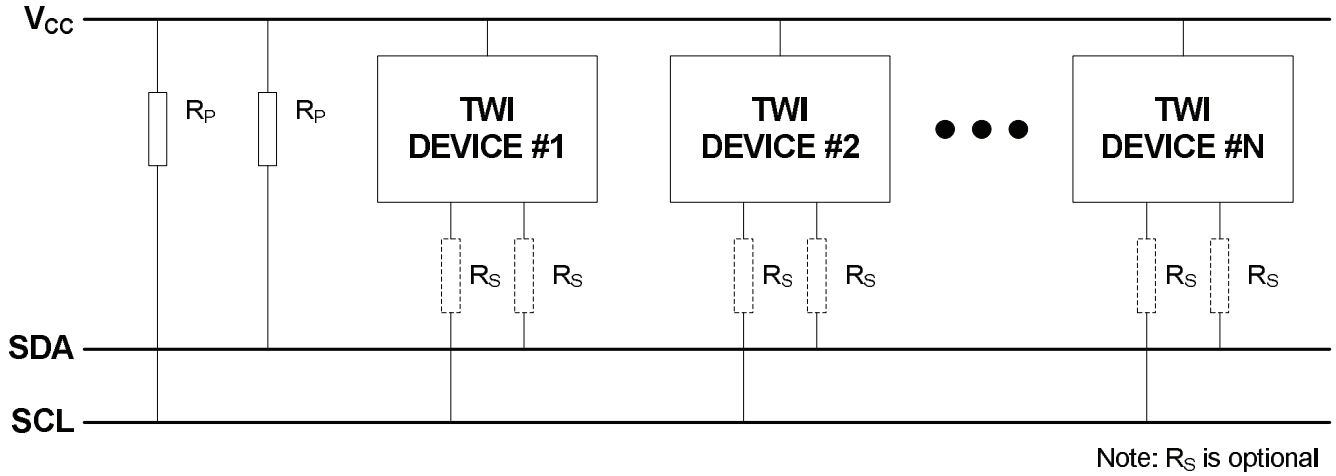
19.3 General TWI Bus Concepts

The Two-Wire Interface (TWI) provides a simple two-wire bi-directional bus consisting of a serial clock line (SCL) and a serial data line (SDA). The two lines are open collector lines (wired-AND), and pull-up resistors (Rp) are the only external components needed to drive the bus. The pull-up resistors will provide a high level on the lines when none of the connected devices are driving the bus. A constant current source can be used as an alternative to the pull-up resistors.

The TWI bus is a simple and efficient method of interconnecting multiple devices on a serial bus. A device connected to the bus can be a master or slave, where the master controls the bus and all communication.

Figure 78 illustrates the TWI bus topology.

Figure 78. TWI Bus Topology



A unique address is assigned to all slave devices connected to the bus, and the master will use this to address a slave and initiate a data transaction. 7-bit or 10-bit addressing can be used.

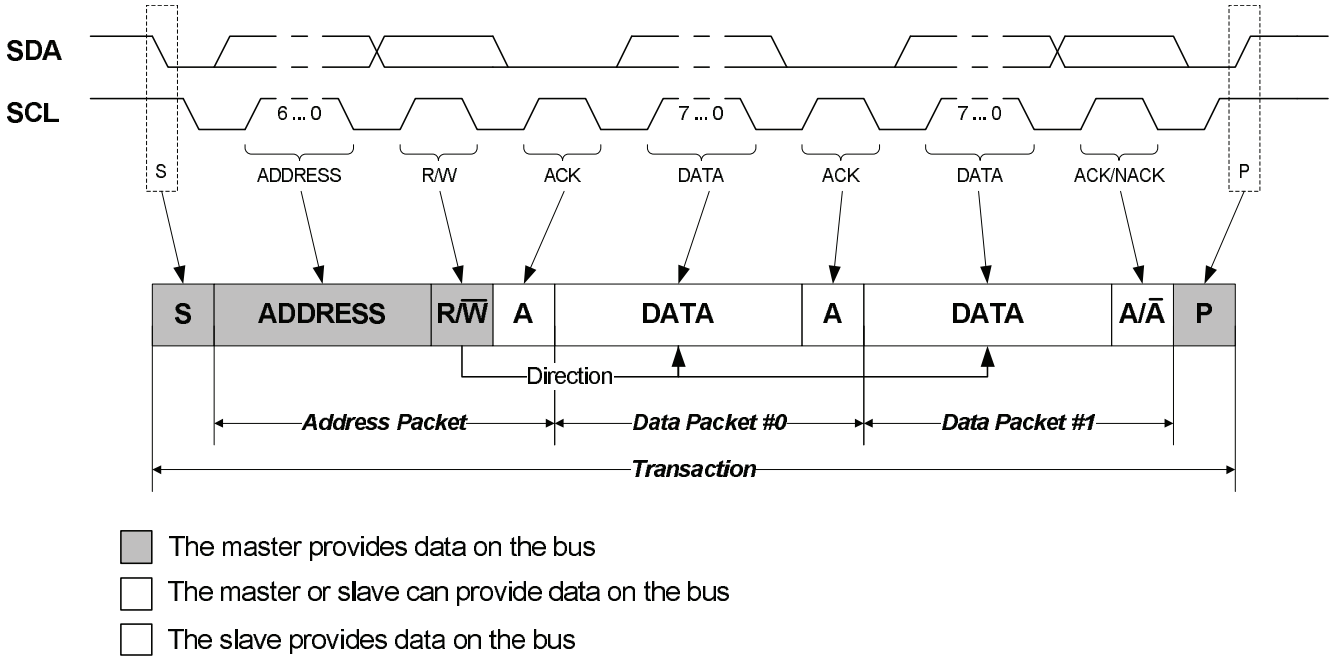
Several masters can be connected to the same bus, and this is called a multi-master environment. An arbitration mechanism is provided for resolving bus ownership between masters since only one master device may own the bus at any given time.

A device can contain both master and slave logic, and can emulate multiple slave devices by responding to more than one address.

A master indicates the start of transaction by issuing a START condition (S) on the bus. An address packet with a slave address (ADDRESS) and an indication whether the master wishes to read or write data (R/$\overline{W}$), is then sent. After all data packets (DATA) are transferred, the master issues a STOP condition (P) on the bus to end the transaction. The receiver must acknowledge (A) or not-acknowledge ($\overline{A}$) each byte received.

Figure 79 shows a TWI transaction.

Figure 79. Basic TWI Transaction Diagram Topology



The master provides the clock signal for the transaction, but a device connected to the bus is allowed to stretch the low level period of the clock to decrease the clock speed.

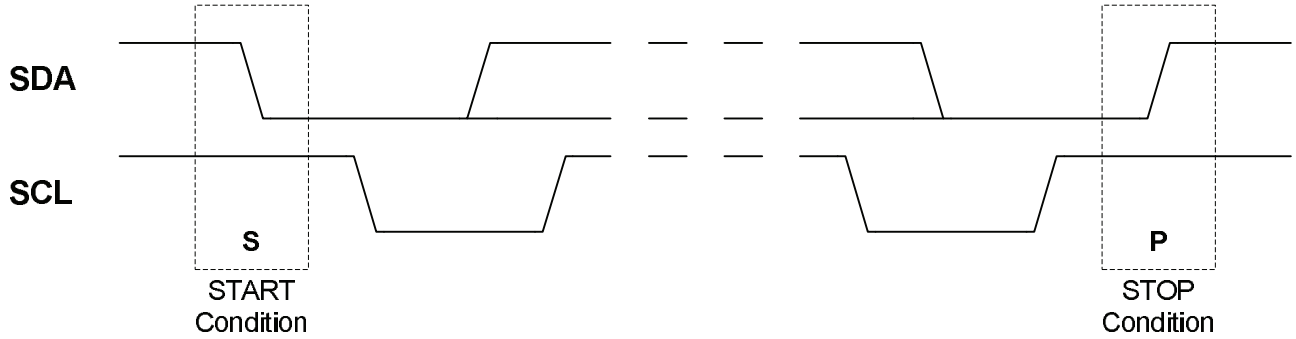
19.3.1 Electrical Characteristics

The TWI follows the electrical specifications and timing of I^2C and SMBus. See “Two-Wire Serial Interface Characteristics” on page 252 and “Compatibility with SMBus” on page 205.

19.3.2 START and STOP Conditions

Two unique bus conditions are used for marking the beginning (START) and end (STOP) of a transaction. The master issues a START condition(S) by indicating a high to low transition on the SDA line while the SCL line is kept high. The master completes the transaction by issuing a STOP condition (P), indicated by a low to high transition on the SDA line while SCL line is kept high.

Figure 80. START and STOP Conditions

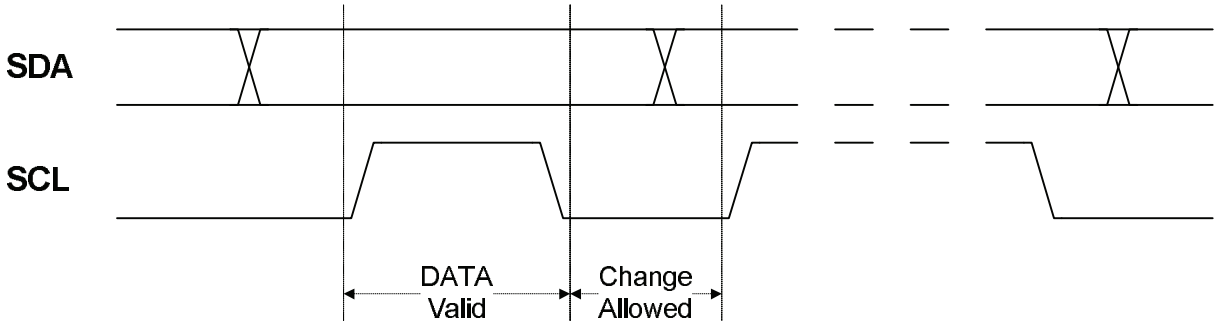


Multiple START conditions can be issued during a single transaction. A START condition not directly following a STOP condition, are named a Repeated START condition (Sr).

19.3.3 Bit Transfer

As illustrated by Figure 81 a bit transferred on the SDA line must be stable for the entire high period of the SCL line. Consequently the SDA value can only be changed during the low period of the clock. This is ensured in hardware by the TWI module.

Figure 81. Data Validity



Combining bit transfers results in the formation of address and data packets. These packets consist of 8 data bits (one byte) with the most significant bit transferred first, plus a single bit not-acknowledge (NACK) or acknowledge (ACK) response. The addressed device signals ACK by pulling the SCL line low, and NACK by leaving the line SCL high during the ninth clock cycle.

19.3.4 Address Packet

After the START condition, a 7-bit address followed by a read/write (R/$\overline{W}$) bit is sent. This is always transmitted by the Master. A slave recognizing its address will ACK the address by pulling the data line low the next SCL cycle, while all other slaves should keep the TWI lines released, and wait for the next START and address. The 7-bit address, the R/$\overline{W}$ bit and the acknowledge bit combined is the address packet. Only one address packet for each START condition is given, also when 10-bit addressing is used.

The R/$\overline{W}$ specifies the direction of the transaction. If the R/$\overline{W}$ bit is low, it indicates a Master Write transaction, and the master will transmit its data after the slave has acknowledged its address. Opposite, for a Master Read operation the slave will start to transmit data after acknowledging its address.

19.3.5 Data Packet

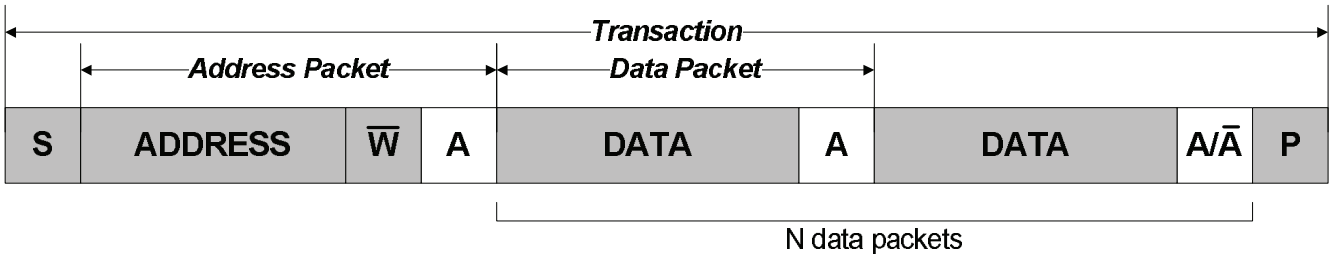
Data packets succeed an address packet or another data packet. All data packets are nine bits long, consisting of one data byte and an acknowledge bit. The direction bit in the previous address packet determines the direction in which the data is transferred.

19.3.6 Transaction

A transaction is the complete transfer from a START to a STOP condition, including any Repeated START conditions in between. The TWI standard defines three fundamental transaction modes: Master Write, Master Read, and combined transaction.

Figure 82 illustrates the Master Write transaction. The master initiates the transaction by issuing a START condition (S) followed by an address packet with direction bit set to zero (ADDRESS+$\overline{W}$).

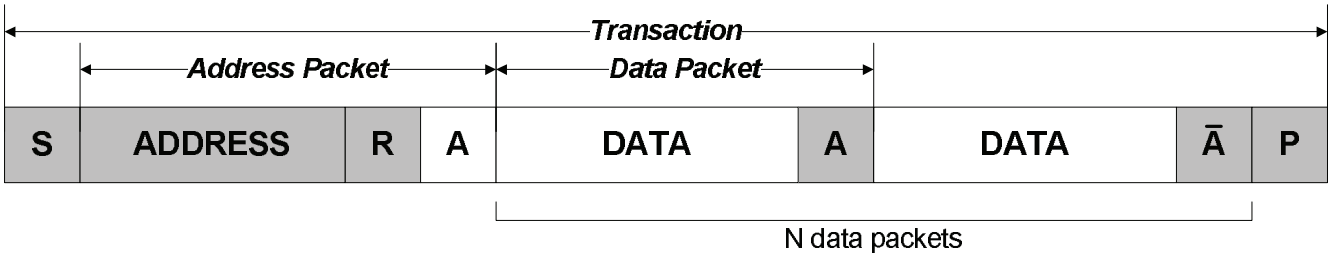
Figure 82. Master Write Transaction



Given that the slave acknowledges the address, the master can start transmitting data (DATA) and the slave will ACK or NACK (A/$\overline{A}$) each byte. If no data packets are to be transmitted, the master terminates the transaction by issuing a STOP condition (P) directly after the address packet. There are no limitations to the number of data packets that can be transferred. If the slave signal a NACK to the data, the master must assume that the slave cannot receive any more data and terminate the transaction.

Figure 83 illustrates the Master Read transaction. The master initiates the transaction by issuing a START condition followed by an address packet with direction bit set to one (ADRESS+R). The addressed slave must acknowledge the address for the master to be allowed to continue the transaction.

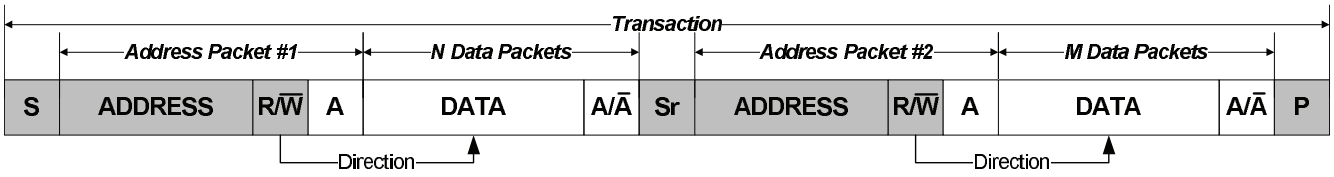
Figure 83. Master Read Transaction



Given that the slave acknowledges the address, the master can start receiving data from the slave. There are no limitations to the number of data packets that can be transferred. The slave transmits the data while the master signals ACK or NACK after each data byte. The master terminates the transfer with a NACK before issuing a STOP condition.

Figure 84 illustrates a combined transaction. A combined transaction consists of several read and write transactions separated by a Repeated START conditions (Sr).

Figure 84. Combined Transaction

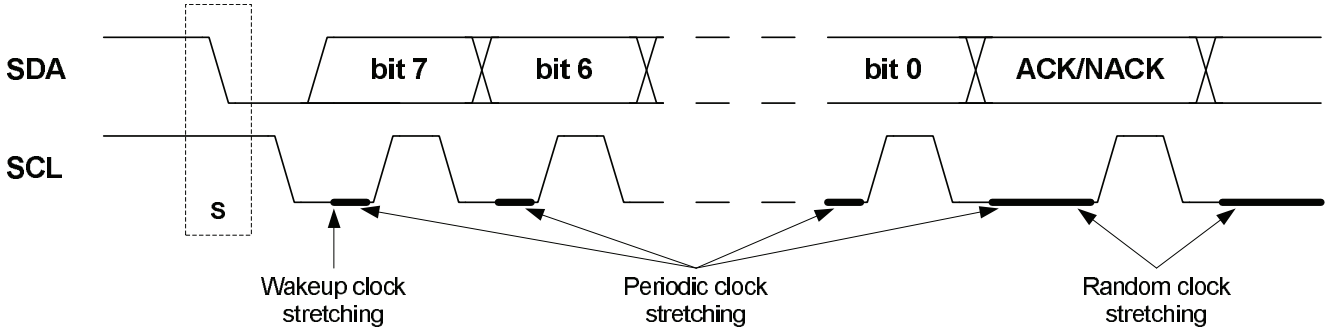


19.3.7 Clock and Clock Stretching

All devices connected to the bus are allowed to stretch the low period of the clock to slow down the overall clock frequency or to insert wait states while processing data. A device that needs to stretch the clock can do this by holding/forcing the SCL line low after it detects a low level on the line.

Three types of clock stretching can be defined as shown in Figure 85.

Figure 85. Clock Stretching



If the device is in a sleep mode and a START condition is detected the clock is stretched during the wake-up period for the device.

A slave device can slow down the bus frequency by stretching the clock periodically on a bit level. This allows the slave to run at a lower system clock frequency. However, the overall performance of the bus will be reduced accordingly. Both the master and slave device can randomly stretch the clock on a byte level basis before and after the ACK/NACK bit. This provides time to process incoming or prepare outgoing data, or performing other time critical tasks.

In the case where the slave is stretching the clock the master will be forced into a wait-state until the slave is ready and vice versa.

19.3.8 Arbitration

A master can only start a bus transaction if it has detected that the bus is idle. As the TWI bus is a multi master bus, it is possible that two devices initiate a transaction at the same time. This results in multiple masters owning the bus simultaneously. This is solved using an arbitration scheme where the master loses control of the bus if it is not able to transmit a high level on the SDA line. The masters who lose arbitration must then wait until the bus becomes idle (i.e. wait for a STOP condition) before attempting to reacquire bus ownership. Slave devices are not involved in the arbitration procedure.

Figure 86. TWI Arbitration

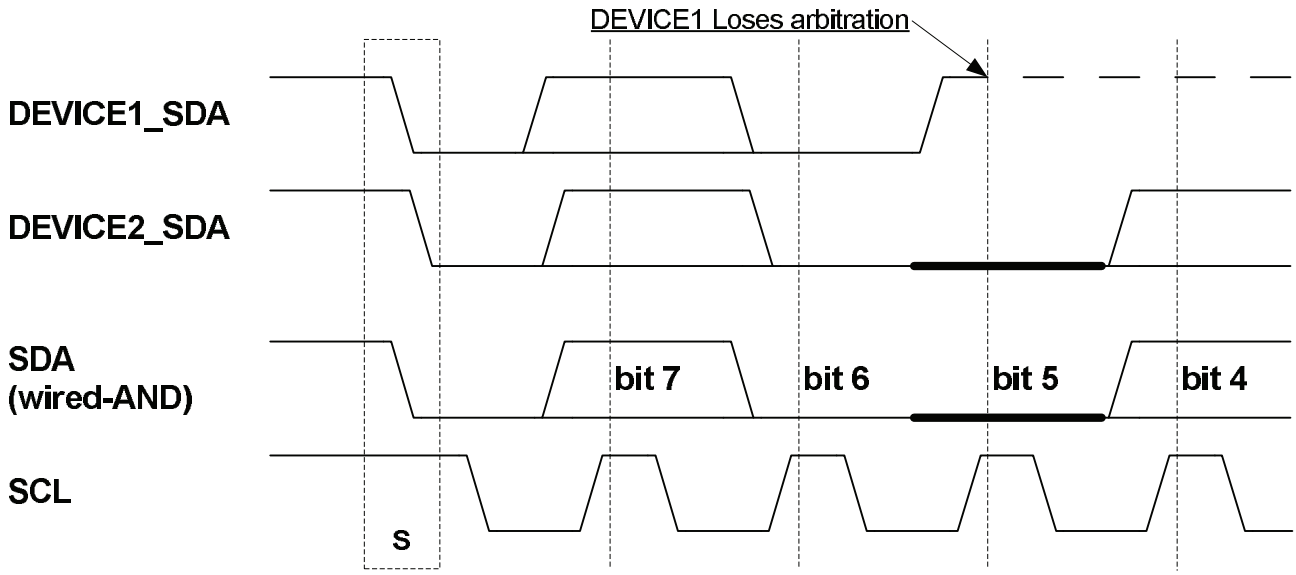


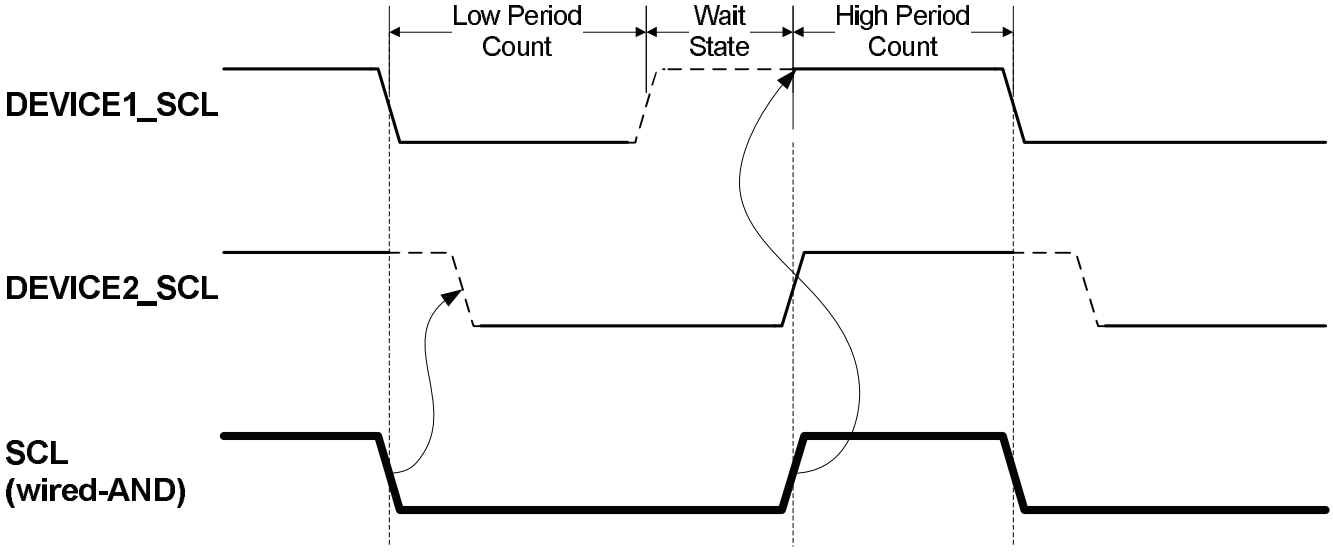
Figure 86 shows an example where two TWI masters are contending for bus ownership. Both devices are able to issue a START condition, but DEVICE1 loses arbitration when attempting to transmit a high level (bit 5) while DEVICE2 is transmitting a low level.

Arbitration between a repeated START condition and a data bit, a STOP condition and a data bit, or a repeated START condition and STOP condition are not allowed and will require special handling by software.

19.3.9 Synchronization

A clock synchronization algorithm is necessary for solving situations where more than one master is trying to control the SCL line at the same time. The algorithm is based on the same principles used for clock stretching previously described. Figure 87 shows an example where two masters are competing for the control over the bus clock. The SCL line is the wired-AND result of the two masters clock outputs.

Figure 87. Clock Synchronization



A high to low transition on the SCL line will force the line low for all masters on the bus and they start timing their low clock period. The timing length of the low clock period can vary between the masters. When a master (DEVICE1 in this

case) has completed its low period it releases the SCL line. However, the SCL line will not go high before all masters have released it. Consequently the SCL line will be held low by the device with the longest low period (DEVICE2). Devices with shorter low periods must insert a wait-state until the clock is released. All masters start their high period when the SCL line is released by all devices and has become high. The device which first completes its high period (DEVICE1) forces the clock line low and the procedure are then repeated. The result of this is that the device with the shortest clock period determines the high period while the low period of the clock is determined by the longest clock period.

19.3.10 Compatibility with SMBus

As with any other I^2C-compliant interface there are known compatibility issues the designer should be aware of before connecting a TWI device to SMBus devices. For use in SMBus environments, the following should be noted:

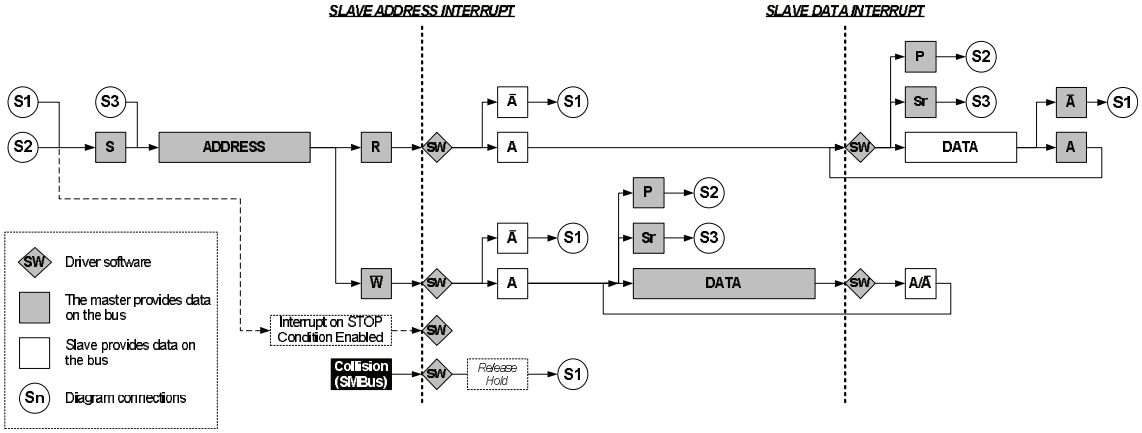
- All I/O pins of an AVR, including those of the two-wire interface, have protection diodes to both supply voltage and ground. See Figure 21 on page 59. This is in contradiction to the requirements of the SMBus specifications. As a result, supply voltage mustn't be removed from the AVR or the protection diodes will pull the bus lines down. Power down and sleep modes is not a problem, provided supply voltages remain.
- The data hold time of the TWI is lower than specified for SMBus. The TWSHE bit of TWSCRA can be used to increase the hold time. See "TWSCRA – TWI Slave Control Register A" on page 207.
- SMBus has a low speed limit, while I^2C hasn't. As a master in an SMBus environment, the AVR must make sure bus speed does not drop below specifications, since lower bus speeds trigger timeouts in SMBus slaves. If the AVR is configured a slave there is a possibility of a bus lockup, since the TWI module doesn't identify timeouts.

19.4 TWI Slave Operation

The TWI slave is byte-oriented with optional interrupts after each byte. There are separate interrupt flags for Data Interrupt and Address/Stop Interrupt. Interrupt flags can be set to trigger the TWI interrupt, or be used for polled operation. There are dedicated status flags for indicating ACK/NACK received, clock hold, collision, bus error and read/write direction.

When an interrupt flag is set, the SCL line is forced low. This will give the slave time to respond or handle any data, and will in most cases require software interaction. Figure 88. shows the TWI slave operation. The diamond shapes symbols (SW) indicate where software interaction is required.

Figure 88. TWI Slave Operation



The number of interrupts generated is kept at a minimum by automatic handling of most conditions. Quick Command can be enabled to auto trigger operations and reduce software complexity.

Promiscuous Mode can be enabled to allow the slave to respond to all received addresses.

19.4.1 Receiving Address Packets

When the TWI slave is properly configured, it will wait for a START condition to be detected. When this happens, the successive address byte will be received and checked by the address match logic, and the slave will ACK the correct address. If the received address is not a match, the slave will not acknowledge the address and wait for a new START condition.

The slave Address/Stop Interrupt Flag is set when a START condition succeeded by a valid address packet is detected. A general call address will also set the interrupt flag.

A START condition immediately followed by a STOP condition, is an illegal operation and the Bus Error flag is set.

The R/W Direction flag reflects the direction bit received with the address. This can be read by software to determine the type of operation currently in progress.

Depending on the R/W direction bit and bus condition one of four distinct cases (1 to 4) arises following the address packet. The different cases must be handled in software.

19.4.1.1 Case 1: Address packet accepted - Direction bit set

If the R/$\overline{W}$ Direction flag is set, this indicates a master read operation. The SCL line is forced low, stretching the bus clock. If ACK is sent by the slave, the slave hardware will set the Data Interrupt Flag indicating data is needed for transmit. If NACK is sent by the slave, the slave will wait for a new START condition and address match.

19.4.1.2 Case 2: Address packet accepted - Direction bit cleared

If the R/$\overline{W}$ Direction flag is cleared this indicates a master write operation. The SCL line is forced low, stretching the bus clock. If ACK is sent by the slave, the slave will wait for data to be received. Data, Repeated START or STOP can be received after this. If NACK is indicated the slave will wait for a new START condition and address match.

19.4.1.3 Case 3: Collision

If the slave is not able to send a high level or NACK, the Collision flag is set and it will disable the data and acknowledge output from the slave logic. The clock hold is released. A START or repeated START condition will be accepted.

19.4.1.4 Case 4: STOP condition received.

Operation is the same as case 1 or 2 above with one exception. When the STOP condition is received, the Slave Address/Stop flag will be set indicating that a STOP condition and not an address match occurred.

19.4.2 Receiving Data Packets

The slave will know when an address packet with R/$\overline{W}$ direction bit cleared has been successfully received. After acknowledging this, the slave must be ready to receive data. When a data packet is received the Data Interrupt Flag is set, and the slave must indicate ACK or NACK. After indicating a NACK, the slave must expect a STOP or Repeated START condition.

19.4.3 Transmitting Data Packets

The slave will know when an address packet, with R/$\overline{W}$ direction bit set, has been successfully received. It can then start sending data by writing to the Slave Data register. When a data packet transmission is completed, the Data Interrupt Flag is set. If the master indicates NACK, the slave must stop transmitting data, and expect a STOP or Repeated START condition.

19.5 Register Description

19.5.1 TWSCRA – TWI Slave Control Register A

Bit	7	6	5	4	3	2	1	0	
(0xB8)	TWSHE	–	TWDIE	TWASIE	TWEN	TWSIE	TWPME	TWSME	TWSCRA
Read/Write	R/W	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 – TWSHE: TWI SDA Hold Time Enable**

When this bit is set the internal hold time on SDA with respect to the negative edge on SCL is enabled.

- **Bit 6 – Res: Reserved Bit**

This bit is reserved and will always read as zero.

- **Bit 5 – TWDIE: TWI Data Interrupt Enable**

When this bit is set and interrupts are enabled, a TWI interrupt will be generated when the data interrupt flag (TWDIF) in TWSSRA is set.

- **Bit 4 – TWASIE: TWI Address/Stop Interrupt Enable**

When this bit is set and interrupts are enabled, a TWI interrupt will be generated when the address/stop interrupt flag (TWASIF) in TWSSRA is set.

- **Bit 3 – TWEN: Two-Wire Interface Enable**

When this bit is set the slave Two-Wire Interface is enabled.

- **Bit 2 – TWSIE: TWI Stop Interrupt Enable**

Setting the Stop Interrupt Enable (TWSIE) bit will set the TWASIF in the TWSSRA register when a STOP condition is detected.

- **Bit 1 – TWPME: TWI Promiscuous Mode Enable**

When this bit is set the address match logic of the slave TWI responds to all received addresses. When this bit is cleared the address match logic uses the TWSA register to determine which address to recognize as its own.

- **Bit 0 – TWSME: TWI Smart Mode Enable**

When this bit is set the TWI slave enters Smart Mode, where the Acknowledge Action is sent immediately after the TWI data register (TWSD) has been read. Acknowledge Action is defined by the TWAA bit in TWSCRB.

When this bit is cleared the Acknowledge Action is sent after TWCMDn bits in TWSCRB are written to 1X.

19.5.2 TWSCRB – TWI Slave Control Register B

Bit	7	6	5	4	3	2	1	0	
(0xB9)	–	–	–	–	–	TWAA	TWCMD1	TWCMD0	TWSCRB
Read/Write	R	R	R	R	R	R/W	W	W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:3 – Res: Reserved Bits**

These bits are reserved and will always read as zero.

- **Bit 2 – TWAA: TWI Acknowledge Action**

This bit defines the slave's acknowledge behavior after an address or data byte has been received from the master. Depending on the TWSME bit in TWSCRA the Acknowledge Action is executed either when a valid command has been written to TWCMDn bits, or when the data register has been read. Acknowledge action is also executed if clearing TWAIF flag after address match or TWDIF flag during master transmit. See Table 77 for details.

Table 77. Acknowledge Action of TWI Slave

TWAA	Action	TWSME	When
0	Send ACK	0	When TWCMDn bits are written to 10 or 11
		1	When TWSD is read
1	Send NACK	0	When TWCMDn bits are written to 10 or 11
		1	When TWSD is read

- **Bits 1:0 – TWCMD[1:0]: TWI Command**

Writing these bits triggers the slave operation as defined by Table 78. The type of operation depends on the TWI slave interrupt flags, TWDIF and TWASIF. The Acknowledge Action is only executed when the slave receives data bytes or address byte from the master.

Table 78. TWI Slave Command

TWCMD[1:0]	TWDIR	Operation
00	X	No action
01	X	Reserved
10	Used to complete transaction	
	0	Execute Acknowledge Action, then wait for any START (S/Sr) condition
	1	Wait for any START (S/Sr) condition

TWCMD[1:0]	TWDIR	Operation
11	**Used in response to an Address Byte (TWASIF is set)**	
	0	Execute Acknowledge Action, then receive next byte
	1	Execute Acknowledge Action, then set TWDIF
	Used in response to a Data Byte (TWDIF is set)	
	0	Execute Acknowledge Action, then wait for next byte
	1	No action

Writing the TWCMD bits will automatically release the SCL line and clear the TWCH and slave interrupt flags.

TWAA and TWCMDn bits can be written at the same time. Acknowledge Action will then be executed before the command is triggered.

The TWCMDn bits are strobed and always read zero.

19.5.3 TWSSRA – TWI Slave Status Register A

Bit	7	6	5	4	3	2	1	0	
(0xBA)	**TWDIF**	**TWASIF**	**TWCH**	**TWRA**	**TWC**	**TWBE**	**TWDIR**	**TWAS**	**TWSSRA**
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 – TWDIF: TWI Data Interrupt Flag**

This flag is set when a data byte has been successfully received, i.e. no bus errors or collisions have occurred during the operation. When this flag is set the slave forces the SCL line low, stretching the TWI clock period. The SCL line is released by clearing the interrupt flags.

Writing a one to this bit will clear the flag. This flag is also automatically cleared when writing a valid command to the TWCMDn bits in TWSCRB.

- **Bit 6 – TWASIF: TWI Address/Stop Interrupt Flag**

This flag is set when the slave detects that a valid address has been received, or when a transmit collision has been detected. When this flag is set the slave forces the SCL line low, stretching the TWI clock period. The SCL line is released by clearing the interrupt flags.

If TWASIE in TWSCRA is set, a STOP condition on the bus will also set TWASIF. STOP condition will set the flag only if system clock is faster than the minimum bus free time between STOP and START.

Writing a one to this bit will clear the flag. This flag is also automatically cleared when writing a valid command to the TWCMDn bits in TWSCRB.

- **Bit 5 – TWCH: TWI Clock Hold**

This bit is set when the slave is holding the SCL line low.

This bit is read-only, and set when TWDIF or TWASIF is set. The bit can be cleared indirectly by clearing the interrupt flags and releasing the SCL line.

- **Bit 4 – TWRA: TWI Receive Acknowledge**

This bit contains the most recently received acknowledge bit from the master.

This bit is read-only. When zero, the most recent acknowledge bit from the maser was ACK and, when one, the most recent acknowledge bit was NACK.

- **Bit 3 – TWC: TWI Collision**

This bit is set when the slave was not able to transfer a high data bit or a NACK bit. When a collision is detected, the slave will commence its normal operation, and disable data and acknowledge output. No low values are shifted out onto the SDA line.

This bit is cleared by writing a one to it. The bit is also cleared automatically when a START or Repeated START condition is detected.

- **Bit 2 – TWBE: TWI Bus Error**

This bit is set when an illegal bus condition has occured during a transfer. An illegal bus condition occurs if a Repeated START or STOP condition is detected, and the number of bits from the previous START condition is not a multiple of nine.

This bit is cleared by writing a one to it.

- **Bit 1 – TWDIR: TWI Read/Write Direction**

This bit indicates the direction bit from the last address packet received from a master. When this bit is one, a master read operation is in progress. When the bit is zero a master write operation is in progress.

- **Bit 0 – TWAS: TWI Address or Stop**

This bit indicates why the TWASIF bit was last set. If zero, a stop condition caused TWASIF to be set. If one, address detection caused TWASIF to be set.

19.5.4 TWSA – TWI Slave Address Register

Bit	7	6	5	4	3	2	1	0	
(0xBC)	TWSA[7:0]								TWSA
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 slave address register contains the TWI slave address used by the slave address match logic to determine if a master has addressed the slave. When using 7-bit or 10-bit address recognition mode, the high seven bits of the address register (TWSA[7:1]) represent the slave address. The least significant bit (TWSA0) is used for general call address recognition. Setting TWSA0 enables general call address recognition logic.

When using 10-bit addressing the address match logic only support hardware address recognition of the first byte of a 10-bit address. If TWSA[7:1] is set to "0b11110nn", 'nn' will represent bits 9 and 8 of the slave address. The next byte received is then bits 7 to 0 in the 10-bit address, but this must be handled by software.

When the address match logic detects that a valid address byte has been received, the TWASIF is set and the TWDIR flag is updated.

If TWPME in TWSCRA is set, the address match logic responds to all addresses transmitted on the TWI bus. TWSA is not used in this mode.

19.5.5 TWSD – TWI Slave Data Register

Bit	7	6	5	4	3	2	1	0	
(0xBD)	TWSD[7:0]								TWSD
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 when transmitting and received data. During transfer, data is shifted from/to the TWSD register and to/from the bus. Therefore, the data register cannot be accessed during byte transfers. This is protected in hardware. The data register can only be accessed when the SCL line is held low by the slave, i.e. when TWCH is set.

When a master reads data from a slave, the data to be sent must be written to the TWSD register. The byte transfer is started when the master starts to clock the data byte from the slave. It is followed by the slave receiving the acknowledge bit from the master. The TWDIF and the TWCH bits are then set.

When a master writes data to a slave, the TWDIF and the TWCH flags are set when one byte has been received in the data register. If Smart Mode is enabled, reading the data register will trigger the bus operation, as set by the TWAA bit in TWSCRB.

Accessing TWSD will clear the slave interrupt flags and the TWCH bit.

19.5.6 TWSAM – TWI Slave Address Mask Register

Bit	7	6	5	4	3	2	1	0	
(0xBB)	TWSAM[7:1]							TWAE	TWSAM
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:1 – TWSAM[7:1]: TWI Address Mask**

These bits can act as a second address match register, or an address mask register, depending on the TWAE setting.

If TWAE is set to zero, TWSAM can be loaded with a 7-bit slave address mask. Each bit in TWSAM can mask (disable) the corresponding address bit in the TWSA register. If the mask bit is one the address match between the incoming address bit and the corresponding bit in TWSA is ignored. In other words, masked bits will always match.

If TWAE is set to one, TWSAM can be loaded with a second slave address in addition to the TWSA register. In this mode, the slave will match on 2 unique addresses, one in TWSA and the other in TWSAM.

- **Bit 0 – TWAE: TWI Address Enable**

By default, this bit is zero and the TWSAM bits acts as an address mask to the TWSA register. If this bit is set to one, the slave address match logic responds to the two unique addresses in TWSA and TWSAM.

20. debugWIRE On-chip Debug System

20.1 Features

- Complete Program Flow Control
- Emulates All On-chip Functions, Both Digital and Analog , except RESET Pin
- Real-time Operation
- Symbolic Debugging Support (Both at C and Assembler Source Level, or for Other HLLs)
- Unlimited Number of Program Break Points (Using Software Break Points)
- Non-intrusive Operation
- Electrical Characteristics Identical to Real Device
- Automatic Configuration System
- High-Speed Operation
- Programming of Non-volatile Memories

20.2 Overview

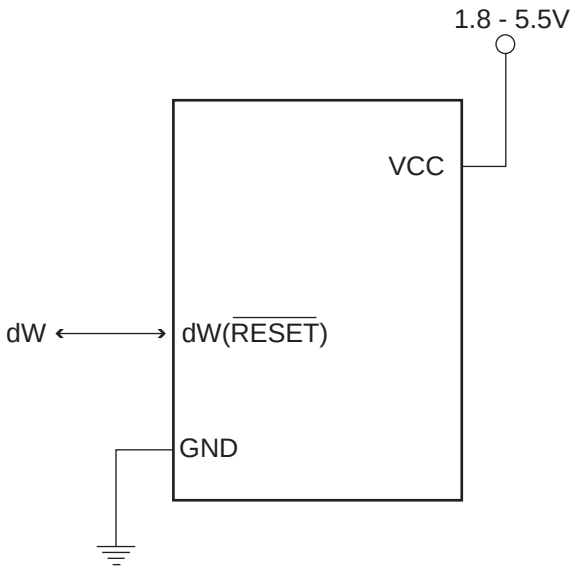
The debugWIRE On-chip debug system uses a One-wire, bi-directional interface to control the program flow, execute AVR instructions in the CPU and to program the different non-volatile memories.

20.3 Physical Interface

When the debugWIRE Enable (DWEN) Fuse is programmed and Lock bits are unprogrammed, the debugWIRE system within the target device is activated. The RESET port pin is configured as a wire-AND (open-drain) bi-directional I/O pin with pull-up enabled and becomes the communication gateway between target and emulator.

Figure 89 shows the schematic of a target MCU, with debugWIRE enabled, and the emulator connector. The system clock is not affected by debugWIRE and will always be the clock source selected by the CKSEL Fuses.

Figure 89. The debugWIRE Setup



When designing a system where debugWIRE will be used, the following must be observed:

- Pull-Up resistor on the dW/(RESET) line must be in the range of 10k to 20 kΩ. However, the pull-up resistor is optional.
- Connecting the RESET pin directly to V_{CC} will not work.
- Capacitors inserted on the RESET pin must be disconnected when using debugWire.
- All external reset sources must be disconnected.

20.4 Software Break Points

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

The Flash must be re-programmed each time a Break Point is changed. This is automatically handled by AVR Studio through the debugWIRE interface. The use of Break Points will therefore reduce the Falsh Data retention. Devices used for debugging purposes should not be shipped to end customers.

20.5 Limitations of debugWIRE

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

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

The debugWIRE interface is asynchronous, which means that the debugger needs to synchronize to the system clock. If the system clock is changed by software (e.g. by writing CLKPS bits) communication via debugWIRE may fail. Also, clock frequencies below 100kHz may cause communication problems.

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

20.6 Register Description

The following section describes the registers used with the debugWire.

20.6.1 DWDR – debugWire Data Register

Bit	7	6	5	4	3	2	1	0	
0x31 (0x51)	DWDR[7:0]								DWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The DWDR Register provides a communication channel from the running program in the MCU to the debugger. This register is only accessible by the debugWIRE and can therefore not be used as a general purpose register in the normal operations.

21. Self-Programming with Boot Loader and Read-While-Write

21.1 Features

- Self-Programming Enables MCU to Erase, Write and Reprogram Application Memory
- Efficient Read-Modify-Write Support
- Support for Read-While-Write Self-Programming
- Boot Loader Section With Variable Size
- Lock Bits Allow Application Memory to Be Securely Closed for Further Access
- Separate Boot Lock Bits Allow High Security and Flexible Protection Schemes
- Separate Fuse for Setting Reset Vector
- Optimized Flash Page Size
- Code Efficient Algorithm

21.2 Overview

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

If boot loader is not needed the entire Flash can be made available for application code.

21.3 Application and Boot Loader Flash Sections

The Flash memory is organized in two main section; the application section and the Boot Loader Section (BLS). See Figure 91. The size of the different sections is configured by the BOOTSZ fuses, as shown in Table 82 on page 217. These two sections can have different levels of protection since they have different sets of lock bits.

21.3.1 Application Section

The application section is the part of Flash that is used for storing the application code. The protection level for the application section can be selected by the application boot lock bit (Boot Lock Bit 0). See Table 87 on page 226.

The application section can never store any boot loader code, since the SPM instruction is disabled when executed from the application section.

21.3.2 BLS – 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 (BLS), since the SPM instruction can initiate a programming when executed from the BLS, only.

The SPM instruction can access the entire Flash, including the BLS itself. The protection level for the BLS can be selected by the boot loader lock bit (Boot Lock Bit 1). See Table 88 on page 226.

21.4 Read-While-Write and No Read-While-Write Flash Sections

During a software update by the boot loader the CPU either works in read-while-write -mode or is halted during the operation. The type of operation dependens on which address is being programmed.

In addition to the application and boot loader areas, the Flash is also divided into two fixed sections; the Read-While-Write (RWW) and the No Read-While-Write (NRWW) section. The main difference between the RWW and NRWW sections are:

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

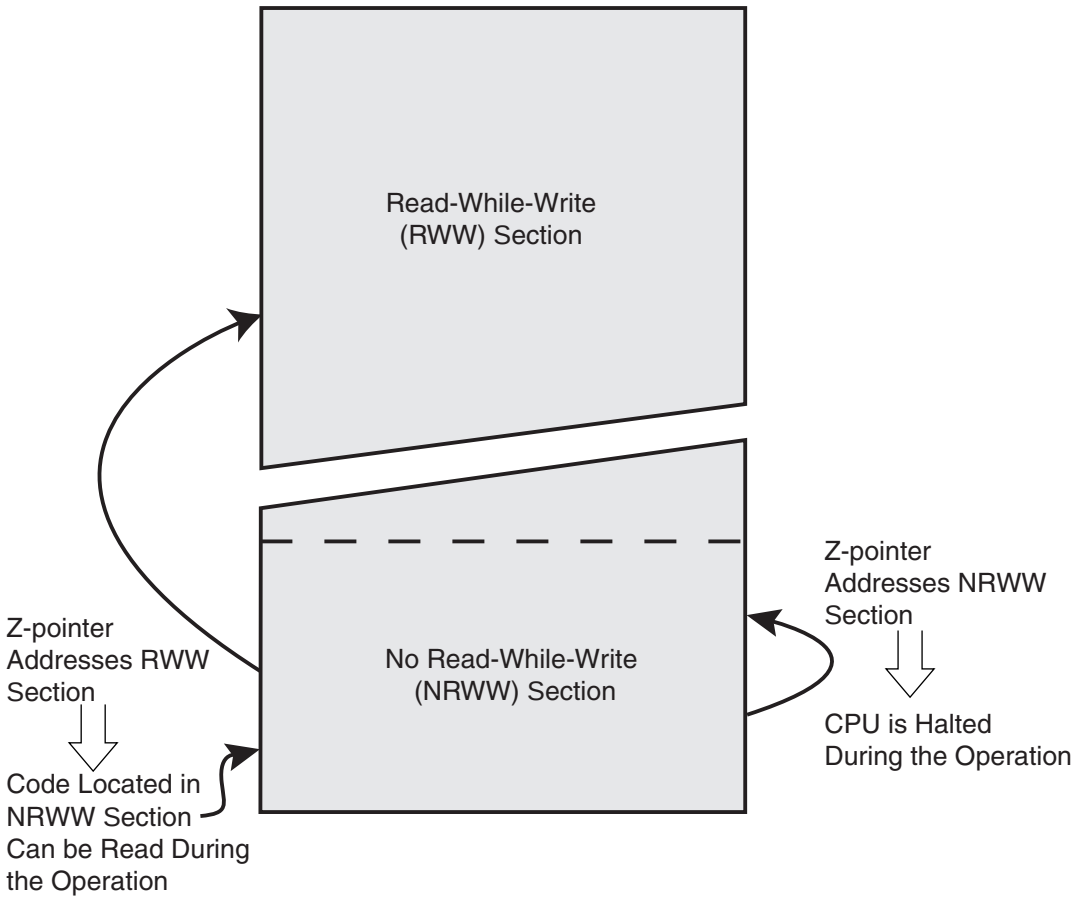
Note that during a boot loader operation, software can never read code located in the RWW section. See Table 79, below.

Table 79. Read-While-Write Features

Section that Z-pointer addresses during programming	Section that can be read during programming?	CPU halted	Read-while-write supported
RWW	NRWW	No	Yes
NRWW	None	Yes	No

The term “read-while-write section” refers to the section being programmed (i.e. erased, or written), not the section that is being read during a software update by the boot loader.

Figure 90. Read-While-Write vs. No Read-While-Write



RWW and NRWW sections are defined in Table 82 on page 217 and illustrated in Figure 90.

21.4.1 RWW – Read-While-Write Section

When the boot loader is programming a page inside the RWW section, it is possible to read code from the Flash, but only code that is located in the NRWW section.

During programming, the software must ensure that the RWW section is not read. If the software during programming tries to read code located inside the RWW section (i.e., by a call/jmp/lpm or an interrupt), it may end up in an unknown state. To avoid this, interrupts should either be disabled or moved to the BLS, which is always located in the NRWW section. The RWW Section Busy bit (RWWSB) in the Store Program Memory Control and Status Register (SPMCSR) is set as long as the RWW section is blocked for reading. After a programming is completed, the RWWSB must be cleared by software before reading code located in the RWW section.

21.4.2 NRWW – No Read-While-Write Section

Code located in the NRWW section can be read when the boot loader is updating a page in the RWW section. When the boot loader updates the NRWW section, the CPU is halted during the entire Page Erase or Page Write operation.

21.5 Entering the Boot Loader Program

Entering the boot loader takes place by a jump or call from the application program. This may be initiated by a trigger, such as a command received via USART, or SPI. Alternatively, the Boot Reset fuse (BOOTRST) can be programmed so that the reset vector is pointing to the boot loader start address, in which case the boot loader is started after a reset. See Table 80, below.

Table 80. Boot Reset Fuse

BOOTRST (1)	Reset Vector
1	Application reset address (0x0000)
0	Boot loader (see Table 82 on page 217)

Note: 1. “1” means unprogrammed, “0” means programmed

After the application code has been loaded, the boot program can start executing the application code.

Note that fuses cannot be changed by the MCU itself. This means that once the BOOTRST fuse is programmed, the reset vector will always point to the boot loader and the fuse can only be changed through the serial or parallel programming interface.

21.6 Configuring the Boot Loader

Read-While-Write (RWW) and No Read-While-Write(NRWW) sections of the Flash are constant, as shown in Table 81. For details on these two sections, see “Read-While-Write and No Read-While-Write Flash Sections” on page 214.

Table 81. Read-While-Write and No Read-While-Write Sections of the Flash

Section	Flash Pages	Address
Read-While-Write section (RWW)	96	0x000 - 0xBFF
No Read-While-Write section (NRWW)	32	0xC00 - 0xFFF

The size of application and boot loader sections can be changed, as shown in Table 82 on page 217.

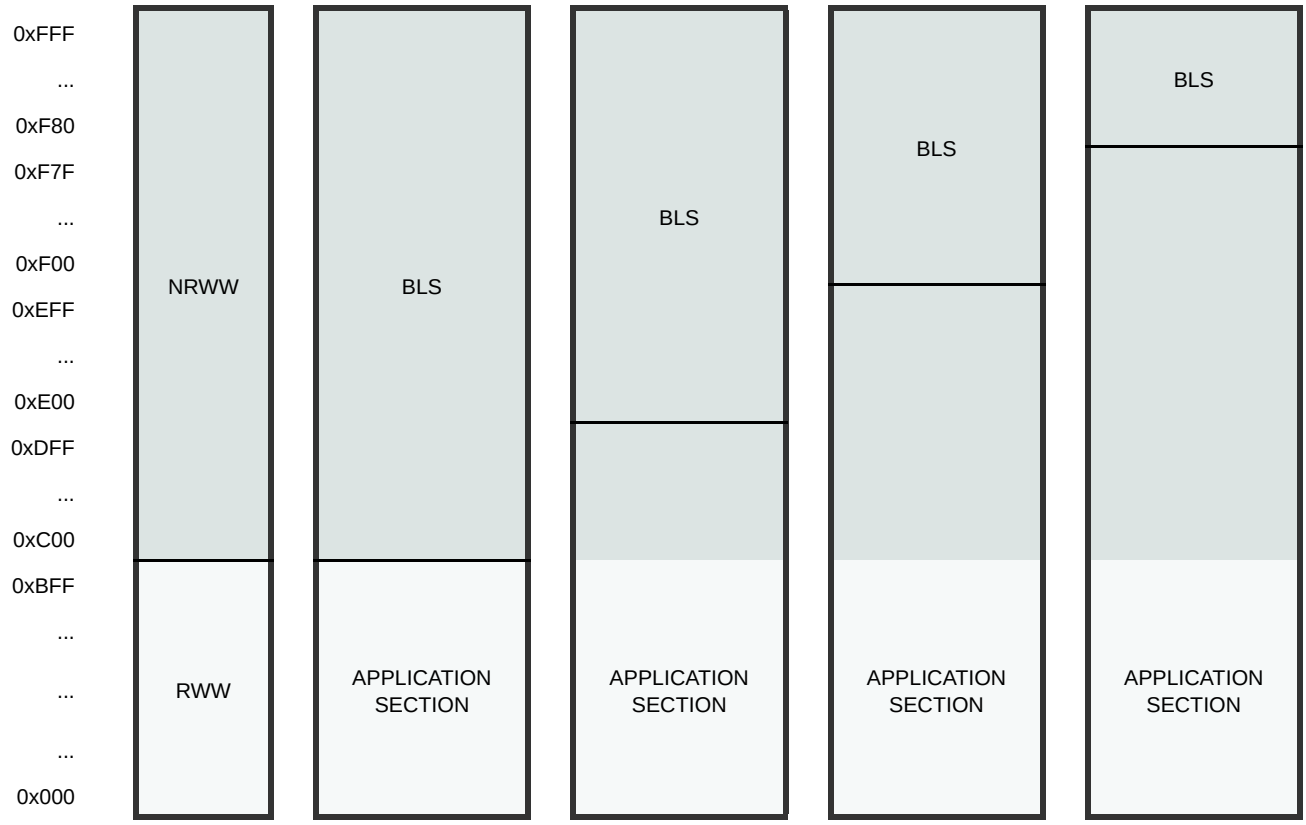
Table 82. Setting the Size of Application and Boot Loader Sections

BOOTSZ1	BOOTSZ0	Application Section		Boot Loader Section		
		Start	End	Start [1]	End	Size (words)
0	0	0x000	0xBFF	0xC00	0xFFF	1024
0	1		0xDFF	0xE00		512
1	0		0xEFF	0xF00		256
1	1		0xF7F	0xF80		128

Note: 1. Start of boot loader section = boot reset address.

Boot Loader Section options are illustrated in Figure 91, below.

Figure 91. Flash Memory Sections vs. BOOTSZ Fuse Bits.



21.7 Boot Loader Lock Bits

The boot loader has two separate sets of lock bits, which can be set independently. This gives the user a unique flexibility to select different levels of protection.

The user can choose any of the following:

- Protect the entire Flash from a software update by the MCU
- Protect only the Boot Loader Flash section from a software update by the MCU
- Protect only the Application Flash section from a software update by the MCU
- Allow software update in the entire Flash.

For more details on lock bits, see “Lock Bits” on page 225.

21.7.1 Programming Boot Loader Lock Bits by SPM

To set boot loader and general lock bits:

- Write the desired data to R0. A cleared bit indicates the corresponding lock bit is to be programmed. See bit mapping of R0 below
- Write “X0001001” to SPMCSR
- Execute SPM within four clock cycles after writing SPMCSR

During lock bit programming, the contents of R0 is treated as shown below.

Bit	7	6	5	4	3	2	1	0
R0	**1**	**1**	**BLB12**	**BLB11**	**BLB02**	**BLB01**	**LB2**	**LB1**

During the operation, the value of Z-pointer is ignored, but for future compatibility it is recommended to load the Z-pointer with 0x0001 (same as used for reading the lock bits). For future compatibility, it is also recommended to set bits 7 and 6 in R0 to “1” when writing the Lock bits.

When programming the Lock bits the entire Flash can be read during the operation.

See Table 87 on page 226 and Table 88 on page 226 for how different settings of the boot loader lock bits affect Flash access. See “Lock Bits” on page 225 for lock bit layout.

21.7.2 Updating the BLS

Special care must be taken if boot lock bit BLB11 is left unprogrammed, allowing the BLS to be updated. An accidental write to the BLS can corrupt the entire boot loader, making further software updates impossible. If boot loader software does not need to be updated, it is recommended to program boot lock bit BLB11 to protect the BLS from being changed by software.

21.8 Self-Programming the Flash

The device provides a self-programming mechanism for downloading and uploading program code by the MCU itself. Self-Programming can use any available data interface and associated protocol to read code and write (program) that code into program memory.

Program memory is updated in a page by page fashion. Before programming a page with the data stored in the temporary page buffer, the page must be erased. The temporary page buffer is filled one word at a time using SPM and the buffer can be filled either before the Page Erase command or between a Page Erase and a Page Write operation:

1. Either, fill the buffer before a Page Erase:
    1. Fill temporary page buffer
    2. Perform a Page Erase
    3. Perform a Page Write

2. Or, fill the buffer after Page Erase:
    1. Perform a Page Erase
    2. Fill temporary page buffer
    3. Perform a Page Write

When using alternative 1, the boot loader provides an effective read-modify-write feature, which allows the user software to first read the page, do the necessary changes, and then write back the modified data. If alternative 2 is used, it is not possible to read the old data while loading since the page is already erased.

If only a part of the page needs to be changed, the rest of the page must be stored (for example in the temporary page buffer) before the erase, and then be re-written.

The temporary page buffer can be accessed in a random sequence.

It is essential that the page address used in both the Page Erase and Page Write operation is addressing the same page.

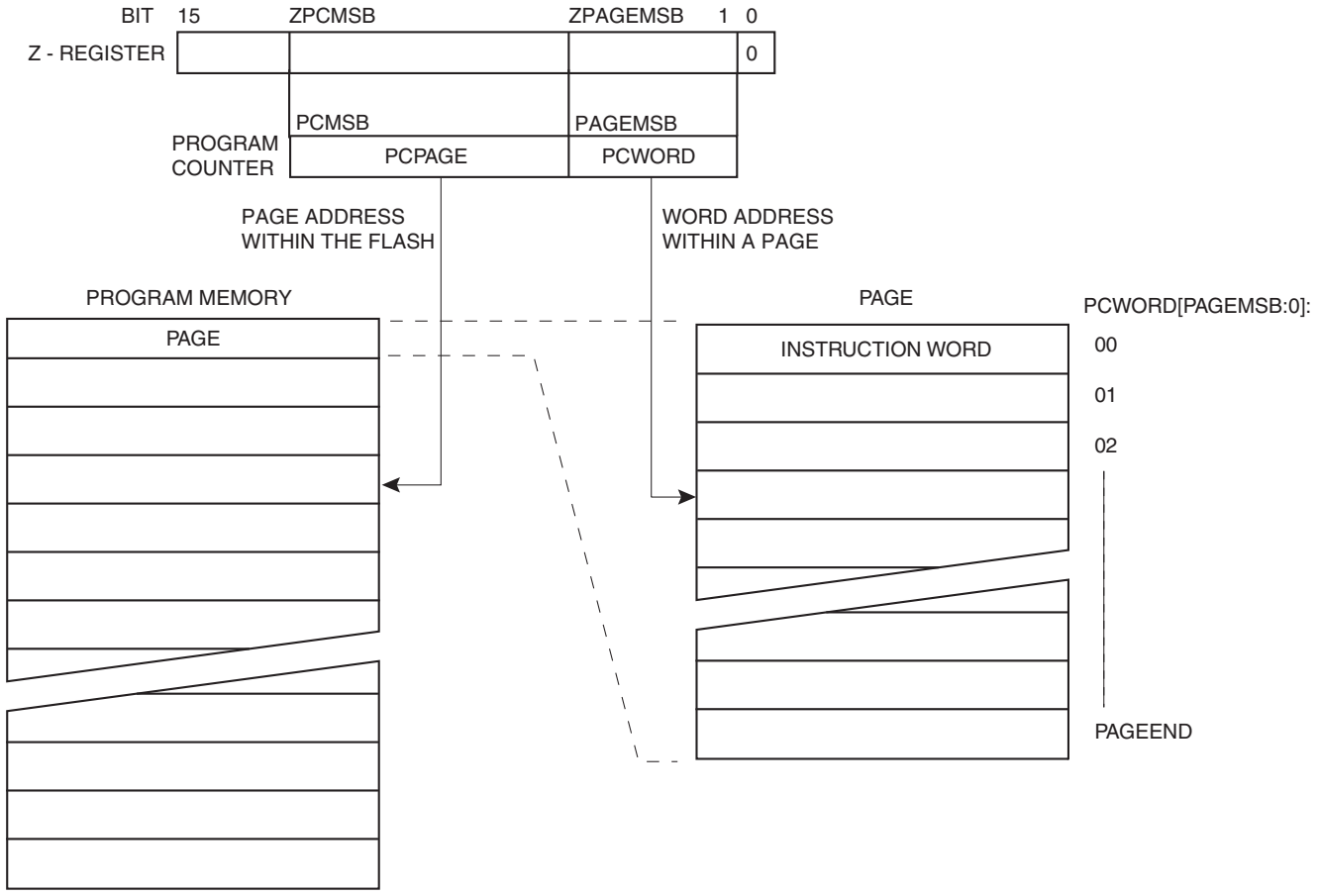
21.8.1 Addressing the Flash During Self-Programming

The Z-pointer is used to address the SPM commands.

Bit	15	14	13	12	11	10	9	8
ZH (R31)	**Z15**	**Z14**	**Z13**	**Z12**	**Z11**	**Z10**	**Z9**	**Z8**
ZL (R30)	**Z7**	**Z6**	**Z5**	**Z4**	**Z3**	**Z2**	**Z1**	**Z0**
	7	6	5	4	3	2	1	0

Since the Flash is organized in pages (see Table 94 on page 232), the Program Counter can be treated as having two different sections. One section, consisting of the least significant bits, is addressing the words within a page, while the most significant bits are addressing the pages. This is shown in Figure 92, below.

Figure 92. Addressing the Flash During SPM



Variables used in Figure 92 are explained in Table 83, below.

Table 83. Variables Used in Flash Addressing

Variable	Description
PCPAGE	Program Counter page address. Selects page of words and is used with Page Erase and Page Write operations. See Table 94 on page 232
PCMSB	The most significant bit of the Program Counter. See Table 94 on page 232
ZPCMSB	The bit in the Z register that is mapped to PCMSB. Because Z[0] is not used, ZPCMSB = PCMSB + 1. Z register bits above ZPCMSB are ignored
PCWORD	Program Counter word address. Selects the word within a page. This is used for filling the temporary buffer and must be zero during page write operations. See Table 94 on page 232
PAGEMSB	The most significant bit used to address the word within one page
ZPAGEMSB	The bit in the Z register that is mapped to PAGEMSB. Because Z[0] is not used, ZPAGEMSB = PAGEMSB + 1

Note that the Page Erase and Page Write operations are addressed independently. Therefore, it is very important that the boot loader addresses the same page in both Page Erase and Page Write operations.

Although the least significant bit of the Z-register (Z0) should be zero for SPM, it should be noted that the LPM instruction addresses the Flash byte-by-byte and uses Z0 as a byte select bit.

The only SPM operation that does not use the Z-pointer is setting the boot loader lock bits. The content of the Z-pointer is ignored and will have no effect on this operation.

Once a programming operation is initiated, the address is latched and the Z-pointer can be used for other operations.

21.8.2 Page Erase

To execute Page Erase:

- Set up the address in the Z-pointer
- Write “00000011” to SPMCSR
- Execute an SPM instruction within four clock cycles after writing SPMCSR

The data in R1 and R0 is ignored. The page address must be written to PCPAGE in the Z-register. Other bits in the Z-pointer are ignored during this operation.

If an interrupt occurs during the timed sequence above the four cycle access cannot be guaranteed. In order to ensure atomic operation interrupts should be disabled before writing to SPMCSR.

When performing a Page Erase of the RWW section, the NRWW section can be read during the operation. When performing a Page Erase of the NRWW section, the CPU is halted during the operation

The CPU is halted during the Page Erase operation.

21.8.3 Page Load

To write an instruction word:

- Set up the address in the Z-pointer
- Set up the data in R1:R0
- Write “00000001” to SPMCSR
- Execute an SPM instruction within four clock cycles after writing SPMCSR

The content of PCWORD in the Z-register is used to address the data in the temporary buffer. The temporary buffer will auto-erase after a Page Write operation, or by writing the RWWSRE bit in SPMCSR. It is also erased after a system reset.

Note that it is not possible to write more than one time to each address without erasing the temporary buffer.

If the EEPROM is written in the middle of an SPM Page Load operation, all data loaded will be lost.

21.8.4 Page Write

To execute Page Write:

- Set up the address in the Z-pointer
- Write “00000101” to SPMCSR
- Execute an SPM instruction within four clock cycles after writing SPMCSR

The data in R1 and R0 is ignored. The page address must be written to PCPAGE. Other bits in the Z-pointer must be written to zero during this operation.

When performing a Page Write of the RWW section, the NRWW section can be read during the operation. When performing a Page Write of the NRWW section, the CPU is halted during the operation.

21.8.5 SPM Interrupt

If the SPM interrupt is enabled, it will generate a constant interrupt when the SPMEN bit in SPMCSR is cleared. This means that the interrupt can be used instead of polling the SPMCSR.

When using the SPM interrupt, the interrupt vectors should be moved to the boot loader section to avoid that an interrupt is accessing the RWW section when blocked for reading. Moving the interrupts is described in section “Interrupts” on page 48.

21.8.6 SPMCSR Can Not Be Written When EEPROM is Being Programmed

Note that an EEPROM write operation will block all software programming to Flash. Reading fuses and lock bits from software will also be prevented during the EEPROM write operation. It is recommended that the user checks the status bit (EEPE) in EECR and verifies that it is cleared before writing to SPMCSR.

21.8.7 RWW Section Can Not Be Read During Self-Programming

During self-programming (Page Erase or Page Write), the RWW section is always blocked for reading and the software should therefore prevent this section from being addressed during the operation. The RWWSB in the SPMCSR is set as long as the RWW section is busy.

During self-programming the interrupt vector table should be moved to the BLS. See IVSEL bit of “MCUCR – MCU Control Register” on page 53. Alternatively, interrupts should be disabled.

After programming is complete, but before addressing the RWW section, the user software must clear RWWSB by writing the RWWSRE bit.

21.9 Preventing Flash Corruption

During periods of low V_{CC}, the Flash program can be corrupted because the supply voltage is too low for the CPU and the Flash to operate properly. These issues are the same as for board level systems using the Flash, and the same design solutions should be applied.

A Flash program corruption can be caused by two situations when the voltage is too low. First, a regular write sequence to the Flash requires a minimum voltage to operate correctly. Secondly, the CPU itself can execute instructions incorrectly, if the supply voltage for executing instructions is too low.

Flash corruption can easily be avoided by following these design recommendations (one is sufficient):

1. Keep the AVR RESET active (low) during periods of insufficient power supply voltage. This can be done by enabling the internal Brown-out Detector (BOD) if the operating voltage matches the detection level. If not, an external low V_{CC} reset protection circuit can be used. If a reset occurs while a write operation is in progress, the write operation will be completed provided that the power supply voltage is sufficient.
2. Keep the AVR core in Power-down sleep mode during periods of low V_{CC}. This will prevent the CPU from attempting to decode and execute instructions, effectively protecting the SPMCSR Register and thus the Flash from unintentional writes.
3. If there is no need for a boot loader, the boot loader lock bits should be programmed to prevent software updates

21.10 Programming Time for Flash when Using SPM

Flash access is timed using the internal, calibrated 8MHz oscillator. Typical Flash programming times for the CPU are shown in Table 84.

Table 84. SPM Programming Time

Operation	Min (1)	Max (1)
SPM: Flash Page Erase, Flash Page Write, and lock bit write	3.7 ms	4.5 ms

Note: 1. Min and max programming times are per individual operation.

21.11 Register Description

21.11.1 SPMCSR – Store Program Memory Control and Status Register

The Store Program Memory Control and Status Register contains the control bits needed to control the Program memory operations.

Bit	7	6	5	4	3	2	1	0	
0x37 (0x57)	SPMIE	RWWSB	RSIG	RWWSRE	RWFLB	PGWRT	PGERS	SPMEN	SPMCSR
Read/Write	R/W	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 – SPMIE: SPM Interrupt Enable**

When the SPMIE bit is written to one, and the I-bit in the Status Register is set (one), the SPM ready interrupt will be enabled. The SPM ready Interrupt will be executed as long as the SPMEN bit is cleared.

- **Bit 6 – RWWSB: Read-While-Write Section Busy**

When this bit is set, the RWW section cannot be accessed.

This bit is set when a self-programming operation (Page Erase or Page Write) to the RWW section is initiated.

This bit is cleared if the RWWSRE bit is written to one after a self-programming operation is completed. This bit is automatically cleared when a page load operation is initiated.

- **Bit 5 – RSIG: Read Device Signature Imprint Table**

Issuing an LPM instruction within three cycles after RSIG and SPMEN bits have been set in SPMCSR will return the selected data (depending on Z-pointer value) from the device signature imprint table into the destination register. See “Device Signature Imprint Table” on page 228 for details.

- **Bit 4 – RWWSRE: Read-While-Write Section Read Enable**

The RWW section is blocked for reading (see RWWSB bit) when the section is being programmed. To re-enable the section, the software must first wait until the programming is completed (see SPMEN bit). The RWW section is then re-enabled by simultaneously writing bits RWWSRE and SPMEN and, within four clock cycles, issuing an SPM instruction.

The RWW section cannot be re-enabled while the Flash is busy with a Page Erase or a Page Write operation (see SPMEN). If the RWWSRE bit is written while the Flash is being loaded, the operation will abort and the data will be lost.

- **Bit 3 – RWFLB: Read/Write Fuse and Lock Bits**

An LPM instruction within three cycles after RWFLB and SPMEN bits are set will read either the lock bits or fuse bits (depending on Z0 in the Z-pointer) into the destination register. See “Reading Lock, Fuse and Signature Data from Software” on page 229 for details.

If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles sets boot lock bits and memory lock bits, according to the data in R0. The data in R1 and the address in the Z-pointer are ignored. This bit is automatically cleared when lock bits have been set, or if no SPM instruction is executed within four clock cycles.

- **Bit 2 – PGWRT: Page Write**

If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles executes Page Write, with the data stored in the temporary buffer. The page address is taken from the high part of the Z-pointer. The data in R1 and R0 are ignored. The PGWRT bit will auto-clear upon completion of a Page Write, or if no SPM instruction is executed within four clock cycles. The CPU is halted during the entire Page Write operation.

- **Bit 1 – PGERS: Page Erase**

If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles executes Page Erase. The page address is taken from the high part of the Z-pointer. The data in R1 and R0 are ignored. The PGERS bit will auto-clear upon completion of a Page Erase, or if no SPM instruction is executed within four clock cycles. The CPU is halted during the entire Page Write operation.

- **Bit 0 – SPMEN: Store Program Memory Enable**

This bit enables the SPM instruction for the next four clock cycles. If set to one together with RSIG, CTPB, RWFLB, PGWRT or PGERS, the following LPM/SPM instruction will have a special meaning, as described elsewhere.

If only SPMEN is written, the following SPM instruction will store the value in R1:R0 in the temporary page buffer addressed by the Z-pointer. The LSB of the Z-pointer is ignored. The SPMEN bit will auto-clear upon completion of an SPM instruction, or if no SPM instruction is executed within four clock cycles. During Page Erase and Page Write, the SPMEN bit remains high until the operation is completed.

22. Lock Bits, Fuse Bits and Device Signature

22.1 Lock Bits

ATtiny828 provides the program and data memory lock bits listed in Table 85.

Table 85. Lock Bit Byte

Lock Bit Byte	Bit No	Description	See	Default Value (1)
–	7	–		1 (unprogrammed)
–	6	–		1 (unprogrammed)
BLB12	5	Boot lock bit	Page 226	1 (unprogrammed)
BLB11	4			1 (unprogrammed)
BLB02	3			1 (unprogrammed)
BLB01	2			1 (unprogrammed)
LB2	1	Lock bit	Below	1 (unprogrammed)
LB1	0			1 (unprogrammed)

Notes: 1. “1” means unprogrammed, “0” means programmed.

Lock bits can be left unprogrammed (“1”) or can be programmed (“0”) to obtain the additional features listed in Table 86, Table 87, and Table 88.

Table 86. Lock Bit Protection Modes

Lock Bits (1)		
LB2	LB1	Mode of Protection
1	1	No memory lock features enabled
1	0	Further programming of Flash and EEPROM is disabled in parallel and serial programming mode. Fuse bits are locked in both serial and parallel programming mode (2)
0	1	Reserved
0	0	Further reading and programming of Flash and EEPROM is disabled in parallel and serial programming mode. Fuse bits are locked in both serial and parallel programming mode (2)

Notes: 1. “1” means unprogrammed, “0” means programmed.
2. Program fuse bits before programming LB1 and LB2.

The general read/write lock bits do not control reading/writing by LPM/SPM. SPM and LPM access to application and boot loader sections is controlled by boot lock bits, listed in Table 87 and Table 88.

Table 87. Lock Bit Protection Modes (Application Section)

Lock Bits (1)		
BLB02	BLB01	Mode of Protection
1	1	No restrictions for SPM or LPM accessing the application section
1	0	SPM is not allowed to write to the application section
0	1	LPM executing from the boot loader section is not allowed to read from the application section. If interrupt vectors are placed in the boot loader section, interrupts are disabled while executing from the application section
0	0	SPM is not allowed to write to the application section, and LPM executing from the boot loader section is not allowed to read from the application section. If interrupt vectors are placed in the boot loader section, interrupts are disabled while executing from the application section

Notes: 1. “1” means unprogrammed, “0” means programmed.

Table 88. Lock Bit Protection Modes (Boot Loader Section)

Lock Bits (1)		
BLB12	BLB11	Mode of Protection
1	1	No restrictions for SPM or LPM accessing the boot loader section
1	0	SPM is not allowed to write to the boot loader section
0	1	LPM executing from the application section is not allowed to read from the boot loader section. If interrupt vectors are placed in the application section, interrupts are disabled while executing from the boot loader section
0	0	SPM is not allowed to write to the boot loader section, and LPM executing from the application section is not allowed to read from the boot loader section. If interrupt vectors are placed in the application section, interrupts are disabled while executing from the boot loader section

Notes: 1. “1” means unprogrammed, “0” means programmed.

Boot lock bits can be set by software or in serial or parallel programming mode, but they can be cleared by a Chip Erase command, only. See “Programming Boot Loader Lock Bits by SPM” on page 218.

22.2 Fuse Bits

Fuse bits are described in Table 89, Table 90, and Table 91. Note that programmed fuses read as zero.

Table 89. Extended Fuse Byte

Bit #	Bit Name	Use	See	Default Value
7	BODPD1	Sets BOD mode of operation when device is in sleep modes other than idle	Page 42	1 (unprogrammed)
6	BODPD0			1 (unprogrammed)
5	BODACT1	Sets BOD mode of operation when device is active or idle	Page 43	1 (unprogrammed)
4	BODACT0			1 (unprogrammed)

Bit #	Bit Name	Use	See	Default Value
3	–	–		1 (unprogrammed)
2	BOOTSZ1	Sets size of boot loader section	Page 216	1 (unprogrammed)
1	BOOTSZ0			1 (unprogrammed)
0	BOOTRST	Defines boot reset vector	Page 216	1 (unprogrammed)

Table 90. High Fuse Byte

Bit #	Bit Name	Use	See	Default Value
7	RSTDISBL	Disables external reset (1)	Page 79	1 (unprogrammed)
6	DWEN	Enables debugWIRE (1)	Page 212	1 (unprogrammed)
5	SPIEN	Enables serial programming and downloading of data to device (2)		0 (programmed) (3)
4	WDTON	Sets watchdog timer permanently on	Page 46	1 (unprogrammed)
3	EESAVE	Preserves EEPROM memory during Chip Erase operation	Page 235	1 (unprogrammed) (4)
2	BODLEVEL2	Sets BOD trigger level	Page 251	1 (unprogrammed)
1	BODLEVEL1			1 (unprogrammed)
0	BODLEVEL0			1 (unprogrammed)

Notes:
1. Programming this fuse bit will change the functionality of the $\overline{\text{RESET}}$ pin and render further programming via the serial interface impossible. The fuse bit can be unprogrammed using the parallel programming algorithm (see page 232).
2. This fuse bit is not accessible in serial programming mode.
3. This setting enables SPI programming.
4. This setting does not preserve EEPROM.

Table 91. Low Fuse Byte

Bit #	Bit Name	Use	See	Default Value
7	CKDIV8	Divides clock by 8 (1)	Page 30	0 (programmed)
6	CKOUT	Outputs system clock on port pin	Page 30	1 (unprogrammed)
5	SUT1	Sets system start-up time	Page 30	1 (unprogrammed) (2)
4	SUT0			0 (programmed) (2)
3	–	–		1 (unprogrammed)
2	–	–		1 (unprogrammed)
1	CKSEL1	Selects clock source	Page 28	1 (unprogrammed) (3)
0	CKSEL0			0 (programmed) (3)

Note:
1. Unprogramming this fuse at low voltages may result in overclocking. See Section 24.3 on page 249 for device speed versus supply voltage.
2. This setting results in maximum start-up time for the default clock source.

3. This setting selects Calibrated Internal 8MHz Oscillator.

Fuse bits are locked when Lock Bit 1 (LB1) is programmed. Hence, fuse bits must be programmed before lock bits.

Fuse bits are not affected by a Chip Erase.

22.2.1 Latching of Fuses

The fuse values are latched when the device enters programming mode and changes of the fuse values will have no effect until the part leaves Programming mode. This does not apply to the EESAVE fuse, which will take effect once it is programmed. The fuses are also latched on Power-up in Normal mode.

22.3 Device Signature Imprint Table

The device signature imprint table is a dedicated memory area used for storing miscellaneous device information, such as the device signature and oscillator calibaration data. Most of this memory segment is reserved for internal use, as outlined in Table 92.

Byte addresses are used when the device itself reads the data with the LPM command. External programming devices must use word addresses.

Table 92. Contents of Device Signature Imprint Table

Word Address (External)	Byte Address (Internal)	Description
0x00	0x00	Signature byte 0 (1)
	0x01	Calibration data for internal 8MHz oscillator (OSCCAL0) (2)
0x01	0x02	Signature byte 1 (1)
	0x03	Oscillator temperature calibration data (OSCTCAL0A)
0x02	0x04	Signature byte 2 (1)
	0x05	Oscillator temperature calibration data (OSCTCAL0B)
0x03	0x06	Reserved
	0x07	Calibration data for internal 32kHz oscillator (OSCCAL1) (2)
0x04 ...0x15	...	Reserved
	...	Reserved
0x16	0x2C	Calibration data for temperature sensor (gain) (3)(4)
	0x2D	Calibration data for temperature sensor (offset) (3)(5)
0x17...0x3F	...	Reserved
	...	Reserved

Notes:
1. For more information, see section “Signature Bytes” below.
2. For more information, see section “Calibration Bytes” below.
3. See “Temperature Measurement” on page 148. Calibration data is valid for ATtiny828R-devices, only.
4. Unsigned, fixed point, two’s complement: [0:(255/128)].
5. Signed integer, two’s complement: [-127:+128].

22.3.1 Signature Bytes

All Atmel microcontrollers have a three-byte signature code which identifies the device. This code can be read in both serial and parallel mode, also when the device is locked.

Signature bytes can also be read by the device firmware. See section “Reading Lock, Fuse and Signature Data from Software” on page 229.

The three signature bytes reside in a separate address space called the device signature imprint table. The signature data for ATtiny828 is given in Table 93.

Table 93. Device Signature Bytes

Part	Signature Byte 0	Signature Byte 1	Signature Byte 0
ATtiny828	0x1E	0x93	0x14

22.3.2 Calibration Bytes

The device signature imprint table of ATtiny828 contains calibration data for the internal oscillators, as shown in Table 92 on page 228. During reset, calibration data is automatically copied to the calibration registers (OSCCAL0, OSCCAL1) to ensure correct frequency of the calibrated oscillators. See “OSCCAL0 – Oscillator Calibration Register” on page 32, and “OSCCAL1 – Oscillator Calibration Register” on page 33.

Calibration bytes can also be read by the device firmware. See section “Reading Lock, Fuse and Signature Data from Software” on page 229.

22.4 Reading Lock, Fuse and Signature Data from Software

Fuse and lock bits can be read by device firmware. Programmed fuse and lock bits read zero. unprogrammed as one. See “Lock Bits” on page 225 and “Fuse Bits” on page 226.

In addition, firmware can also read data from the device signature imprint table. See “Device Signature Imprint Table” on page 228.

22.4.1 Lock Bit Read

Lock bit values are returned in the destination register after an LPM instruction has been issued within three CPU cycles after RWFLB and SPMEN bits have been set in SPMCSR (see page 223). The RWFLB and SPMEN bits automatically clear upon completion of reading the lock bits, or if no LPM instruction is executed within three CPU cycles, or if no SPM instruction is executed within four CPU cycles. When RWFLB and SPMEN are cleared LPM functions normally.

To read the lock bits, follow the below procedure:

1. Load the Z-pointer with 0x0001.
2. Set RWFLB and SPMEN bits in SPMCSR.
3. Issue an LPM instruction within three clock cycles.
4. Read the lock bits from the LPM destination register.

If successful, the contents of the destination register are as follows.

Bit	7	6	5	4	3	2	1	0
Rd	–	–	BLB12	BLB11	BLB02	BLB01	LB2	LB1

See section “Lock Bits” on page 225 for more information.

See “Boot Loader Lock Bits” on page 218 for how different settings of the boot loader lock bits affect Flash access.

22.4.2 Fuse Bit Read

The algorithm for reading fuse bytes is similar to the one described above for reading lock bits, only the addresses are different.

To read the Fuse Low Byte (FLB), follow the below procedure:

1. Load the Z-pointer with 0x0000.
2. Set RWFLB and SPMEN bits in SPMCSR.
3. Issue an LPM instruction within three clock cycles.
4. Read the FLB from the LPM destination register.

If successful, the contents of the destination register are as follows.

Bit	7	6	5	4	3	2	1	0
Rd	FLB7	FLB6	FLB5	FLB4	FLB3	FLB2	FLB1	FLB0

For a detailed description and mapping of the Fuse Low Byte, see Table 91 on page 227.

To read the Fuse High Byte (FHB), replace the address in the Z-pointer with 0x0003 and repeat the procedure above. If successful, the contents of the destination register are as follows.

Bit	7	6	5	4	3	2	1	0
Rd	FHB7	FHB6	FHB5	FHB4	FHB3	FHB2	FHB1	FHB0

For a detailed description and mapping of the Fuse High Byte, see Table 90 on page 227.

To read the Fuse Extended Byte (FEB), replace the address in the Z-pointer with 0x0002 and repeat the previous procedure. If successful, the contents of the destination register are as follows.

Bit	7	6	5	4	3	2	1	0
Rd	FEB7	FEB6	FEB5	FEB4	FEB3	FEB2	FEB1	FEB0

For a detailed description and mapping of the Fuse Extended Byte, see Table 89 on page 226.

22.4.3 Device Signature Imprint Table Read

To read the contents of the device signature imprint table, follow the below procedure:

1. Load the Z-pointer with the table index.
2. Set RSIG and SPMEN bits in SPMCSR.
3. Issue an LPM instruction within three clock cycles.
4. Read table data from the LPM destination register.

If successful, the contents of the destination register are as described in section “Device Signature Imprint Table” on page 228.

See program example below.

Assembly Code Example

```
DSIT_read:
      ; Uses Z-pointer as table index
      ldi               ZH, 0
      ldi               ZL, 1

      ; Preload SPMCSR bits into R16, then write to SPMCSR
      ldi               r16, (1<<RSIG)|(1<<SPMEN)
      out          SPMCSR, r16

      ; Issue LPM. Table data will be returned into r17
      lpm               r17, Z
      ret
```

Note: See “Code Examples” on page 7.

23. External Programming

This section describes how to program and verify Flash memory, EEPROM, lock bits, and fuse bits in ATtiny828.

23.1 Memory Parametrics

Flash memory parametrics are summarised in Table 94, below.

Table 94. Flash Parametrics

Device	Flash Size	Page Size	PCWORD (1)	Pages	PCPAGE (1)	PCMSB (1)
ATtiny828	4K words (8K bytes)	32 words	PC[4:0]	128	PC[11:5]	11

Note: 1. See Table 96 on page 233.

EEPROM parametrics are summarised in Table 95, below.

Table 95. EEPROM Parametrics

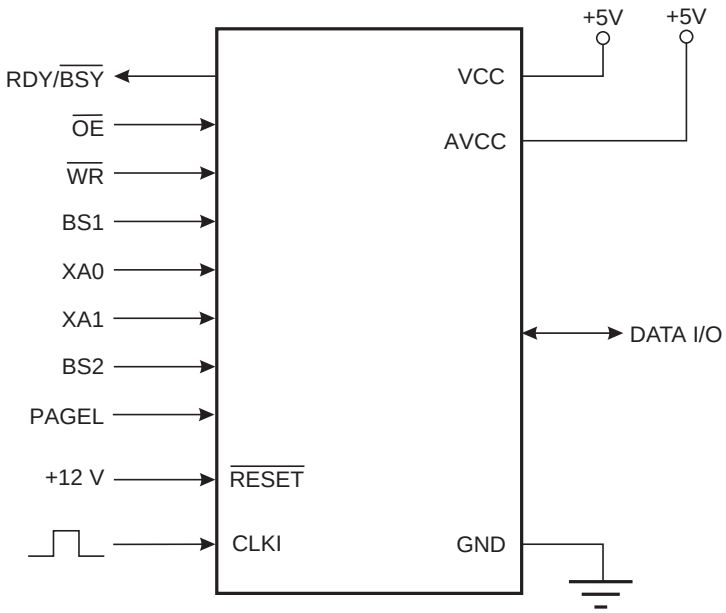
Device	EEPROM Size	Page Size	PCWORD (1)	Pages	PCPAGE (1)	EEAMSB
ATtiny828	256 bytes	4 bytes	EEA[1:0]	64	EEA[7:2]	7

Note: 1. See Table 96 on page 233.

23.2 Parallel Programming

Parallel programming signals and connections are illustrated in Figure 93, below.

Figure 93. Parallel Programming Signals



Signals are described in Table 96, below. Pins not listed in the table are referenced by pin names.

Table 96. Pin and Signal Names Used in Programming Mode

Signal Name	Pin(s)	I/O	Function
RDY/$\overline{\text{BSY}}$	PC0	O	0: Device is busy programming, 1: Device is ready for new command
$\overline{\text{OE}}$	PC1	I	Output enable (active low)
$\overline{\text{WR}}$	PC2	I	Write pulse (active low)
BS1	PC3	I	Byte select 1 (0: low byte, 1: high byte)
XA0	PA0	I	XTAL action bit 0
XA1	PA1	I	XTAL action bit 1
BS2	PB6	I	Byte Select 2 (0: low byte, 1: 2nd high byte)
PAGEL	PA2	I	Program memory and EEPROM data page load
DATA I/O	PB5 (MSB) PB4 PB0 PA7 PA6 PA5 PA4 PA3 (LSB)	I/O	Bi-directional data bus. Output when $\overline{\text{OE}}$ is low

Note: V_{CC} - 0.3V < AV_{CC} < V_{CC} + 0.3V, however, AV_{CC} should always be within 4.5 – 5.5.

Pulses are assumed to be at least 250 ns, unless otherwise noted.

Table 97. Pin Values Used to Enter Programming Mode

Pin	Symbol	Value
$\overline{\text{PAGEL}}$	Prog_enable[3]	0
XA1	Prog_enable[2]	0
XA0	Prog_enable[1]	0
BS1	Prog_enable[0]	0

Pins XA1 and XA0 determine the action when CLKI is given a positive pulse, as shown in Table 98.

Table 98. XA1 and XA0 Coding

XA1	XA0	Action when CLKI is Pulsed
0	0	Load Flash or EEPROM address (high or low address byte, determined by BS1)
0	1	Load data (high or low data byte for Flash, determined by BS1)
1	0	Load command
1	1	No action, idle

When pulsing $\overline{WR}$ or $\overline{OE}$, the command loaded determines the action executed. The different command options are shown in Table 99.

Table 99. Command Byte Bit Coding

Command Byte	Command
1000 0000	Chip Erase
0100 0000	Write fuse bits
0010 0000	Write lock bits
0001 0000	Write Flash
0001 0001	Write EEPROM
0000 1000	Read signature bytes and calibration byte
0000 0100	Read fuse and lock bits
0000 0010	Read Flash
0000 0011	Read EEPROM

23.2.1 Enter Programming Mode

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

1. Set Prog_enable pins (see Table 97 on page 233) to “0000”, $\overline{RESET}$ pin to 0V and V_{CC} to 0V.
2. Apply 4.5 – 5.5V between V_{CC} and GND. Ensure that V_{CC} reaches at least 1.7V within the next 20 µs.
3. Wait 20 – 60 µs, and apply 11.5 – 12.5V to $\overline{RESET}$.
4. Keep the Prog_enable pins unchanged for at least 10µs after the high voltage has been applied to ensure Prog_enable signature has been latched.
5. Wait at least 300 µs before giving any parallel programming commands.
6. Exit programming mode by powering the device down or by bringing $\overline{RESET}$ pin to 0V.

If the rise time of the V_{CC} is unable to fulfill the requirements listed above, the following alternative algorithm can be used:

1. Set Prog_enable pins (Table 97 on page 233) to “0000”, $\overline{RESET}$ pin to 0V and V_{CC} to 0V.
2. Apply 4.5 – 5.5V between V_{CC} and GND.
3. Monitor V_{CC}, and as soon as V_{CC} reaches 0.9 – 1.1V, apply 11.5 – 12.5V to $\overline{RESET}$.
4. Keep the Prog_enable pins unchanged for at least 10µs after the high voltage has been applied to ensure Prog_enable signature has been latched.

5. Wait until V_{CC} actually reaches 4.5 – 5.5V before giving any parallel programming commands.
6. Exit programming mode by powering the device down or by bringing $\overline{\text{RESET}}$ pin to 0V.

23.2.2 Considerations for Efficient Programming

Loaded commands and addresses are retained in the device during programming. For efficient programming, the following should be considered.

- When writing or reading multiple memory locations, the command needs only be loaded once
- Do not write the data value 0xFF, since this already is the contents of the entire Flash and EEPROM (unless the EESAVE Fuse is programmed) after a Chip Erase
- Address high byte needs only be loaded before programming or reading a new 256 word window in Flash or 256 byte EEPROM. This also applies to reading signature bytes

23.2.3 Chip Erase

A Chip Erase must be performed before the Flash and/or EEPROM are reprogrammed. The Chip Erase command will erase all Flash and EEPROM plus lock bits. If the EESAVE fuse is programmed, the EEPROM is not erased.

Lock bits are not reset until the program memory has been completely erased. Fuse bits are not changed.

The Chip Erase command is loaded as follows:

1. Set XA1, XA0 to “10”. This enables command loading
2. Set BS1 to “0”
3. Set DATA to “1000 0000”. This is the command for Chip Erase
4. Give CLKI a positive pulse. This loads the command
5. Give $\overline{\text{WR}}$ a negative pulse. This starts the Chip Erase. RDY/$\overline{\text{BSY}}$ goes low
6. Wait until RDY/$\overline{\text{BSY}}$ goes high before loading a new command

23.2.4 Programming the Flash

Flash is organized in pages, as shown in Table 94 on page 232. When programming the Flash, the program data is first latched into a page buffer. This allows one page of program data to be programmed simultaneously. The following procedure describes how to program the entire Flash memory:

A. Load Command “Write Flash”

1. Set XA1, XA0 to “10”. This enables command loading.
2. Set BS1 to “0”.
3. Set DATA to “0001 0000”. This is the command for Write Flash.
4. Give CLKI a positive pulse. This loads the command.

B. Load Address Low byte

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

C. Load Data Low Byte

1. Set XA1, XA0 to “01”. This enables data loading.
2. Set DATA = Data low byte (0x00 – 0xFF).

3. Give CLKI a positive pulse. This loads the data byte.

D. Load Data High Byte

1. Set BS1 to “1”. This selects high data byte.
2. Set XA1, XA0 to “01”. This enables data loading.
3. Set DATA = Data high byte (0x00 – 0xFF).
4. Give CLKI a positive pulse. This loads the data byte.

E. Latch Data

1. Set BS1 to “1”. This selects high data byte.
2. Give PAGEL a positive pulse. This latches the data bytes. (See Figure 95 for signal waveforms)

F. Repeat B through E until the entire buffer is filled or until all data within the page is loaded.

While the lower bits in the address are mapped to words within the page, the higher bits address the pages within the FLASH. This is illustrated in Figure 94 on page 237. Note that if less than eight bits are required to address words in the page (pagesize < 256), the most significant bit(s) in the address low byte are used to address the page when performing a Page Write.

G. Load Address High byte

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

H. Program Page

1. Give $\overline{WR}$ a negative pulse. This starts programming of the entire page of data. RDY/$\overline{BSY}$ goes low.
2. Wait until RDY/$\overline{BSY}$ goes high (See Figure 95 for signal waveforms).

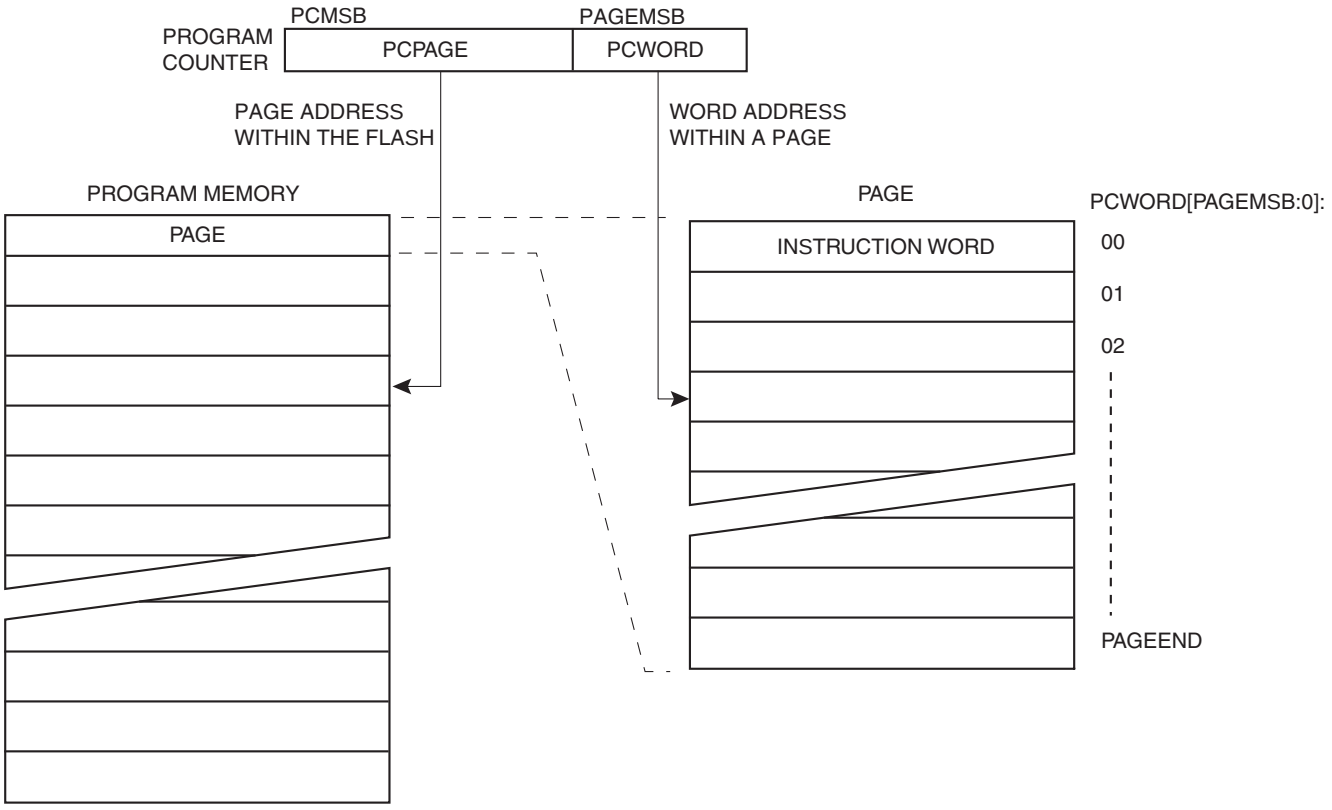
I. Repeat B through H until the entire Flash is programmed or until all data has been programmed.

J. End Page Programming

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

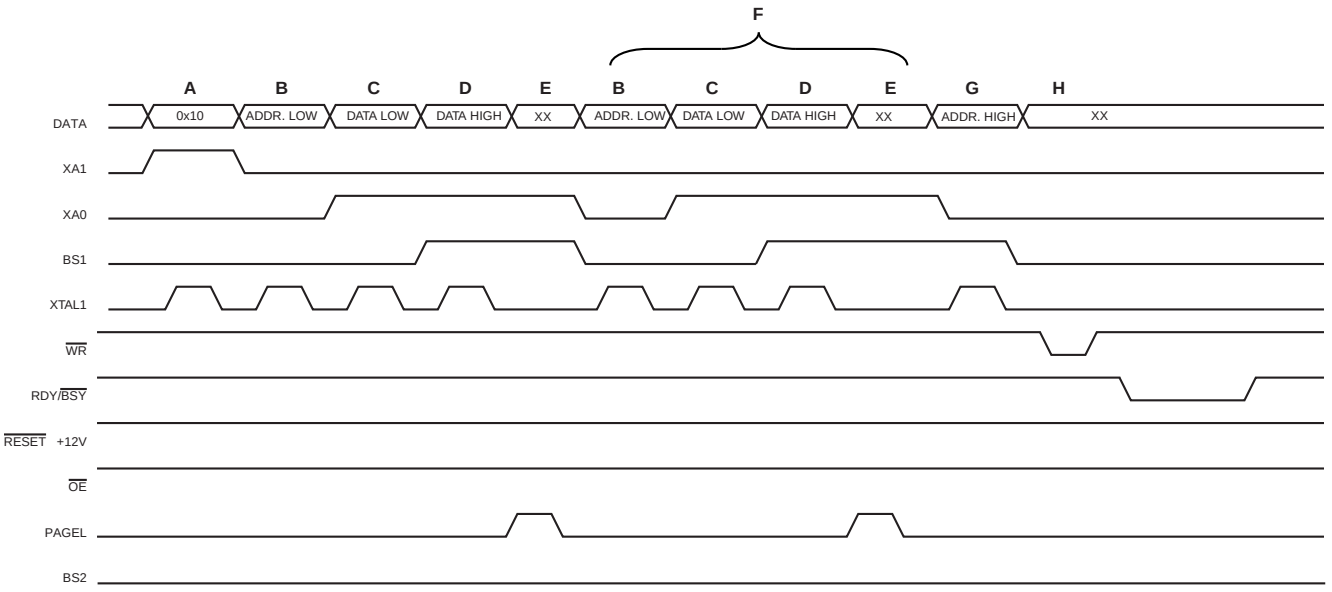
Flash page addressing is illustrated in Figure 94, below. Symbols used are described in Table 96 on page 233.

Figure 94. Addressing the Flash Which is Organized in Pages



Flash programming waveforms are illustrated in Figure 95, where XX means “don’t care” and letters refer to the programming steps described earlier.

Figure 95. Flash Programming Waveforms



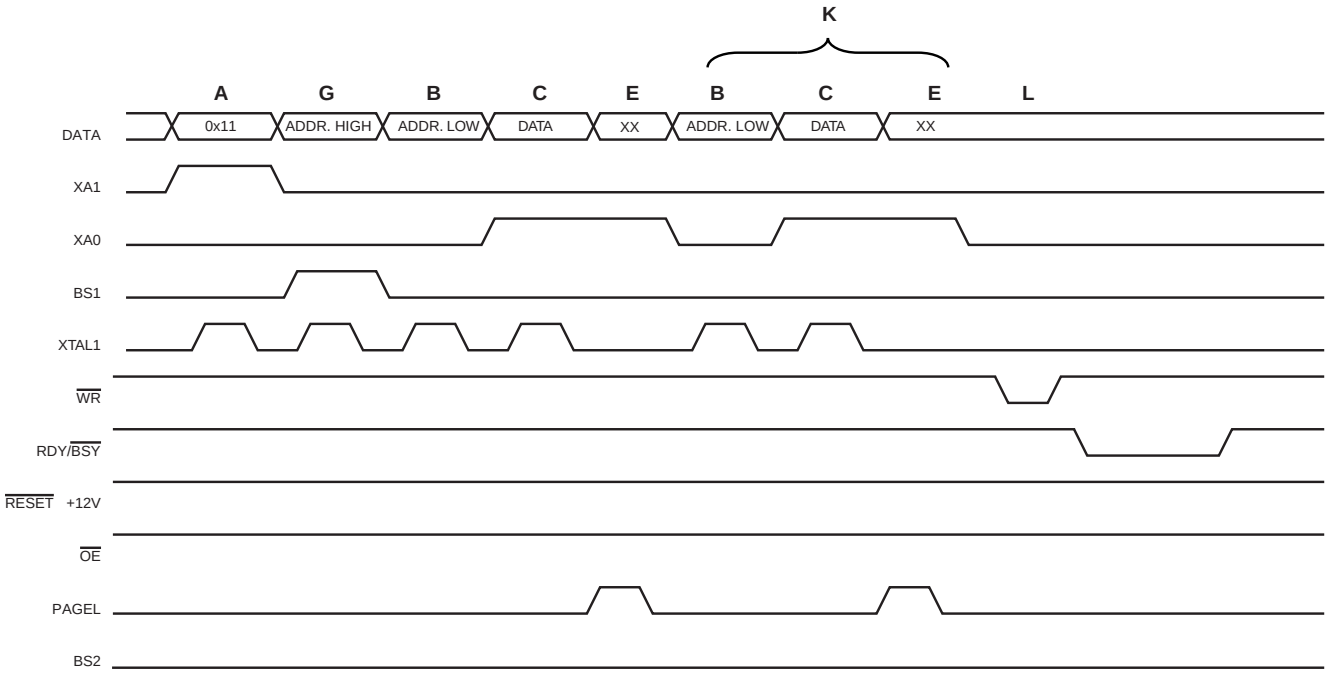
23.2.5 Programming the EEPROM

The EEPROM is organized in pages, see Table 95 on page 232. When programming the EEPROM, the program data is latched into a page buffer. This allows one page of data to be programmed simultaneously. The programming algorithm for the EEPROM data memory is as follows (see "Programming the Flash" on page 235 for details on loading command, address and data):

- A: Load command "0001 0001"
- G: Load address high byte (0x00 – 0xFF)
- B: Load address low byte (0x00 – 0xFF)
- C: Load data (0x00 – 0xFF)
- E: Latch data (give PAGEL a positive pulse)
- K: Repeat steps B, C, and E until the entire buffer is filled
- L: Program EEPROM page:
  - Set BS1 to "0"
  - Give $\overline{WR}$ a negative pulse. This starts programming of the EEPROM page. RDY/$\overline{BSY}$ goes low
  - Wait until to RDY/$\overline{BSY}$ goes high before programming the next page (See Figure 96 for signal waveforms)

EEPROM programming waveforms are illustrated in Figure 96, where XX means "don't care" and letters refer to the programming steps described above.

Figure 96. EEPROM Programming Waveforms



23.2.6 Reading the Flash

The algorithm for reading the Flash memory is as follows (see "Programming the Flash" on page 235 for details on command and address loading):

- A: Load command "0000 0010"
- G: Load address high byte (0x00 – 0xFF)
- B: Load address low byte (0x00 – 0xFF)
- Set $\overline{OE}$ to "0", and BS1 to "0". The Flash word low byte can now be read at DATA

- Set BS1 to “1”. The Flash word high byte can now be read at DATA
- Set $\overline{OE}$ to “1”

23.2.7 Reading the EEPROM

The algorithm for reading the EEPROM memory is as follows (see “Programming the Flash” on page 235 for details on command and address loading):

- A: Load command “0000 0011”
- G: Load address high byte (0x00 – 0xFF)
- B: Load address low byte (0x00 – 0xFF)
- Set $\overline{OE}$ to “0”, and BS1 to “0”. The EEPROM Data byte can now be read at DATA
- Set $\overline{OE}$ to “1”

23.2.8 Programming Low Fuse Bits

The algorithm for programming the low fuse bits is as follows (see “Programming the Flash” on page 235 for details on command and data loading):

- A: Load command “0100 0000”
- C: Load data low byte. Bit n = “0” programs and bit n = “1” erases the fuse bit
- Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high

23.2.9 Programming High Fuse Bits

The algorithm for programming the high fuse bits is as follows (see “Programming the Flash” on page 235 for details on command and data loading):

- A: Load command “0100 0000”
- C: Load data low byte. Bit n = “0” programs and bit n = “1” erases the fuse bit
- Set BS1 to “1” and BS2 to “0”. This selects high data byte
- Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high
- Set BS1 to “0”. This selects low data byte

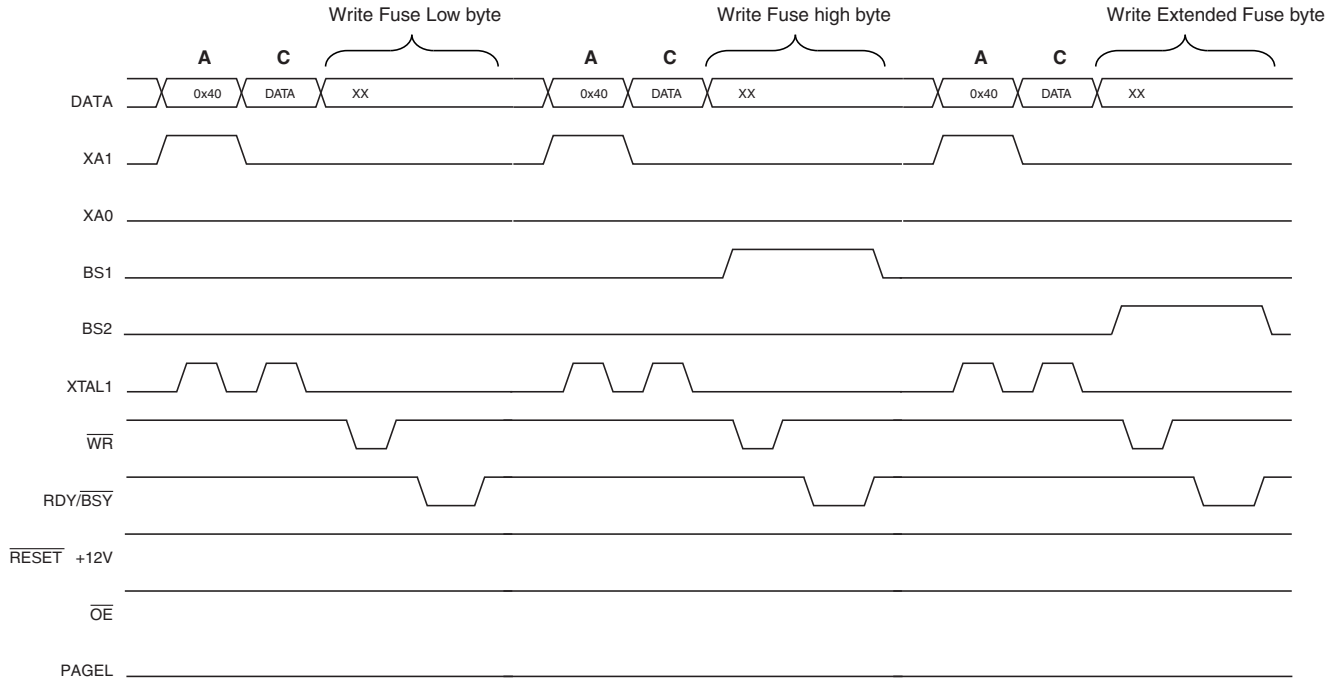
23.2.10 Programming Extended Fuse Bits

The algorithm for programming the extended fuse bits is as follows (see “Programming the Flash” on page 235 for details on command and data loading):

- A: Load command “0100 0000”
- C: Load data low byte. Bit n = “0” programs and bit n = “1” erases the fuse bit
- Set BS1 to “0” and BS2 to “1”. This selects extended data byte
- Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high
- Set BS2 to “0”. This selects low data byte

EEPROM programming waveforms are illustrated in Figure 96, where XX means “don’t care” and letters refer to the programming steps described above.

Figure 97. Fuses Programming Waveforms



23.2.11 Programming the Lock Bits

The algorithm for programming the lock bits is as follows (see “Programming the Flash” on page 235 for details on command and data loading):

- A: Load command “0010 0000”
- C: Load data low byte. Bit n = “0” programs the Lock bit. If LB1 and LB2 have been programmed, it is not possible to program the Lock Bits by any External Programming mode
- Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high

Lock bits can only be cleared by executing Chip Erase.

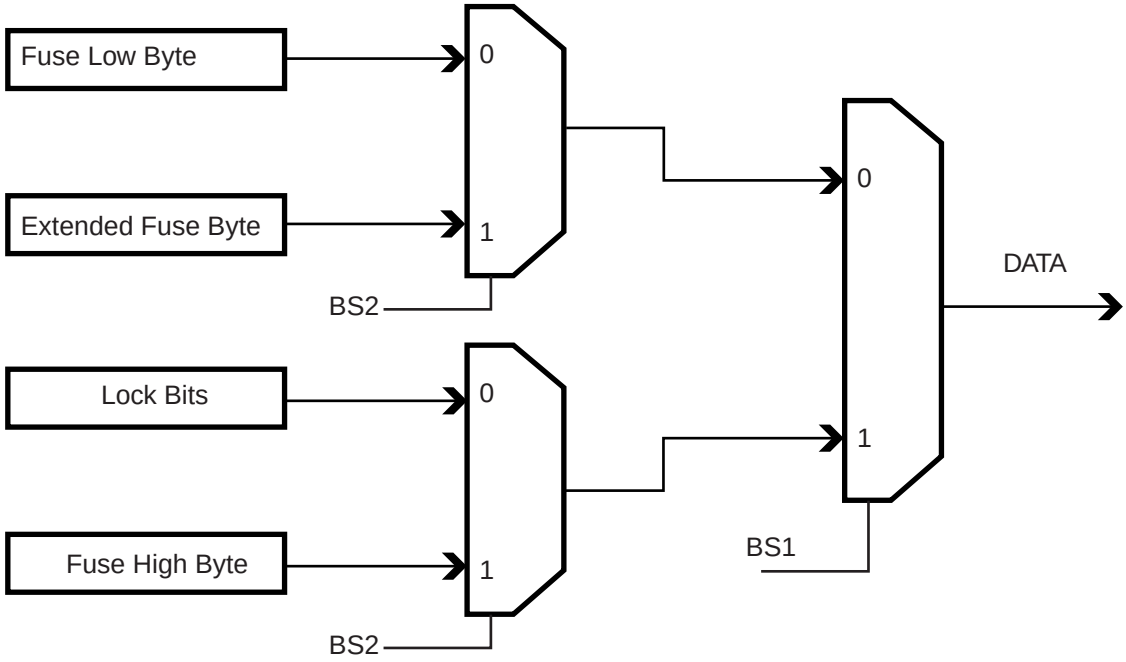
23.2.12 Reading Fuse and Lock Bits

The algorithm for reading fuse and lock bits is as follows (see “Programming the Flash” on page 235 for details on command loading):

- A: Load command “0000 0100”
- Set $\overline{OE}$ to “0”, BS2 to “0” and BS1 to “0”. Low fuse bits can now be read at DATA (“0” means programmed)
- Set $\overline{OE}$ to “0”, BS2 to “1” and BS1 to “1”. High fuse bits can now be read at DATA (“0” means programmed)
- Set $\overline{OE}$ to “0”, BS2 to “1”, and BS1 to “0”. Extended fuse bits can now be read at DATA (“0” means programmed)
- Set $\overline{OE}$ to “0”, BS2 to “0” and BS1 to “1”. Lock bits can now be read at DATA (“0” means programmed)
- Set $\overline{OE}$ to “1”

Fuse and lock bit mapping is illustrated in Figure 98, below.

Figure 98. Mapping Between BS1, BS2 and the Fuse and Lock Bits During Read



23.2.13 Reading Signature Bytes

The algorithm for reading the signature bytes is as follows (see “Programming the Flash” on page 235 for details on command and address loading):

1. A: Load command “0000 1000”
2. B: Load address low byte (0x00 – 0x02)
3. Set $\overline{OE}$ to “0”, and BS1 to “0”. The selected signature byte can now be read at DATA.
4. Set $\overline{OE}$ to “1”.

23.2.14 Reading the Calibration Byte

The algorithm for reading the calibration byte is as follows (see “Programming the Flash” on page 235 for details on command and address loading):

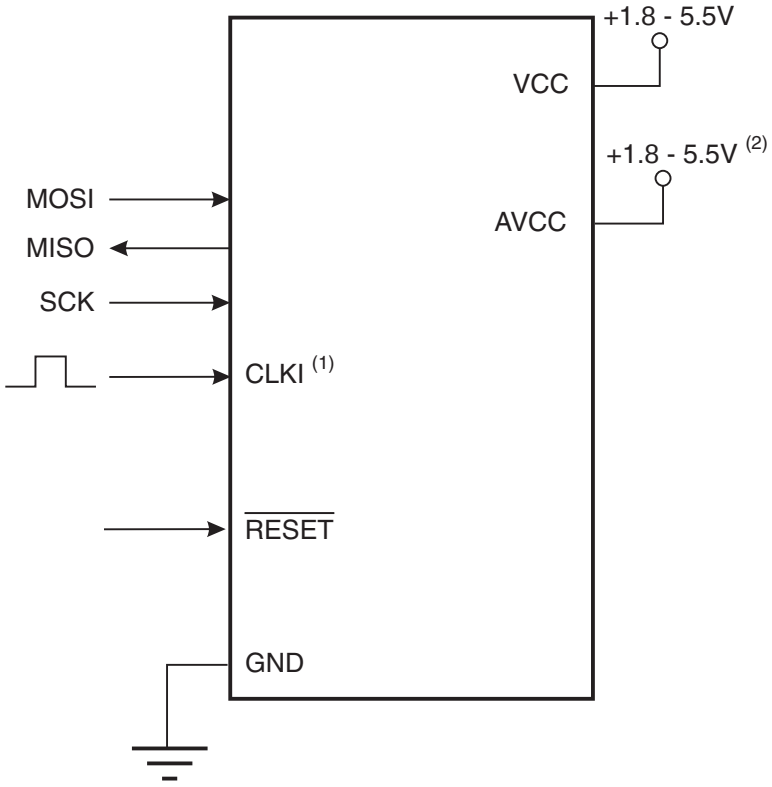
1. A: Load command “0000 1000”.
2. B: Load address low byte, 0x00.
3. Set $\overline{OE}$ to “0”, and BS1 to “1”. The calibration byte can now be read at DATA.
4. Set $\overline{OE}$ to “1”.

23.3 Serial Programming

Flash and EEPROM memory arrays can both be programmed using the serial SPI bus while $\overline{RESET}$ is pulled to GND. The serial interface consists of pins SCK, MOSI (input) and MISO (output). After $\overline{RESET}$ is set low, the Programming Enable instruction needs to be executed before program/erase operations can be executed.

Serial programming signals and connections are illustrated in Figure 99, below. The pin mapping is listed in Table 100 on page 243.

Figure 99. Serial Programming Signals



Notes: 1. If the device is clocked by the internal oscillator there is no need to connect a clock source to the CLKI pin.
2. V_{CC} - 0.3V < AV_{CC} < V_{CC} + 0.3V, however, AV_{CC} should always be within 1.7 – 5.5V.

When programming the EEPROM, an auto-erase cycle is built into the self-timed programming operation and there is no need to first execute the Chip Erase instruction. This applies for serial programming mode, only.

The Chip Erase operation turns the content of every memory location in Flash and EEPROM arrays into 0xFF.

Depending on CKSEL fuses, a valid clock must be present. The minimum low and high periods for the serial clock (SCK) input are defined as follows:

- Minimum low period of serial clock:
  - When f_{ck} < 12MHz: > 2 CPU clock cycles
  - When f_{ck} >= 12MHz: 3 CPU clock cycles
- Minimum high period of serial clock:
  - When f_{ck} < 12MHz: > 2 CPU clock cycles
  - When f_{ck} >= 12MHz: 3 CPU clock cycles

23.3.1 Pin Mapping

The pin mapping is listed in Table 100 on page 243. Note that not all parts use the SPI pins dedicated for the internal SPI interface.

Table 100. Pin Mapping Serial Programming

Symbol	Pins	I/O	Description
MOSI	PD0	I	Serial Data in
MISO	PD1	O	Serial Data out
SCK	PD3	I	Serial Clock

23.3.2 Programming Algorithm

When writing serial data to the ATtiny828, data is clocked on the rising edge of SCK. When reading data from the ATtiny828, data is clocked on the falling edge of SCK. See Figure 107 on page 257 and Figure 108 on page 257 for timing details.

To program and verify the ATtiny828 in the serial programming mode, the following sequence is recommended (See Table 101 on page 244):

1. Power-up sequence: apply power between V_{CC} and GND while $\overline{\text{RESET}}$ and SCK are set to “0”
   - In some systems, the programmer can not guarantee that SCK is held low during power-up. In this case, $\overline{\text{RESET}}$ must be given a positive pulse after SCK has been set to '0'. The duration of the pulse must be at least t_{RST} plus two CPU clock cycles. See Table 107 on page 250 for definition of minimum pulse width on $\overline{\text{RESET}}$ pin, t_{RST}
2. Wait for at least 20 ms and then enable serial programming by sending the Programming Enable serial instruction to the MOSI pin
3. The serial programming instructions will not work if the communication is out of synchronization. When in sync, the second byte (0x53) will echo back when issuing the third byte of the Programming Enable instruction
   - Regardless if the echo is correct or not, all four bytes of the instruction must be transmitted
   - If the 0x53 did not echo back, give $\overline{\text{RESET}}$ a positive pulse and issue a new Programming Enable command
4. The Flash is programmed one page at a time. The memory page is loaded one byte at a time by supplying the 6 LSB of the address and data together with the Load Program Memory Page instruction
   - To ensure correct loading of the page, data low byte must be loaded before data high byte for a given address is applied
   - The Program Memory Page is stored by loading the Write Program Memory Page instruction with the 7 MSB of the address
   - If polling (RDY/$\overline{\text{BSY}}$) is not used, the user must wait at least t_{WD_FLASH} before issuing the next page (See Table 102 on page 246). Accessing the serial programming interface before the Flash write operation completes can result in incorrect programming.
5. The EEPROM can be programmed one byte or one page at a time.
   - **A**: Byte programming. The EEPROM array is programmed one byte at a time by supplying the address and data together with the Write instruction. EEPROM memory locations are automatically erased before new data is written. If polling (RDY/$\overline{\text{BSY}}$) is not used, the user must wait at least t_{WD_EEPROM} before issuing the next byte (See Table 102 on page 246). In a chip erased device, no 0xFFs in the data file(s) need to be programmed
   - **B**: Page programming (the EEPROM array is programmed one page at a time). The memory page is loaded one byte at a time by supplying the 6 LSB of the address and data together with the Load EEPROM Memory Page instruction. The EEPROM memory page is stored by loading the Write EEPROM Memory Page Instruction with the 7 MSB of the address. When using EEPROM page access only byte locations loaded with the Load EEPROM Memory Page instruction are altered and the remaining locations remain unchanged. If polling (RDY/$\overline{\text{BSY}}$) is not used, the user must wait at least t_{WD_EEPROM} before issuing the next byte (See Table 102 on page 246). In a chip erased device, no 0xFF in the data file(s) need to be programmed

6. Any memory location can be verified by using the Read instruction, which returns the content at the selected address at the serial output pin (MISO)
7. At the end of the programming session, $\overline{\text{RESET}}$ can be set high to commence normal operation
8. Power-off sequence (if required): set $\overline{\text{RESET}}$ to “1”, and turn V_{CC} power off

23.3.3 Programming Instruction set

The instruction set for serial programming is described in Table 101 and Figure 100 on page 245.

Table 101. Serial Programming Instruction Set

	Instruction Format			
Instruction/Operation	**Byte 1**	**Byte 2**	**Byte 3**	**Byte4**
Programming Enable	$AC	$53	$00	$00
Chip Erase (Program Memory/EEPROM)	$AC	$80	$00	$00
Poll RDY/$\overline{\text{BSY}}$	$F0	$00	$00	data byte out
Load Instructions				
Load Extended Address byte (1)	$4D	$00	Extended adr	$00
Load Program Memory Page, High byte	$48	$00	adr LSB	high data byte in
Load Program Memory Page, Low byte	$40	$00	adr LSB	low data byte in
Load EEPROM Memory Page (page access)	$C1	$00	0000 000aa (2)	data byte in
Read Instructions				
Read Program Memory, High byte	$28	adr MSB	adr LSB	high data byte out
Read Program Memory, Low byte	$20	adr MSB	adr LSB	low data byte out
Read EEPROM Memory	$A0	0000 00aa (2)	aaaa aaaa (2)	data byte out
Read Lock bits	$58	$00	$00	data byte out
Read Signature Byte	$30	$00	0000 000aa (2)	data byte out
Read Fuse bits	$50	$00	$00	data byte out
Read Fuse High bits	$58	$08	$00	data byte out
Read Fuse Extended Bits	$50	$08	$00	data byte out
Read Calibration Byte	$38	$00	$00	data byte out
Write Instructions (3)				
Write Program Memory Page	$4C	adr MSB (4)	adr LSB (4)	$00
Write EEPROM Memory	$C0	0000 00aa (2)	aaaa aaaa (2)	data byte in
Write EEPROM Memory Page (page access)	$C2	0000 00aa (2)	aaaa aa00 (2)	$00
Write Lock bits (5)	$AC	$E0	$00	data byte in

Instruction/Operation	Instruction Format			
	Byte 1	Byte 2	Byte 3	Byte4
Write Fuse bits [5]	$AC	$A0	$00	data byte in
Write Fuse High bits [5]	$AC	$A8	$00	data byte in
Write Fuse Extended Bits [5]	$AC	$A4	$00	data byte in

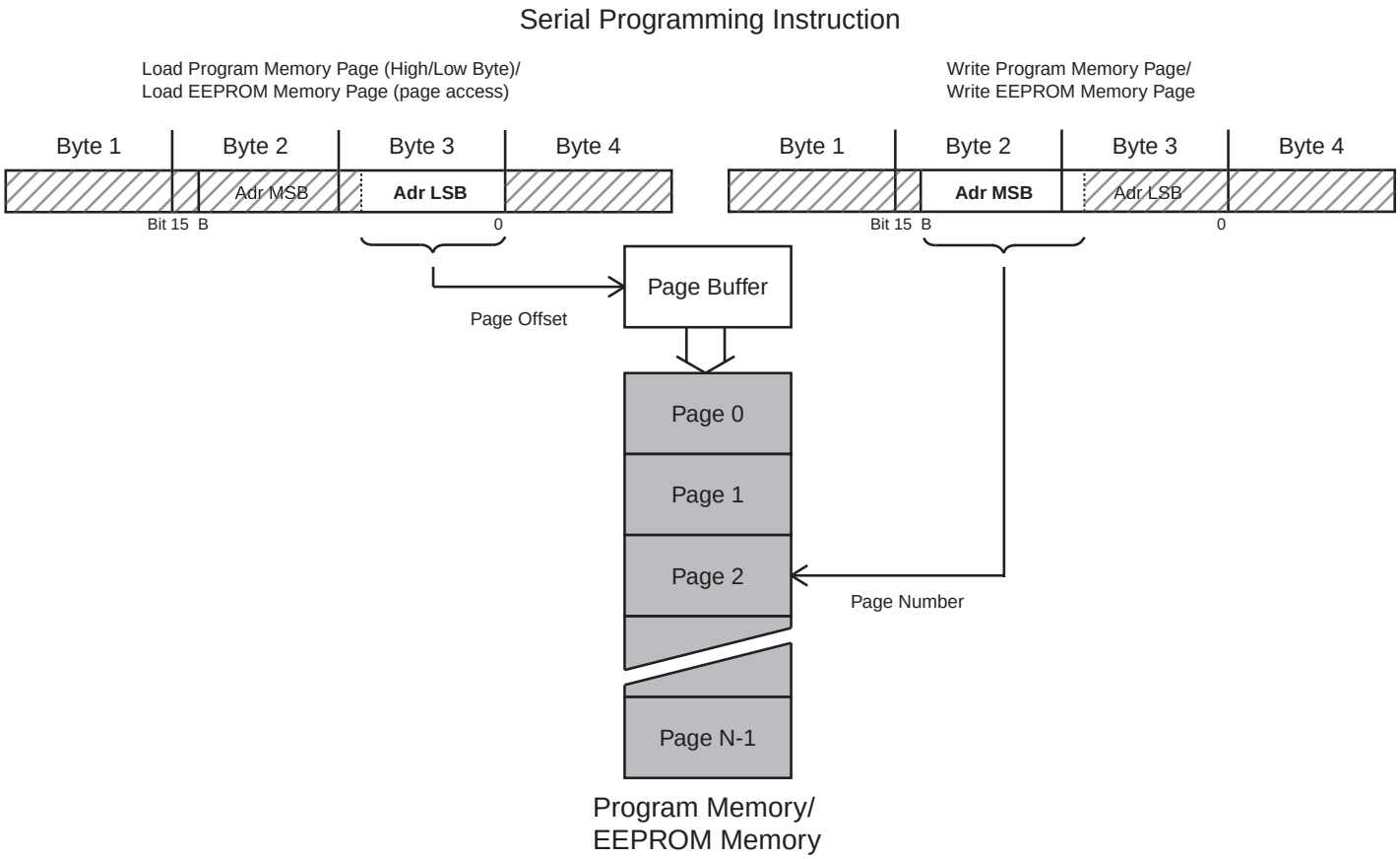
Notes:
1. Not all instructions are applicable for all parts.
2. a = address.
3. Instructions accessing program memory use a word address. This address may be random within the page range.
4. Word addressing.
5. To ensure future compatibility, unused fuses and lock bits should be unprogrammed (‘1’) .

If the LSB of RDY/$\overline{\text{BSY}}$ data byte out is ‘1’, a programming operation is still pending. Wait until this bit returns ‘0’ before the next instruction is carried out.

Within the same page, the low data byte must be loaded prior to the high data byte.

After data is loaded to the page buffer, program the EEPROM page, see Figure 100 on page 245.

Figure 100. Serial Programming Instruction example



23.4 Programming Time for Flash and EEPROM

Flash and EEPROM wait times are listed in Table 102 on page 246.

Table 102. Typical Wait Delays Before Next Flash or EEPROM Location Can Be Written

Symbol	Minimum Wait Delay
t_{WD_FLASH}	4.5 ms
t_{WD_EEPROM}	3.6 ms
t_{WD_ERASE}	9.0 ms

24. Electrical Characteristics

24.1 Absolute Maximum Ratings*

Operating Temperature . . . . . .-55°C to +125°C

Storage Temperature . . . . . .-65°C to +150°C

Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground. . . . . -0.5V to V_{CC}+0.5V

Voltage on $\overline{\text{RESET}}$ with respect to Ground-0.5V to +13.0V

Maximum Operating Voltage 6.0V

DC Current per I/O Pin. . . . . 40.0 mA

DC Current V_{CC} and GND Pins . . . . . 200.0 mA

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

24.2 DC Characteristics

Table 103. DC Characteristics. T = -40°C to +85°C

Symbol	Parameter	Condition	Min	Typ (1)	Max	Units
V_{IL}	Input Low Voltage	V_{CC} = 1.7V – 2.4V V_{CC} = 2.4V – 5.5V	-0.5		$0.2V_{CC}$ (3) $0.3V_{CC}$ (3)	V
V_{IH}	Input High-voltage Except $\overline{\text{RESET}}$ pin	V_{CC} = 1.7V – 2.4V V_{CC} = 2.4V – 5.5V	$0.7V_{CC}$ (2) $0.6V_{CC}$ (2)		V_{CC} +0.5	V
	Input High-voltage $\overline{\text{RESET}}$ pin	V_{CC} = 1.7V to 5.5V	$0.9V_{CC}$ (2)		V_{CC} +0.5	V
V_{OL}	Output Low Voltage (4) $\overline{\text{RESET}}$ pin as I/O (6)	V_{CC} = 5V, I_{OL} = 2 mA (5)			0.6	V
		V_{CC} = 3V, I_{OL} = 1 mA (5)			0.5	
		V_{CC} = 1.8V, I_{OL} = 0.4mA (5)			0.4	
	Output Low Voltage (4) Standard Sink I/O Pin (7)	V_{CC} = 5V, I_{OL} = 10 mA (5)			0.6	
		V_{CC} = 3V, I_{OL} = 5 mA (5)			0.5	
		V_{CC} = 1.8V, I_{OL} = 2mA (5)			0.4	
	Output Low Voltage (4) High Sink I/O Pin (8)	V_{CC} = 5V, I_{OL} = 20 mA (5)			0.6	
		V_{CC} = 3V, I_{OL} = 10 mA (5)			0.5	
		V_{CC} = 1.8V, I_{OL} = 4mA (5)			0.4	
	Output Low Voltage (4) Extra High Sink I/O Pin (9)	V_{CC} = 5V, I_{OL} = 20 mA (5)			0.6	
		V_{CC} = 3V, I_{OL} = 20 mA (5)			0.6	
		V_{CC} = 1.8V, I_{OL} = 8mA (5)			0.5	

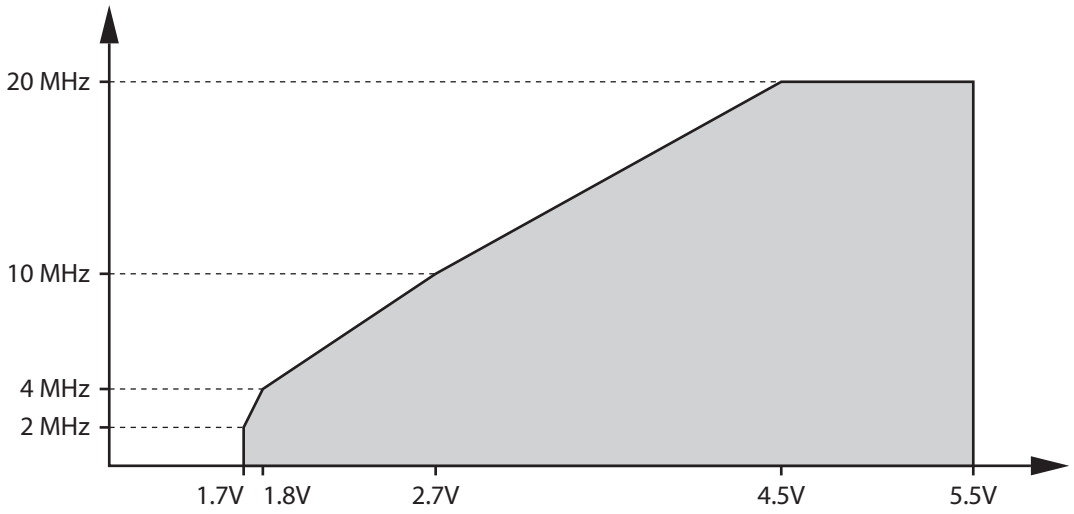
Symbol	Parameter	Condition	Min	Typ [1]	Max	Units
V_{OH}	Output High-voltage[4] Except RESET pin[6]	V_{CC} = 5V, I_{OH} = -10 mA [5]	4.3			V
		V_{CC} = 3V, I_{OH} = -5 mA [5]	2.5			
		V_{CC} = 1.8V, I_{OH} = -2 mA [5]	1.4			
I_{LIL}	Input Leakage Current, I/O Pin (absolute value)	V_{CC} = 5.5V, pin low		<0.05	1	µA
I_{LIH}	Input Leakage Current, I/O Pin (absolute value)	V_{CC} = 5.5V, pin high		<0.05	1	µA
I_{LIAC}	Input Leakage Current, Analog Comparator	V_{CC} = 5V V_{IN} = V_{CC}/2	-50		50	nA
R_{RST}	Reset Pull-up Resistor	V_{CC} = 5.5V, input low	30		60	kΩ
R_{PU}	I/O Pin Pull-up Resistor	V_{CC} = 5.5V, input low	20		50	kΩ
I_{CC}	Power Supply Current[10]	Active 1 MHz, V_{CC} = 2V		0.2	0.4	mA
		Active 4 MHz, V_{CC} = 3V		1.2	2	mA
		Active 8 MHz, V_{CC} = 5V		3.9	5	mA
		Idle 1 MHz, V_{CC} = 2V		0.03	0.1	mA
		Idle 4 MHz, V_{CC} = 3V		0.2	0.4	mA
		Idle 8 MHz, V_{CC} = 5V		0.9	1.5	mA
	Power-down mode[11]	WDT enabled, V_{CC} = 3V		1.8	4	µA
		WDT disabled, V_{CC} = 3V		0.1	2	µA

Notes:
1. Typical values at 25°C.
2. “Min” means the lowest value where the pin is guaranteed to be read as high.
3. “Max” means the highest value where the pin is guaranteed to be read as low.
4. Under steady-state (non-transient) conditions I/O ports can sink/source more current than the test conditions, however, the sum current of PORTA and PORTB mustn’t exceed 100mA. Also, the sum current of PORTC and PORTD mustn’t exceed 120mA. V_{OL}/V_{OH} is not guaranteed to meet specifications if pin or port currents exceed the limits given.
5. Pins are not guaranteed to sink/source currents greater than those listed at the given supply voltage.
6. The RESET pin must tolerate high voltages when entering and operating in programming modes and, as a consequence, has a weak drive strength as compared to regular I/O pins. See “Reset Pin as I/O” on page 279, and “Reset Pin as I/O” on page 285.
7. Ports with standard sink strength: PORTD0, PORTD3.
8. Ports with high sink strength: PORTA[7:0], PORTB[7:0], PORTC[7:0], PORTD1.
9. Ports with extra high strength: PORTC[7:0]. See “PHDE – Port High Drive Enable Register” on page 81.
10. Results obtained using external clock and methods described in “Minimizing Power Consumption” on page 35. Power reduction fully enabled (PRR = 0xFF) and with no I/O drive.
11. BOD Disabled.

24.3 Speed

The maximum operating frequency of the device is dependent on supply voltage, V_{CC}. The relationship between supply voltage and maximum operating frequency is piecewise linear, as shown in Figure 101.

Figure 101. Maximum Operating Frequency vs. Supply Voltage



24.4 Clock Characteristics

24.4.1 Accuracy of Calibrated Internal Oscillator

It is possible to manually calibrate the internal oscillator to be more accurate than default factory calibration. Note that the oscillator frequency depends on temperature and voltage. Voltage and temperature characteristics can be found in "Internal Oscillator Speed" on page 293.

Table 104. Calibration Accuracy of Internal 8MHz Oscillator

Calibration Method	Target Frequency	V_{CC}	Temperature	Accuracy (1)
Factory Calibration	8.0 MHz	3V	25°C	±2% (2) ±10% (2)
User Calibration (3)	Within: 7.3 – 8.1 MHz	Within: 1.7V – 5.5V	Within: -40°C to +85°C	±1%

Notes:
1. Accuracy of oscillator frequency at calibration point (fixed temperature and voltage).
2. See device ordering codes on page 303 for alternatives.
3. Not available in ATtiny828R devices.

24.4.2 Accuracy of Calibrated 32kHz Oscillator

It is possible to manually calibrate the internal 32kHz oscillator to be more accurate than default factory calibration. Note that the oscillator frequency depends on temperature and voltage. Voltage and temperature characteristics can be found in "Internal Oscillator Speed" on page 293.

Table 105. Calibration Accuracy of Internal 32kHz Oscillator

Calibration Method	Target Frequency	V_{CC}	Temperature	Accuracy
Factory Calibration	32kHz	1.7 – 5.5V	-40°C to +85°C	±30%

24.4.3 External Clock Drive

Figure 102. External Clock Drive Waveform

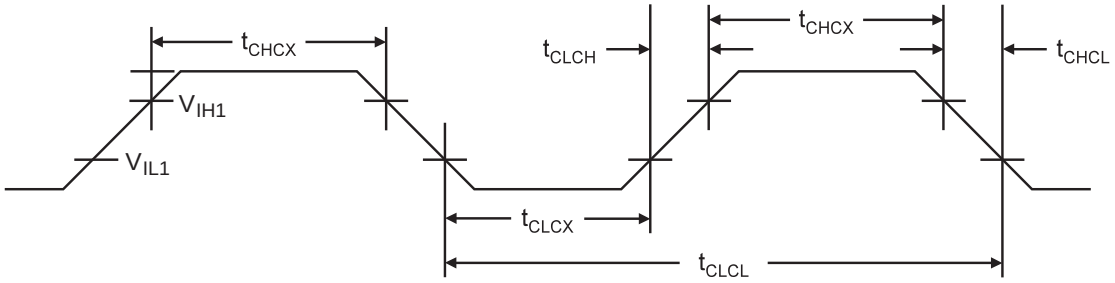


Table 106. External Clock Drive Characteristics

Symbol	Parameter	V_{CC} = 1.7 – 5.5V		V_{CC} = 2.7–5.5V		V_{CC} = 4.5–5.5V		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	
$1/t_{CLCL}$	Clock Frequency	0	4	0	8	0	12	MHz
t_{CLCL}	Clock Period	250		125		83		ns
t_{CHCX}	High Time	100		40		20		ns
t_{CLCX}	Low Time	100		40		20		ns
t_{CLCH}	Rise Time		2.0		1.6		0.5	µs
t_{CHCL}	Fall Time		2.0		1.6		0.5	µs
Δt_{CLCL}	Period change from one clock cycle to next		2		2		2	%

24.5 System and Reset Characteristics

Table 107. Reset and Internal Voltage Characteristics

Symbol	Parameter	Condition	Min(1)	Typ(1)	Max(1)	Units
V_{RST}	$\overline{\text{RESET}}$ Pin Threshold Voltage		0.2 V_{CC}		0.9V_{CC}	V
V_{BG}	Internal bandgap voltage	V_{CC} = 2.7V T_A = 25°C	1.0	1.1	1.2	V
t_{RST}	Minimum pulse width on $\overline{\text{RESET}}$ Pin	V_{CC} = 1.8V V_{CC} = 3V V_{CC} = 5V		2000 700 400		ns

Note: 1. Values are guidelines, only

24.5.1 Power-On Reset

Table 108. Characteristics of Enhanced Power-On Reset. T_A = -40 to +85°C

Symbol	Parameter	Min[1]	Typ[1]	Max[1]	Units
V_{POR}	Release threshold of power-on reset [2]	1.1	1.4	1.6	V
V_{POA}	Activation threshold of power-on reset [3]	0.6	1.3	1.6	V
SR_{ON}	Power-On Slope Rate	0.01			V/ms

Note:
1. Values are guidelines, only
2. Threshold where device is released from reset when voltage is rising
3. The Power-on Reset will not work unless the supply voltage has been below V_{POT} (falling)

24.5.2 Brown-Out Detection

Table 109. V_{BOT} vs. BODLEVEL Fuse Coding

BODLEVEL[2:0] Fuses	Min[1]	Typ[1]	Max[1]	Units
11X	1.7	1.8	2.0	V
101	2.5	2.7	2.9	
100	4.1	4.3	4.5	
0XX	Reserved			

Note:
1. V_{BOT} may be below nominal minimum operating voltage for some devices. For devices where this is the case, the device is tested down to $V_{CC} = V_{BOT}$ during the production test. This guarantees that a Brown-out Reset will occur before V_{CC} drops to a voltage where correct operation of the microcontroller is no longer guaranteed.

24.6 Temperature Sensor

Table 110. Accuracy of Temperature Sensor at Factory Calibration

Symbol	Parameter	Condition	Min	Typ	Max	Units
A_{TS}	Accuracy	V_{CC} = 4.0, T_A = 25°C – 85°C		10		°C

Note:
1. Firmware calculates temperature based on factory calibration value.
2. Min and max values are not guaranteed. Contact your local Atmel sales office if higher accuracy is required.

24.7 Two-Wire Serial Interface Characteristics

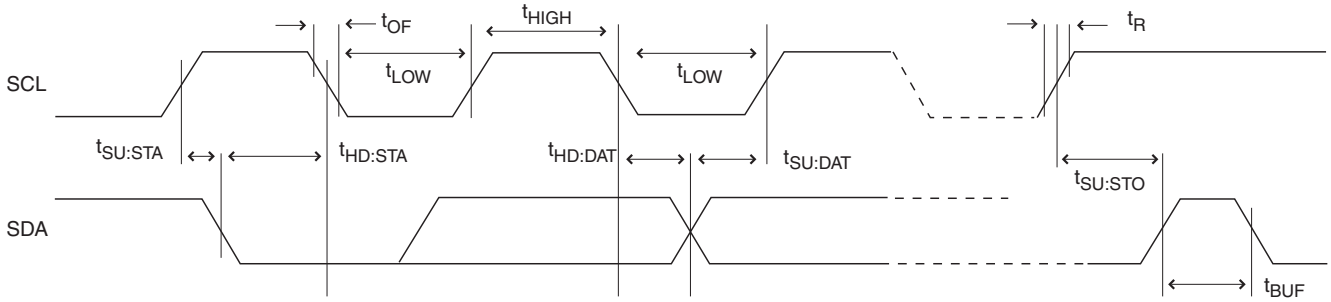
The following data is based on simulations and characterisations. Parameters listed in Table 111 on page 252 are not tested in production. Symbols refer to Figure 103.

Table 111. Two-Wire Serial Interface Characteristics

Symbol	Parameter	Condition	Min	Max	Unit
V_{IL}	Input Low voltage		-0.5	0.3 V_{CC}	V
V_{IH}	Input High voltage		0.7 V_{CC}	V_{CC} + 0.5	V
V_{HYS}	Hysteresis of Schmitt-trigger inputs	V_{CC} > 2.7V	0.05 V_{CC}	–	V
		V_{CC} < 2.7V	0		
V_{OL}	Output Low voltage	I_{OL} = 3mA, V_{CC} > 2.7V	0	0.4	V
		I_{OL} = 2mA, V_{CC} < 2.7V			
t_{SP}	Spikes suppressed by input filter		0	50	ns
f_{SCL}	SCL clock frequency (1)		0	400	kHz
$t_{HD:STA}$	Hold time (repeated) START Condition		0.6	–	µs
t_{LOW}	Low period of SCL clock		1.3	–	µs
t_{HIGH}	High period of SCL clock		0.6	–	µs
$t_{SU:STA}$	Set-up time for repeated START condition		0.6	–	µs
$t_{HD:DAT}$	Data hold time		0	0.9	µs
$t_{SU:DAT}$	Data setup time		100	–	ns
$t_{SU:STO}$	Setup time for STOP condition		0.6	–	µs
t_{BUF}	Bus free time between STOP and START condition		1.3	–	µs

Notes: 1. f_{CK} = CPU clock frequency.

Figure 103. Two-Wire Serial Bus Timing



24.8 ADC Characteristics

Table 112. ADC Characteristics. T = -40°C to +85°C. V_{CC} = 1.7 – 5.5V

Symbol	Parameter	Condition	Min	Typ	Max	Units
	Resolution				10	Bits
	Absolute accuracy (Including INL, DNL, and Quantization, Gain and Offset Errors)	V_{REF} = V_{CC} = 4V, ADC clock = 200 kHz		2		LSB
		V_{REF} = V_{CC} = 4V, ADC clock = 1 MHz		3		LSB
		V_{REF} = V_{CC} = 4V, ADC clock = 200 kHz Noise Reduction Mode		1.5		LSB
		V_{REF} = V_{CC} = 4V, ADC clock = 1 MHz Noise Reduction Mode		2.5		LSB
	Integral Non-Linearity (INL) (Accuracy after Offset and Gain Calibration)	V_{REF} = V_{CC} = 4V, ADC clock = 200 kHz		1		LSB
	Differential Non-linearity (DNL)	V_{REF} = V_{CC} = 4V, ADC clock = 200 kHz		0.5		LSB
	Gain Error	V_{REF} = V_{CC} = 4V, ADC clock = 200 kHz		2.5		LSB
	Offset Error	V_{REF} = V_{CC} = 4V, ADC clock = 200 kHz		1.5		LSB
	Conversion Time	Free Running Conversion	13		260	µs
	Clock Frequency		50		1000	kHz
V_{IN}	Input Voltage		GND		V_{REF}	V
	Input Bandwidth			38.5		kHz
R_{AIN}	Analog Input Resistance			100		MΩ
	ADC Conversion Output		0		1023	LSB

24.9 Analog Comparator Characteristics

Table 113. Analog Comparator Characteristics, T = -40°C to +85°C

Symbol	Parameter	Condition	Min	Typ	Max	Units
V_{AIO}	Input Offset Voltage	V_{CC} = 5V, VIN = V_{CC} / 2		< 10	40	mV
I_{LAC}	Input Leakage Current	V_{CC} = 5V, VIN = V_{CC} / 2	-50		50	nA
t_{APD}	Analog Propagation Delay (from saturation to slight overdrive)	V_{CC} = 2.7V		750		ns
		V_{CC} = 4.0V		500		
	Analog Propagation Delay (large step change)	V_{CC} = 2.7V		100		
		V_{CC} = 4.0V		75		
t_{DPD}	Digital Propagation Delay	V_{CC} = 1.7V – 5.5		1	2	CLK

24.10 Parallel Programming Characteristics

Figure 104. Parallel Programming Timing, Including some General Timing Requirements

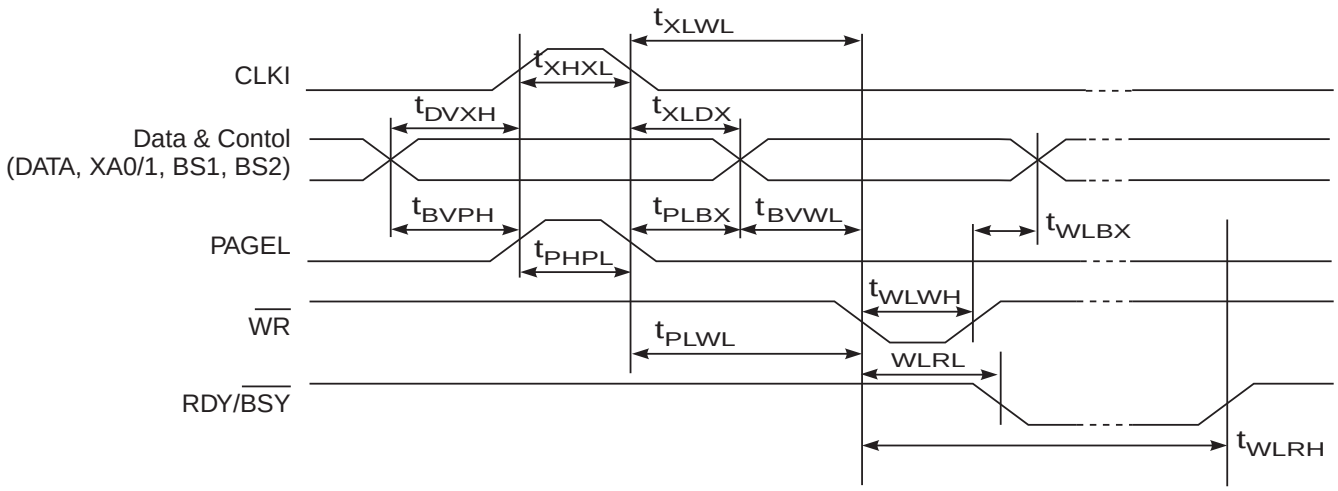
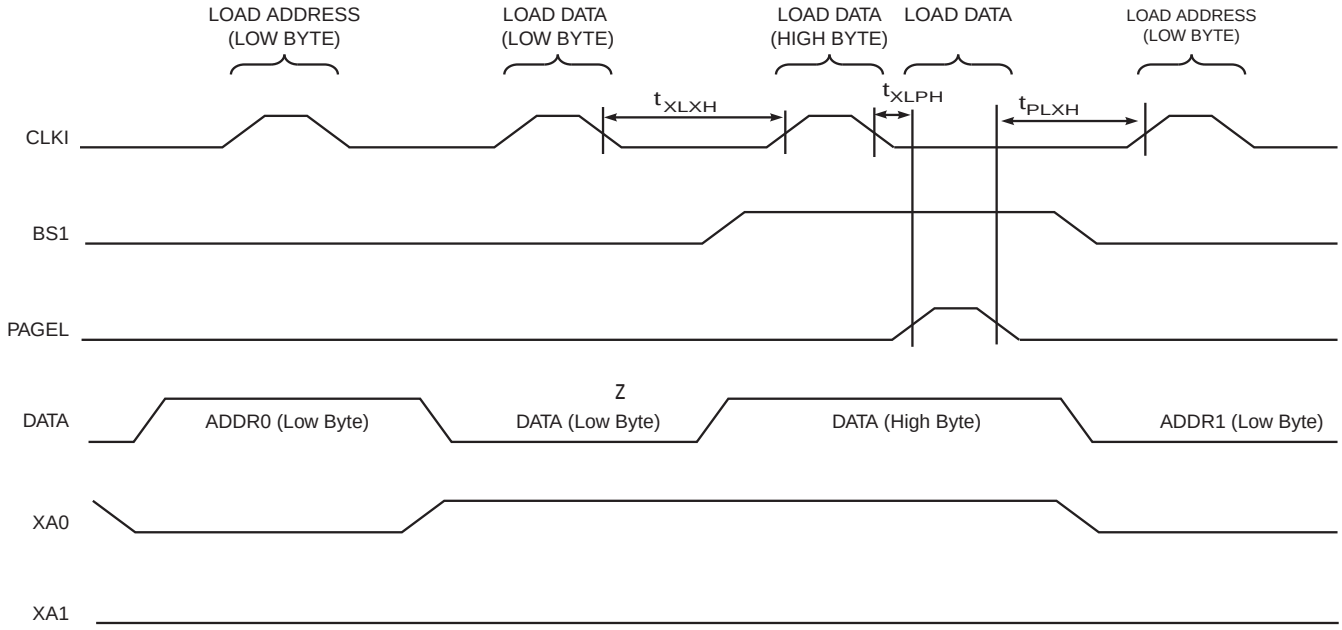
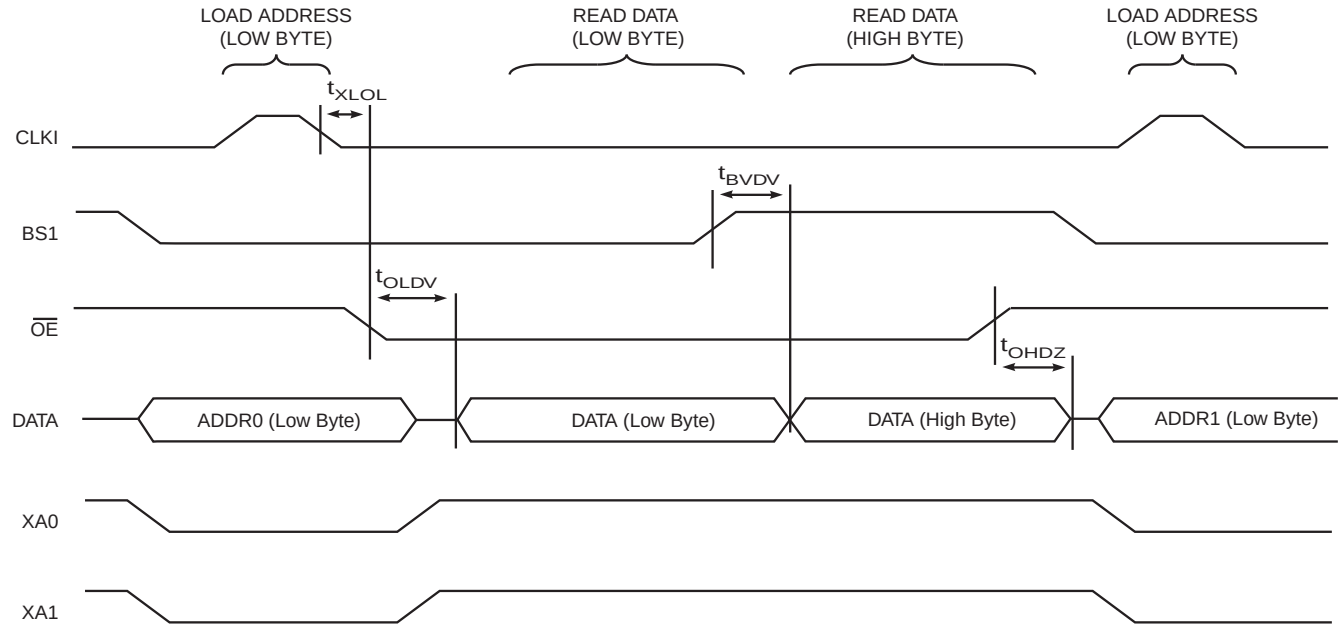


Figure 105. Parallel Programming Timing, Loading Sequence with Timing Requirements[1]



Note: 1. The timing requirements shown in Figure 104 (i.e., t_{DVXH}, t_{XHXL}, and t_{XLDX}) also apply to loading operation.

Figure 106. Parallel Programming Timing, Reading Sequence (within the Same Page) with Timing Requirements[1]



Note: 1. The timing requirements shown in Figure 104 (i.e., t_{DVXH}, t_{XHXL}, and t_{XLDX}) also apply to reading operation.

Table 114. Parallel Programming Characteristics, T = 25°C, V_{CC} = 5V

Symbol	Parameter	Min	Typ	Max	Units
V_{PP}	Programming Enable Voltage	11.5		12.5	V
I_{PP}	Programming Enable Current			250	µA
t_{DVXH}	Data and Control Valid before CLKI High	67			ns
t_{XLXH}	CLKI Low to CLKI High	200			ns
t_{XHXL}	CLKI Pulse Width High	150			ns
t_{XLDX}	Data and Control Hold after CLKI Low	67			ns
t_{XLWL}	CLKI Low to $\overline{WR}$ Low	0			ns
t_{XLPH}	CLKI Low to PAGEL high	0			ns
t_{PLXH}	PAGEL low to CLKI high	150			ns
t_{BVPH}	BS1 Valid before PAGEL High	67			ns
t_{PHPL}	PAGEL Pulse Width High	150			ns
t_{PLBX}	BS1 Hold after PAGEL Low	67			ns
t_{WLBX}	BS2/1 Hold after $\overline{WR}$ Low	67			ns
t_{PLWL}	PAGEL Low to $\overline{WR}$ Low	67			ns
t_{BVWL}	BS1 Valid to $\overline{WR}$ Low	67			ns
t_{WLWH}	$\overline{WR}$ Pulse Width Low	150			ns
t_{WLRL}	$\overline{WR}$ Low to RDY/$\overline{BSY}$ Low	0		1	µs
t_{WLRH}	$\overline{WR}$ Low to RDY/$\overline{BSY}$ High(1)	3.7		4.5	ms
t_{WLRH_CE}	$\overline{WR}$ Low to RDY/$\overline{BSY}$ High for Chip Erase(2)	7.5		9	ms
t_{XLOL}	CLKI Low to $\overline{OE}$ Low	0			ns
t_{BVDV}	BS1 Valid to DATA valid	0		250	ns
t_{OLDV}	$\overline{OE}$ Low to DATA Valid			250	ns
t_{OHDZ}	$\overline{OE}$ High to DATA Tri-stated			250	ns

Notes: 1. t_{WLRH} is valid for the Write Flash, Write EEPROM, Write Fuse bits and Write Lock bits commands.
2. t_{WLRH_CE} is valid for the Chip Erase command.

24.11 Serial Programming Characteristics

Figure 107. Serial Programming Timing

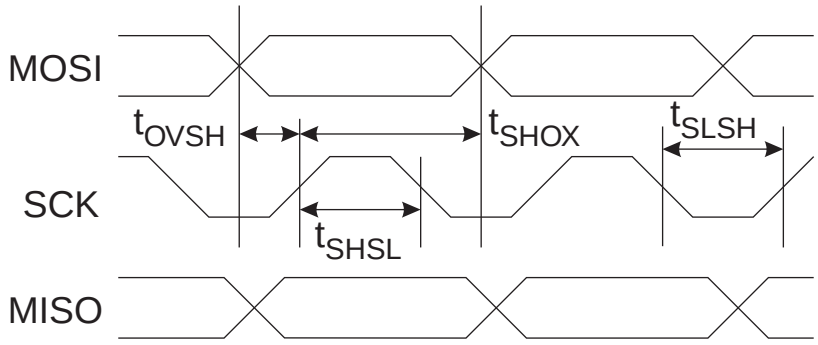


Figure 108. Serial Programming Waveform

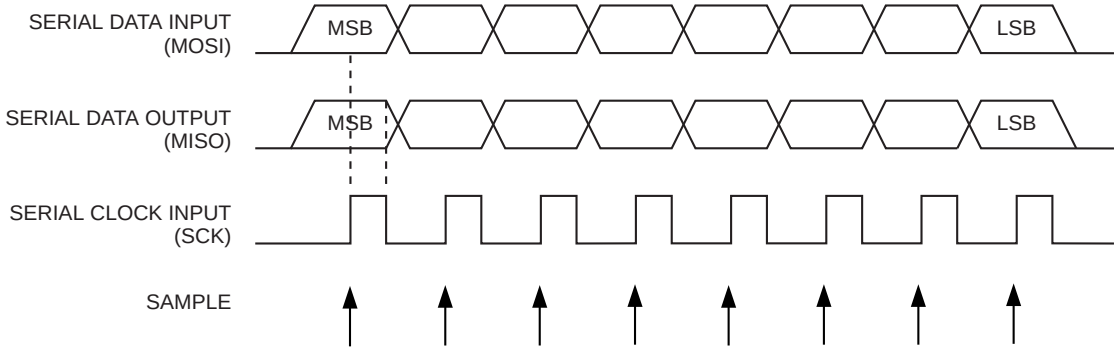


Table 115. Serial Programming Characteristics, T = -40°C to +85°C, V_{CC} = 1.7 – 5.5V (Unless Otherwise Noted)

Symbol	Parameter	Min	Typ	Max	Units
$1/t_{CLCL}$	Oscillator Frequency	0		4	MHz
t_{CLCL}	Oscillator Period	250			ns
$1/t_{CLCL}$	Oscillator Freq. (V_{CC} = 4.5V - 5.5V)	0		20	MHz
t_{CLCL}	Oscillator Period (V_{CC} = 4.5V - 5.5V)	50			ns
t_{SHSL}	SCK Pulse Width High	2 t_{CLCL} (1)			ns
t_{SLSH}	SCK Pulse Width Low	2 t_{CLCL} (1)			ns
t_{OVSH}	MOSI Setup to SCK High	t_{CLCL}			ns
t_{SHOX}	MOSI Hold after SCK High	2 t_{CLCL}			ns

Note: 1. 2 t_{CLCL} for f_{ck} < 12MHz, 3 t_{CLCL} for f_{ck} >= 12 MHz

25. Typical Characteristics

The data contained in this section is largely based on simulations and characterization of similar devices in the same process and design methods. Thus, the data should be treated as indications of how the part will behave.

The following charts show typical behavior. These figures are not tested during manufacturing. During characterisation devices are operated at frequencies higher than test limits but they are not guaranteed to function properly at frequencies higher than the ordering code indicates.

All current consumption measurements are performed with all I/O pins configured as inputs and with internal pull-ups enabled. Current consumption is a function of several factors such as operating voltage, operating frequency, loading of I/O pins, switching rate of I/O pins, code executed and ambient temperature. The dominating factors are operating voltage and frequency.

A sine wave generator with rail-to-rail output is used as clock source but current consumption in Power-Down mode is independent of clock selection. The difference between current consumption in Power-Down mode with Watchdog Timer enabled and Power-Down mode with Watchdog Timer disabled represents the differential current drawn by the Watchdog Timer.

The current drawn from pins with a capacitive load may be estimated (for one pin) as follows:

$$I_{CP} \approx V_{CC} \times C_L \times f_{SW}$$

where V_{CC} = operating voltage, C_L = load capacitance and f_{SW} = average switching frequency of I/O pin.

25.1 Current Consumption in Active Mode

Figure 109. Active Supply Current vs. Low Frequency (0.1 - 1.0 MHz)

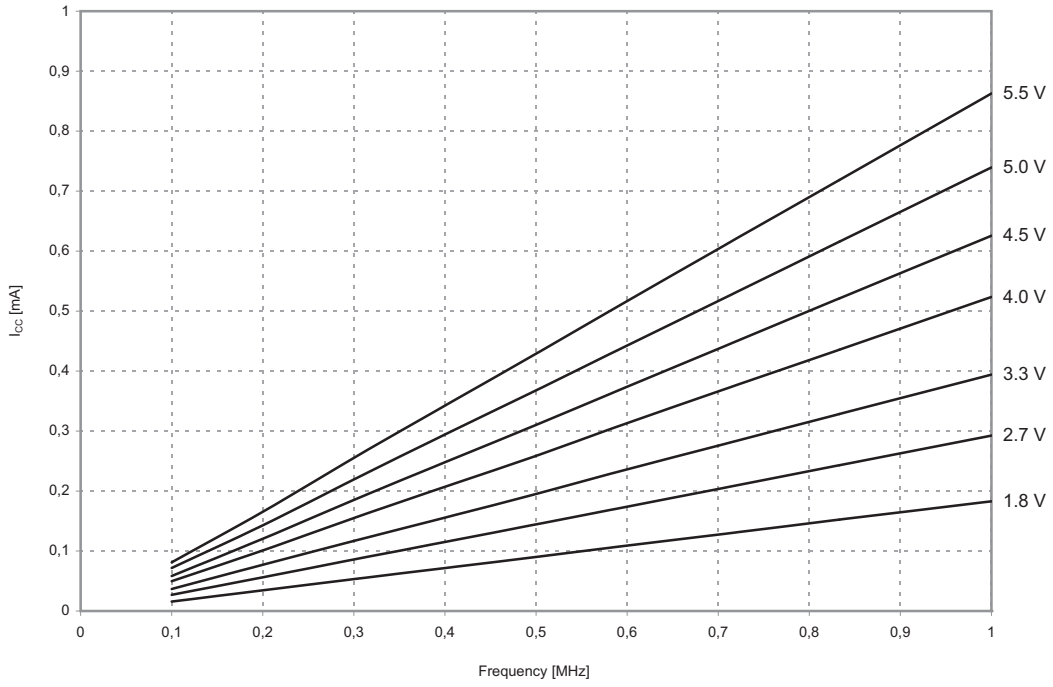


Figure 110. Active Supply Current vs. Frequency (1 - 20 MHz)

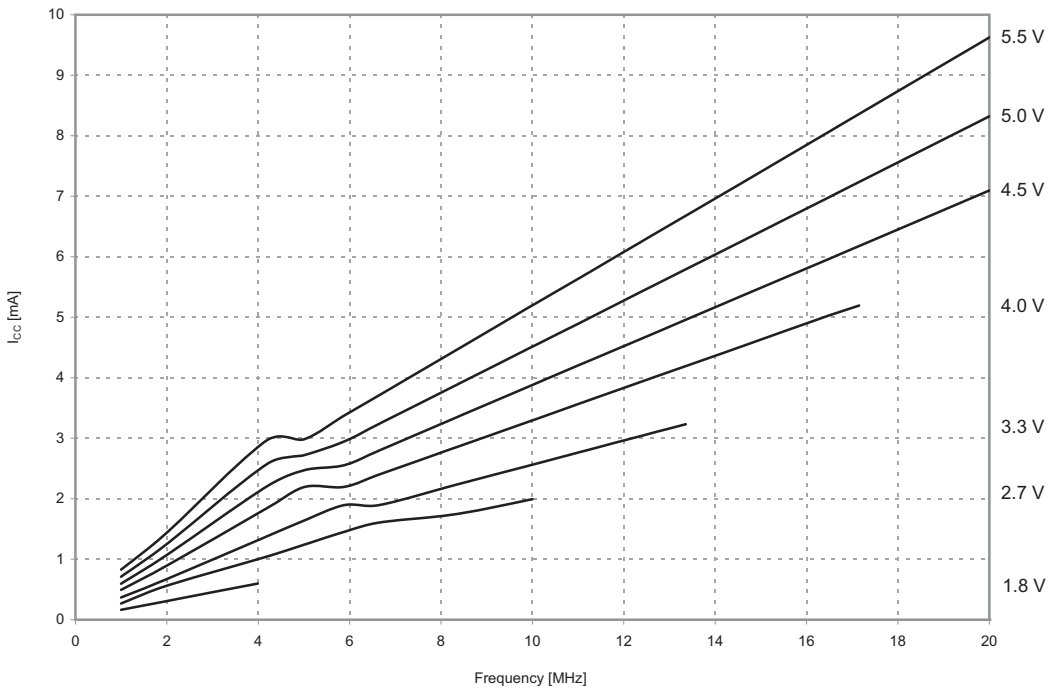


Figure 111. Active Supply Current vs. V_{CC} (Internal Oscillator, 8 MHz)

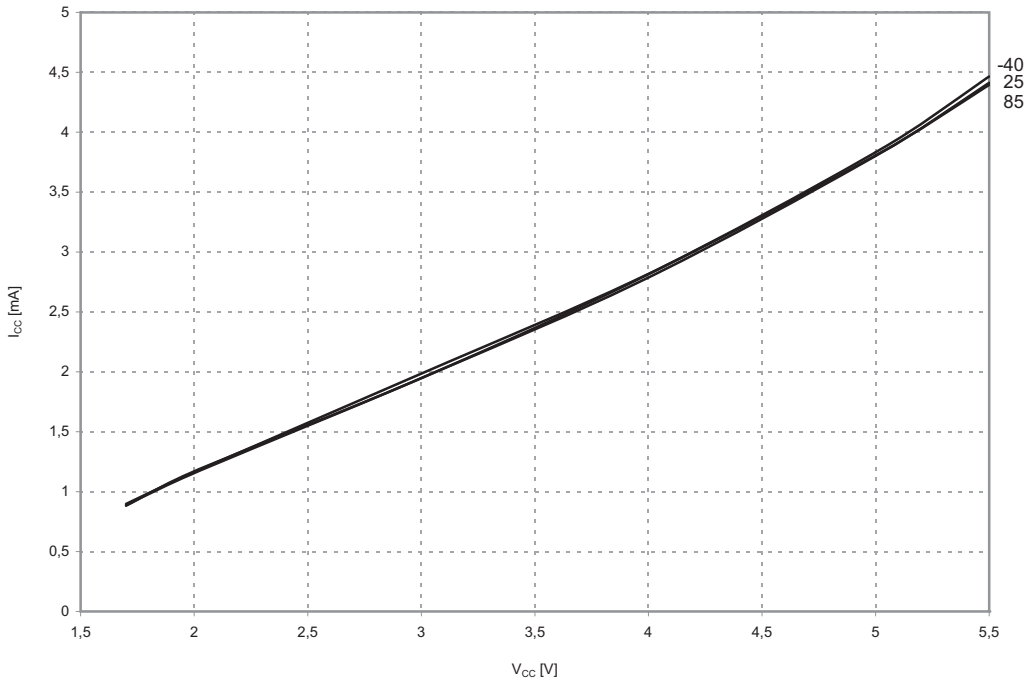


Figure 112. Active Supply Current vs. V_{CC} (Internal Oscillator, 1 MHz)

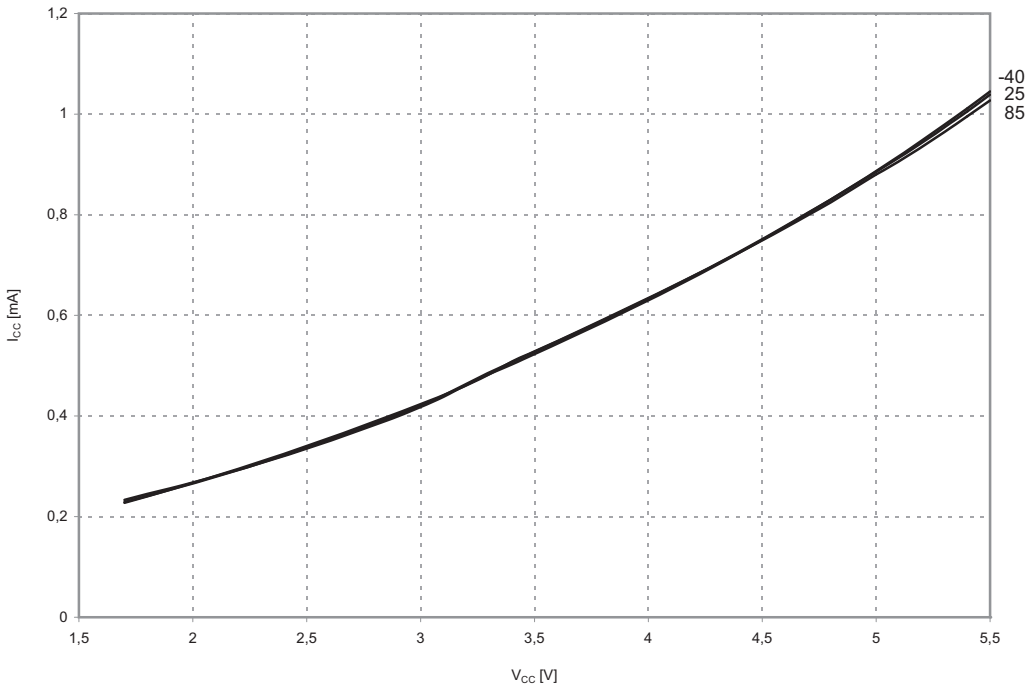
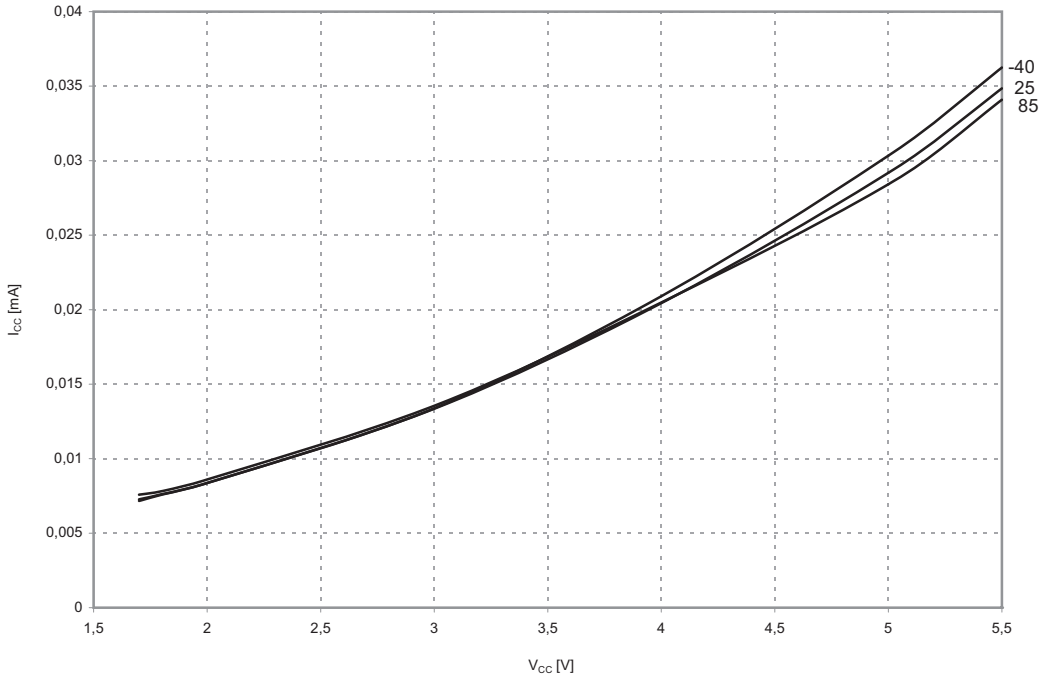


Figure 113. Active Supply Current vs. V_{CC} (Internal Oscillator, 32kHz)



25.2 Current Consumption in Idle Mode

Figure 114. Idle Supply Current vs. Low Frequency (0.1 - 1.0 MHz)

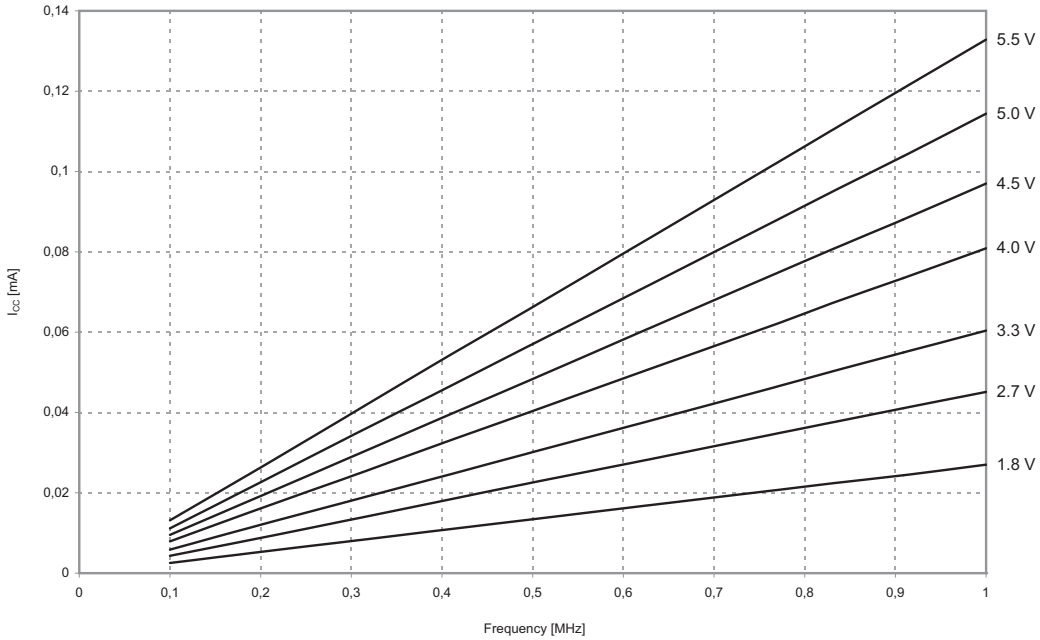


Figure 115. Idle Supply Current vs. Frequency (1 - 20 MHz)

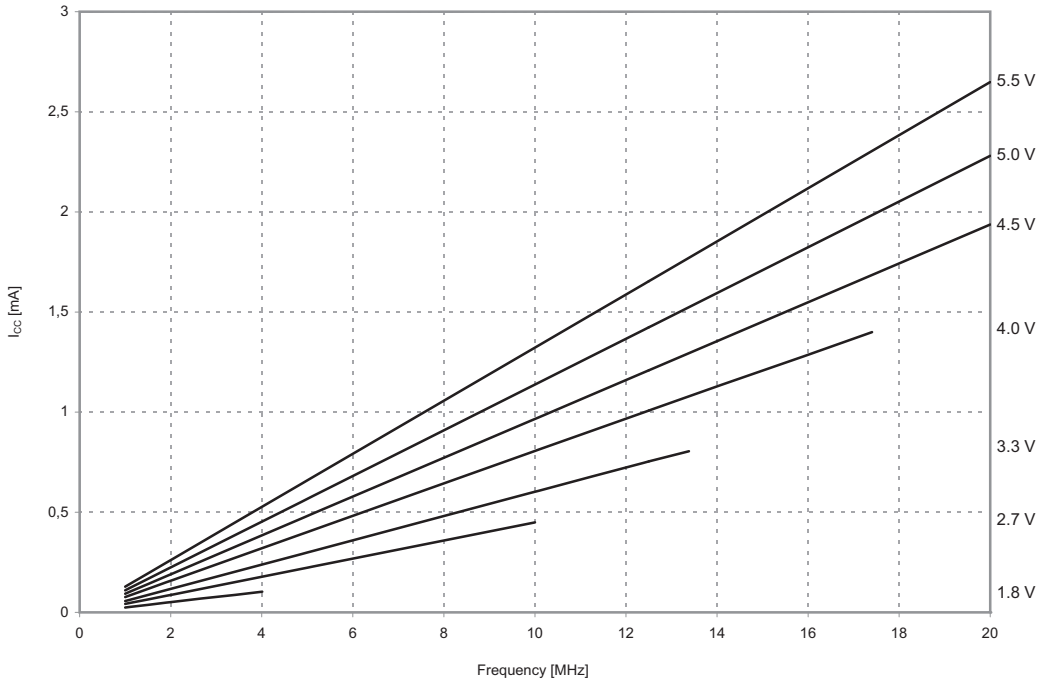


Figure 116. Idle Supply Current vs. V_{CC} (Internal Oscillator, 8 MHz)

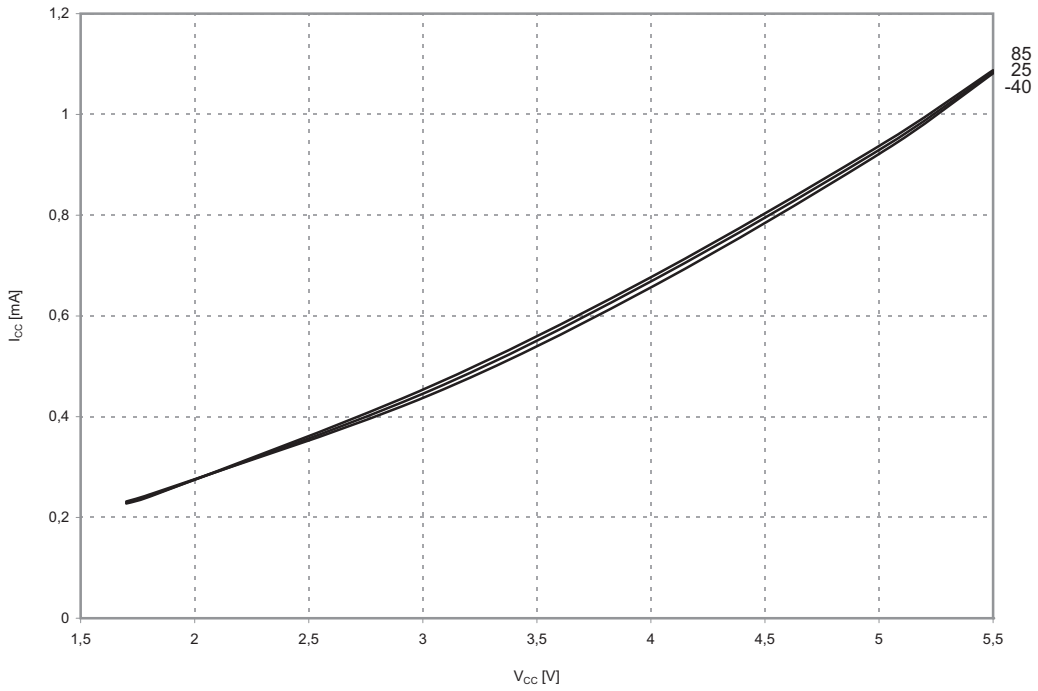


Figure 117. Idle Supply Current vs. V_{CC} (Internal Oscillator, 1 MHz)

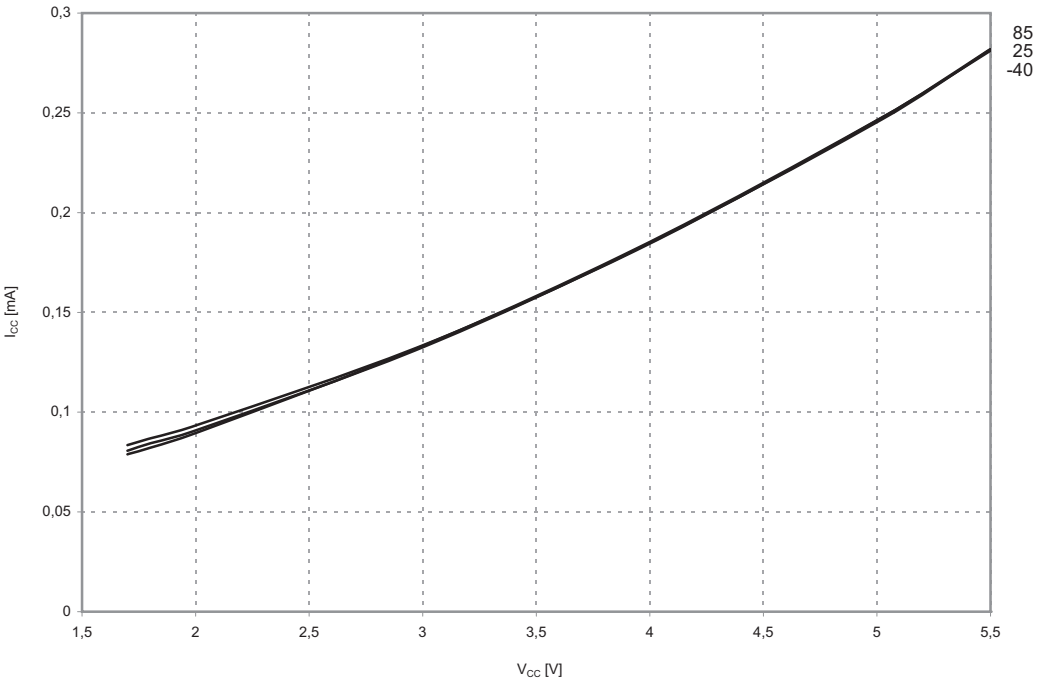
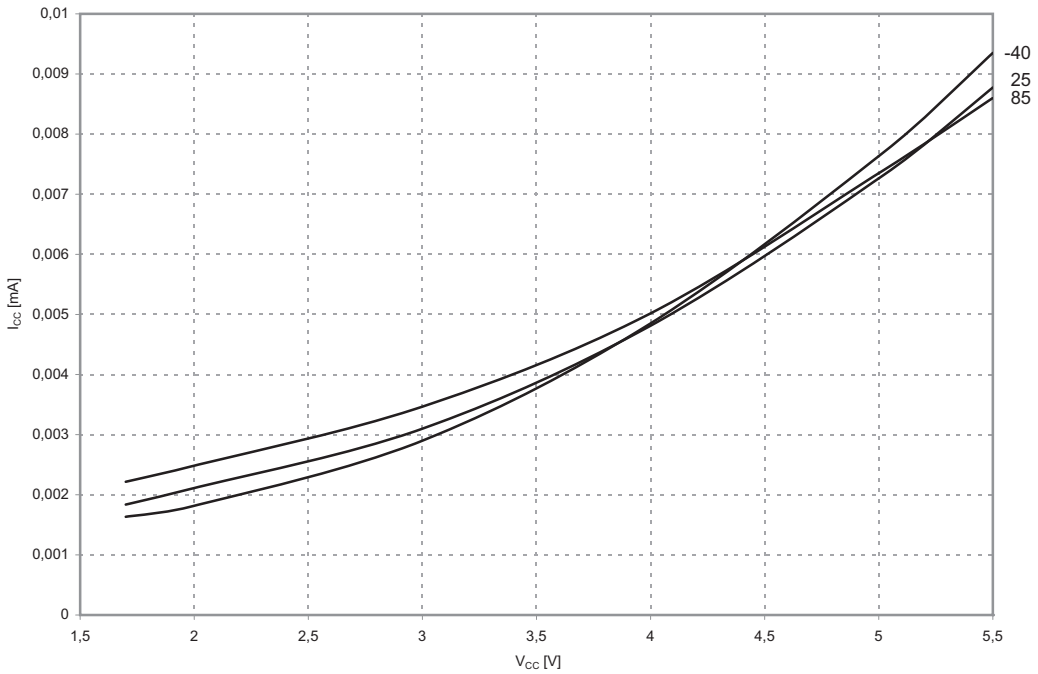


Figure 118. Idle Supply Current vs. V_{CC} (Internal Oscillator, 32kHz)



25.3 Current Consumption in Power-down Mode

Figure 119. Power-down Supply Current vs. V_{CC} (Watchdog Timer Disabled)

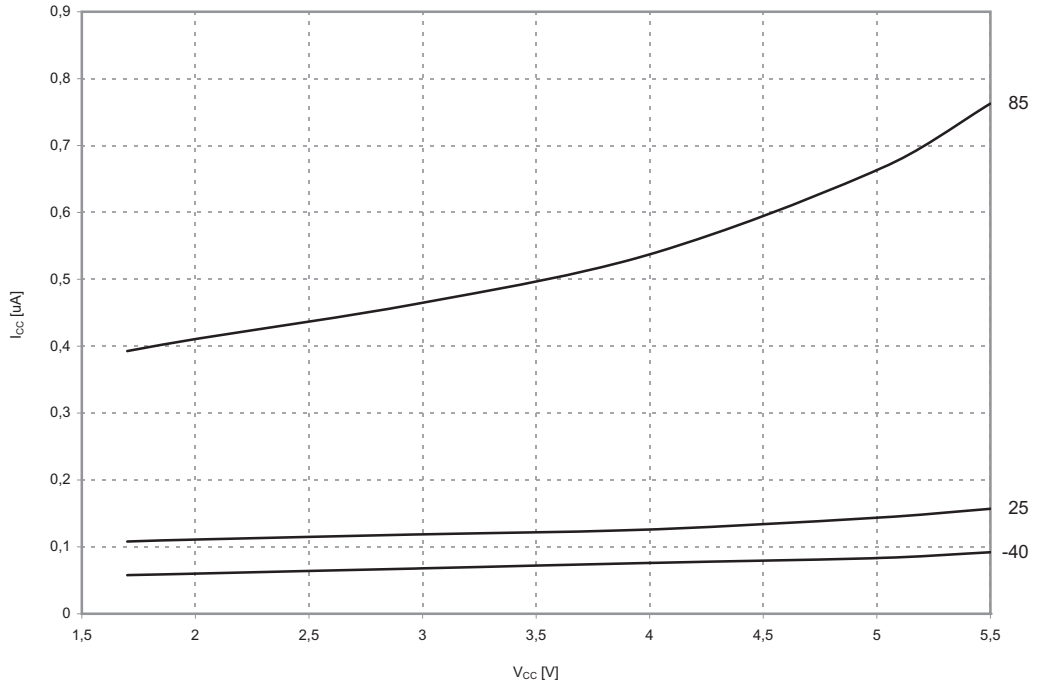
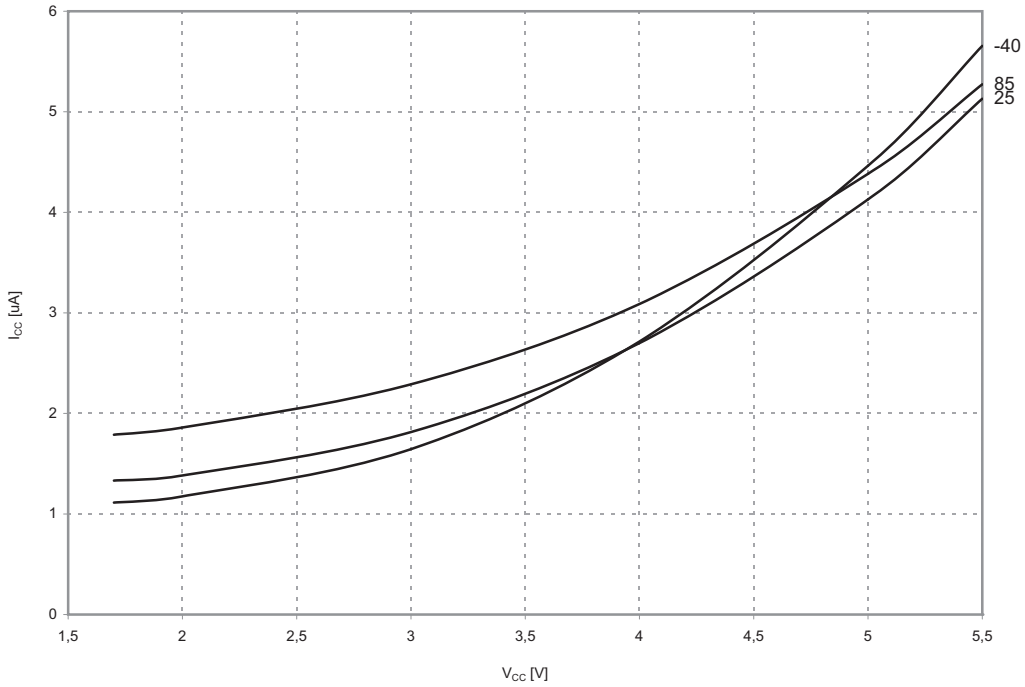


Figure 120. Power-down Supply Current vs. V_{CC} (Watchdog Timer Enabled)



25.4 Current Consumption in Reset

Figure 121. Reset Current vs. Frequency (0.1 – 1MHz, Excluding Pull-Up Current)

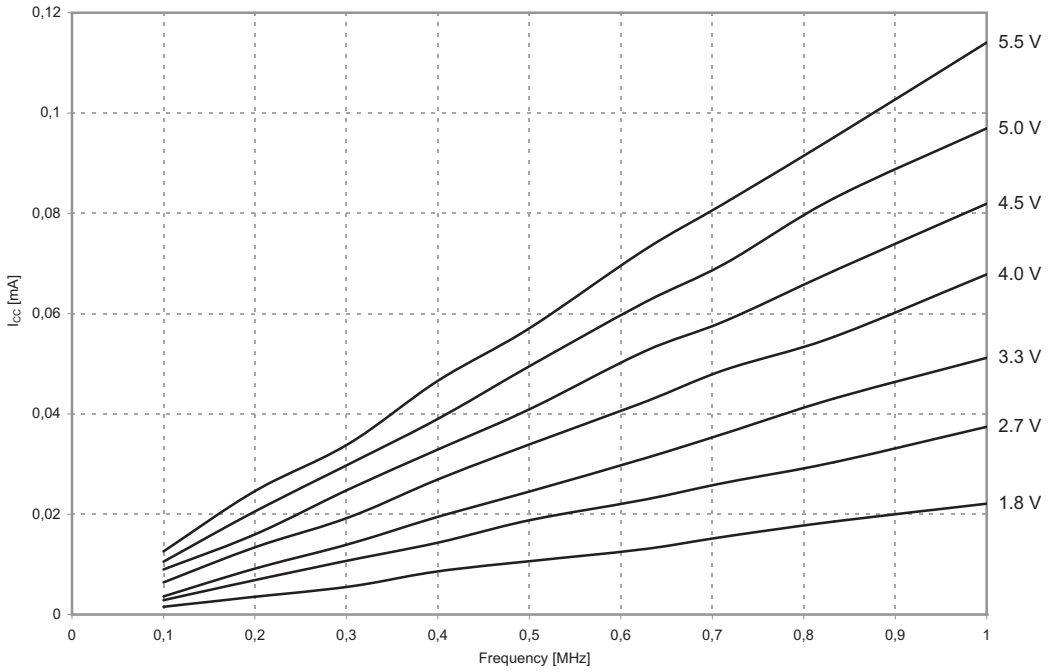


Figure 122. Reset Current vs. Frequency (1 – 20MHz, Excluding Pull-Up Current)

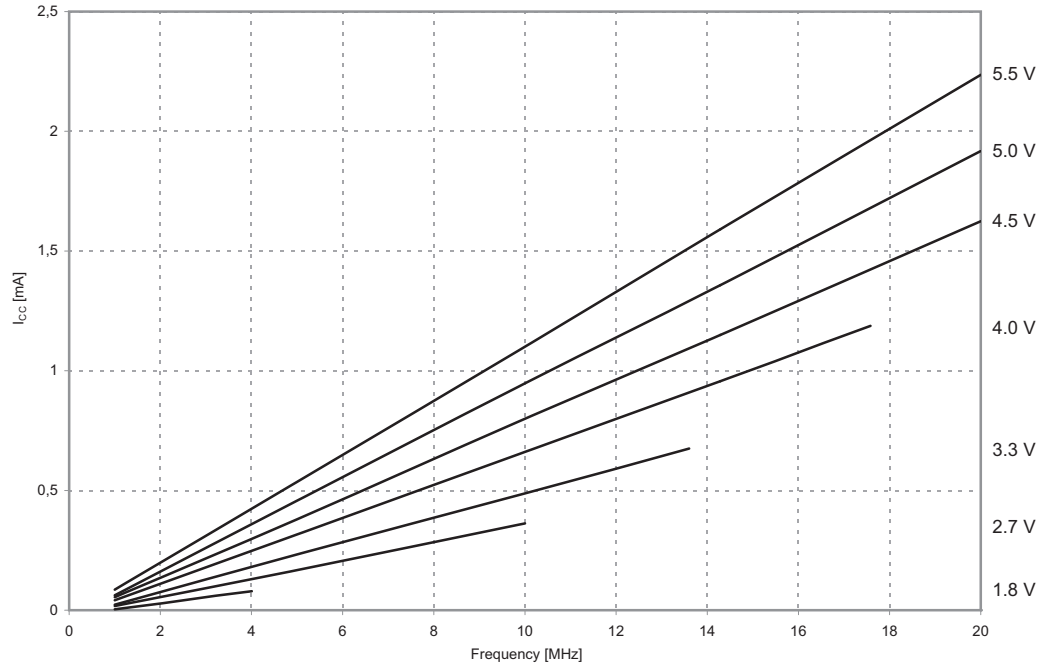
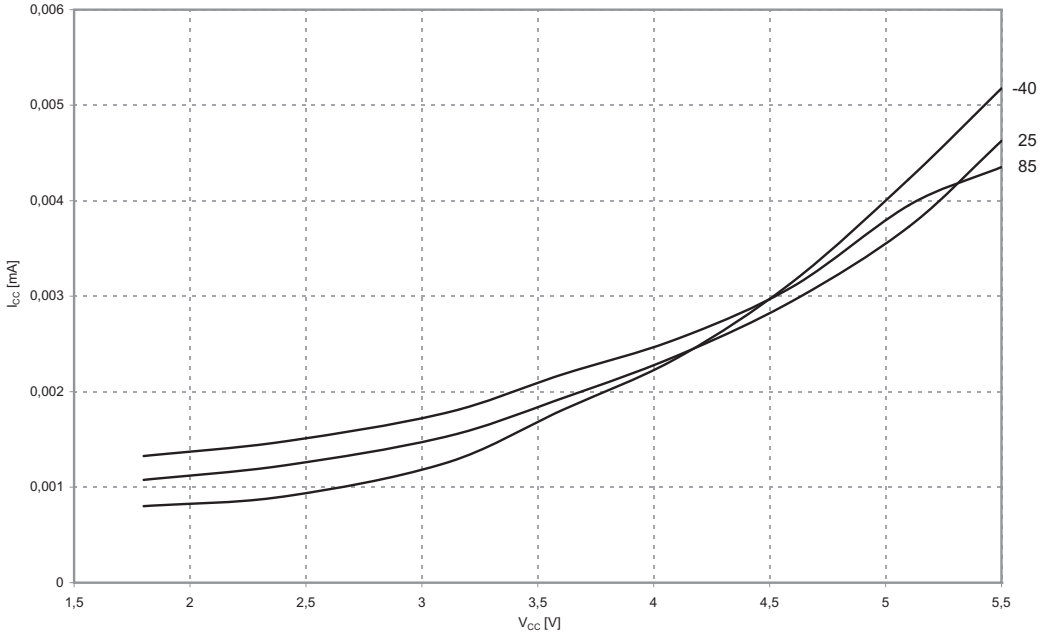


Figure 123. Reset Current vs. V_{CC} (No Clock, excluding Reset Pull-Up Current)



25.5 Current Consumption of Peripheral Units

Figure 124. Current Consumption of Peripherals at 4MHz vs. V_{CC}

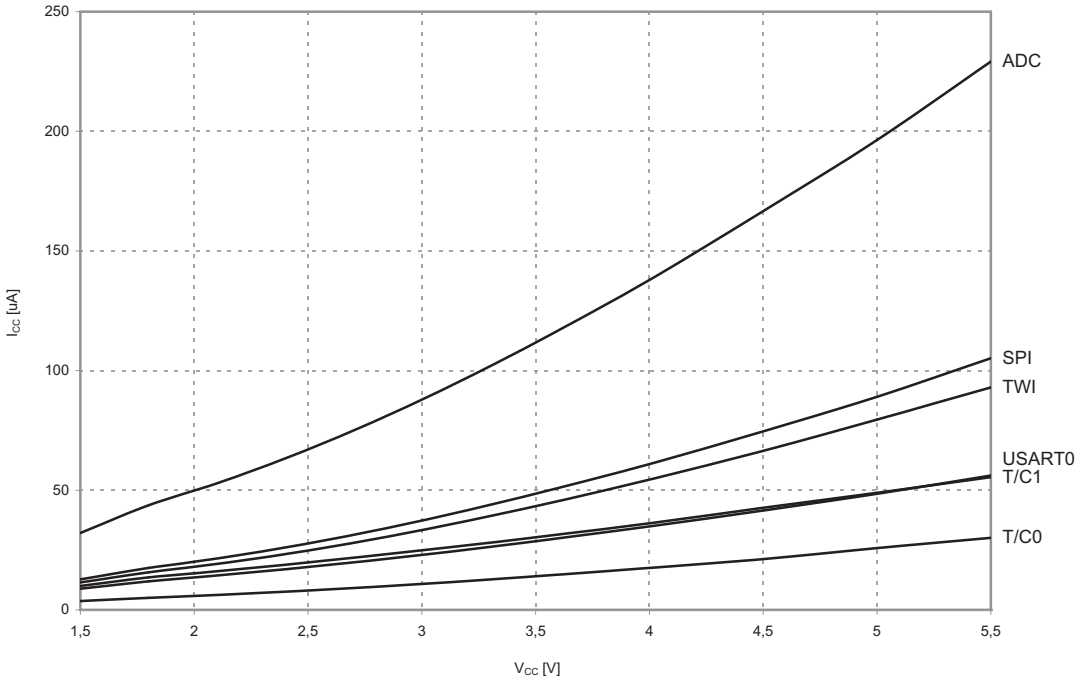


Figure 125. Current Consumption of Peripherals at 3V vs. Frequency

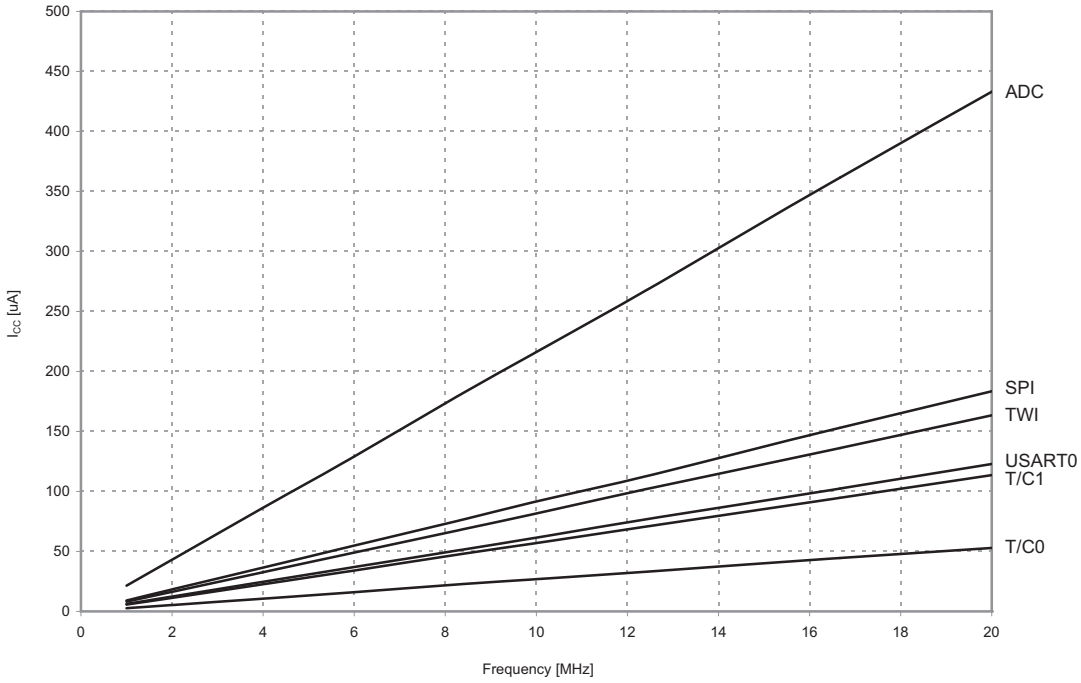


Figure 126. Watchdog Timer Current vs. V_{CC}

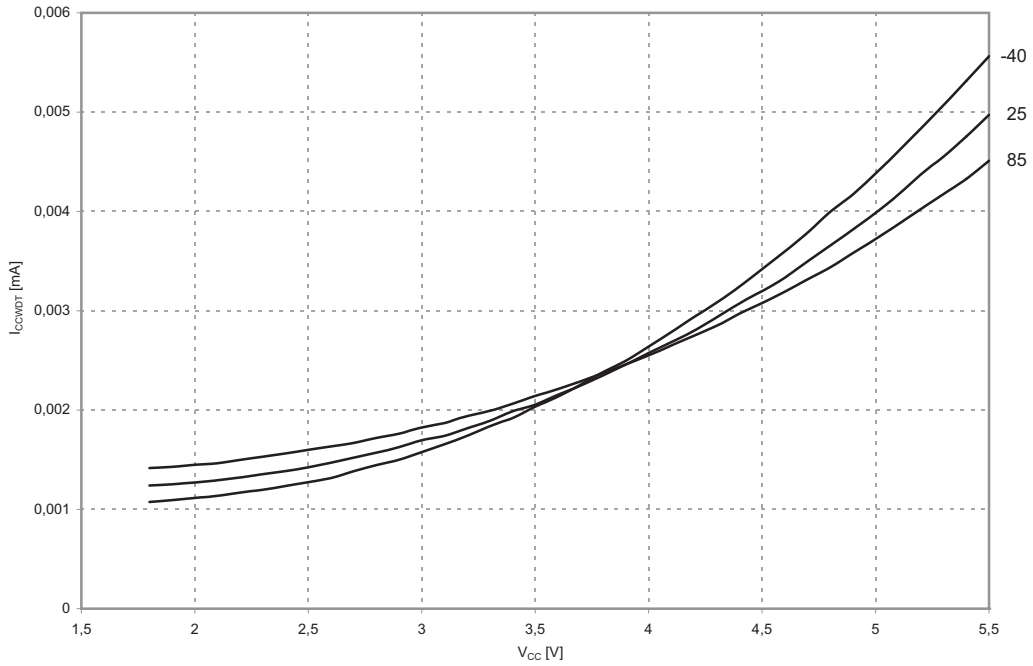


Figure 127. Brownout Detector Current vs. V_{CC}

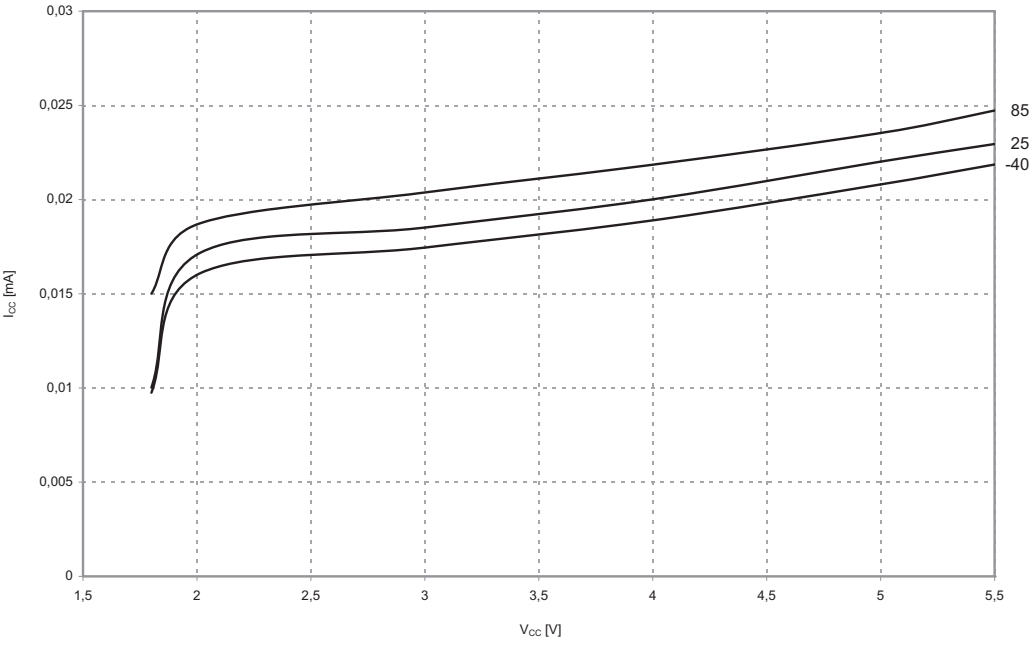
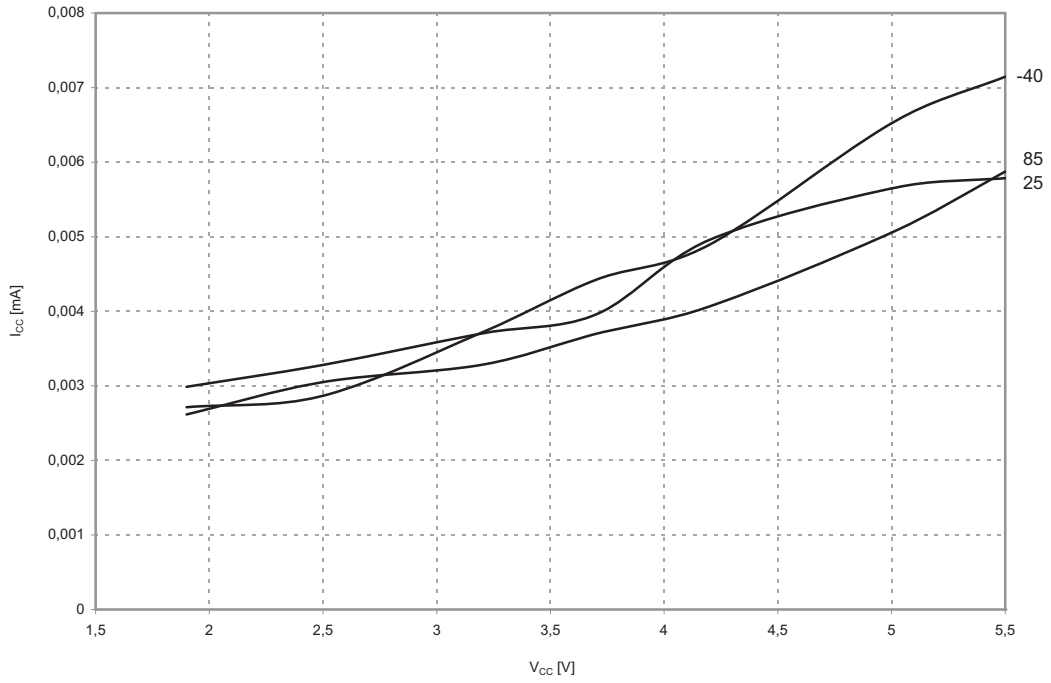


Figure 128. Sampled Brownout Detector Current vs. V_{CC}



25.6 Pull-up Resistors

25.6.1 I/O Pins

Figure 129. I/O pin Pull-up Resistor Current vs. Input Voltage (V_{CC} = 1.8V)

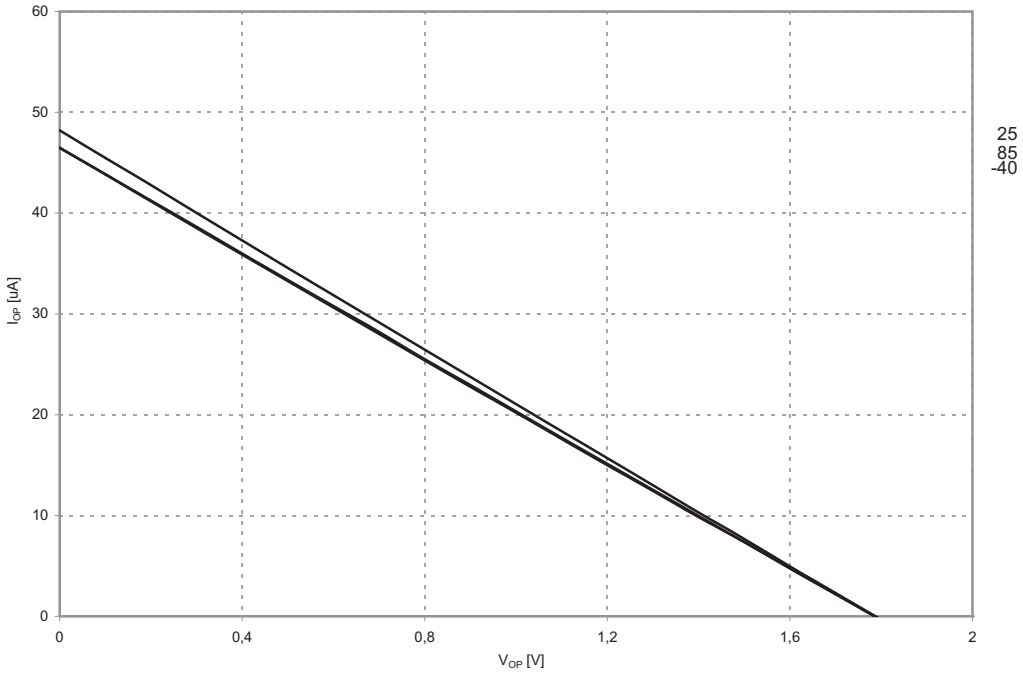


Figure 130. I/O Pin Pull-up Resistor Current vs. input Voltage (V_{CC} = 2.7V)

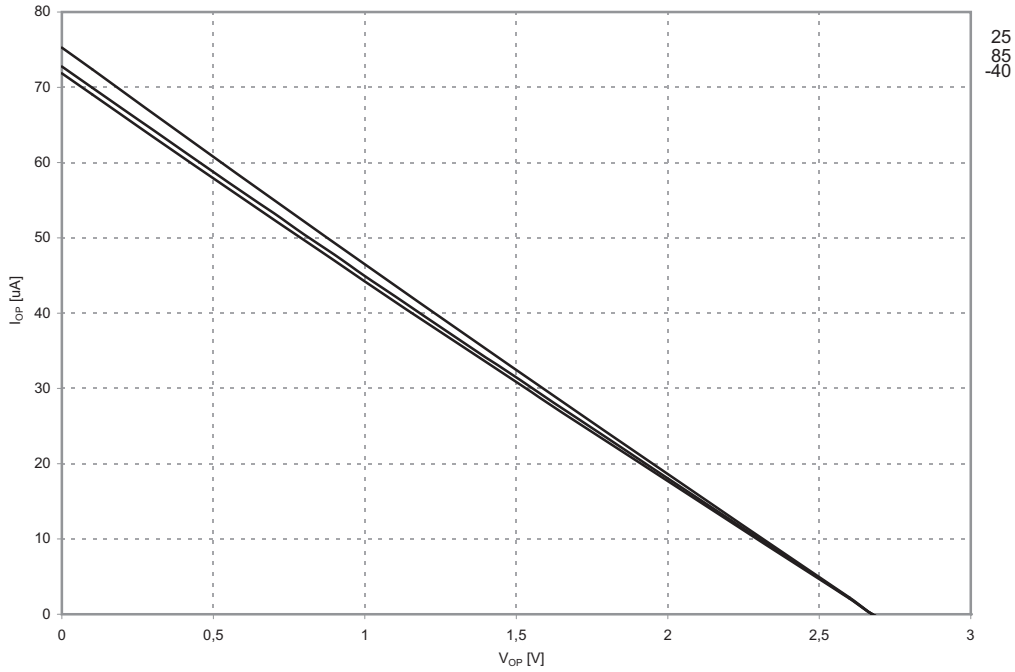
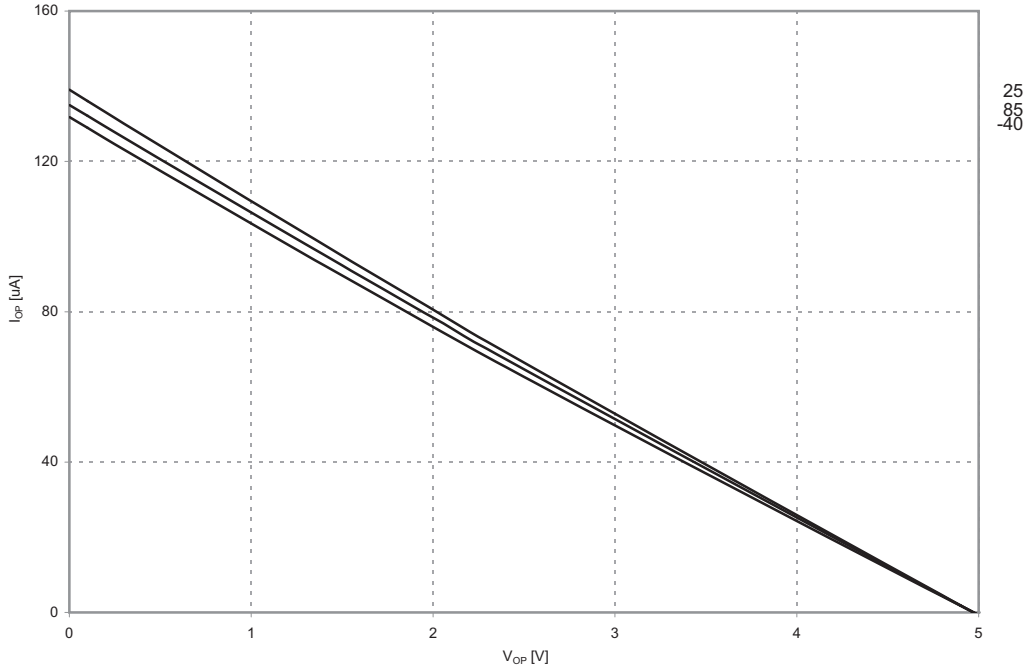


Figure 131. I/O pin Pull-up Resistor Current vs. Input Voltage (V_{CC} = 5V)



25.6.2 Reset Pin

Figure 132. Reset Pull-up Resistor Current vs. Reset Pin Voltage (V_{CC} = 1.8V)

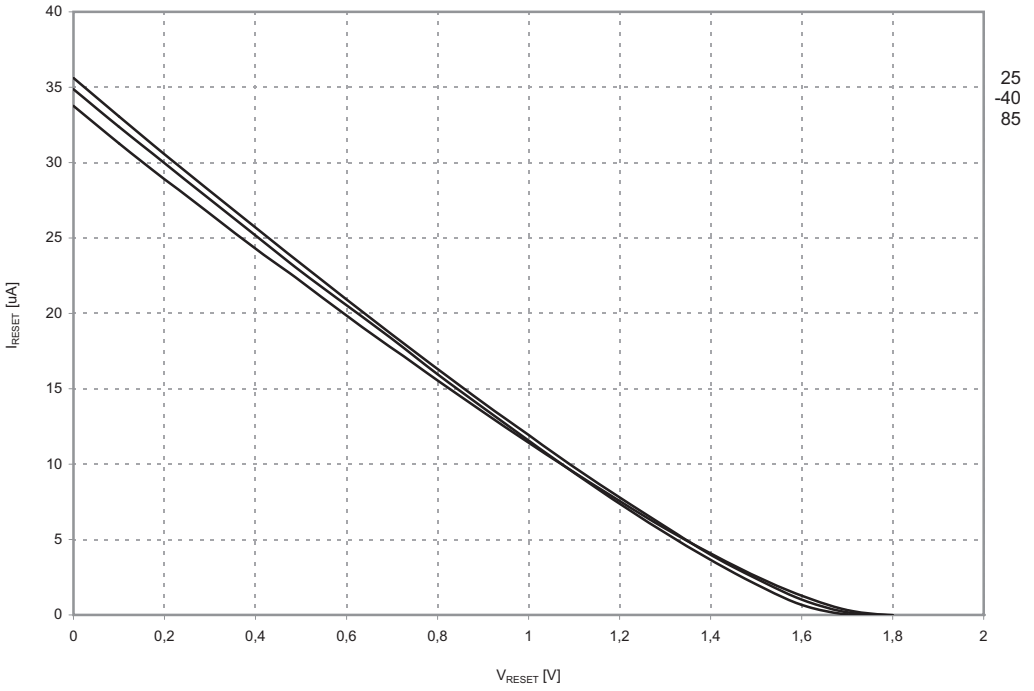


Figure 133. Reset Pull-up Resistor Current vs. Reset Pin Voltage (V_{CC} = 2.7V)

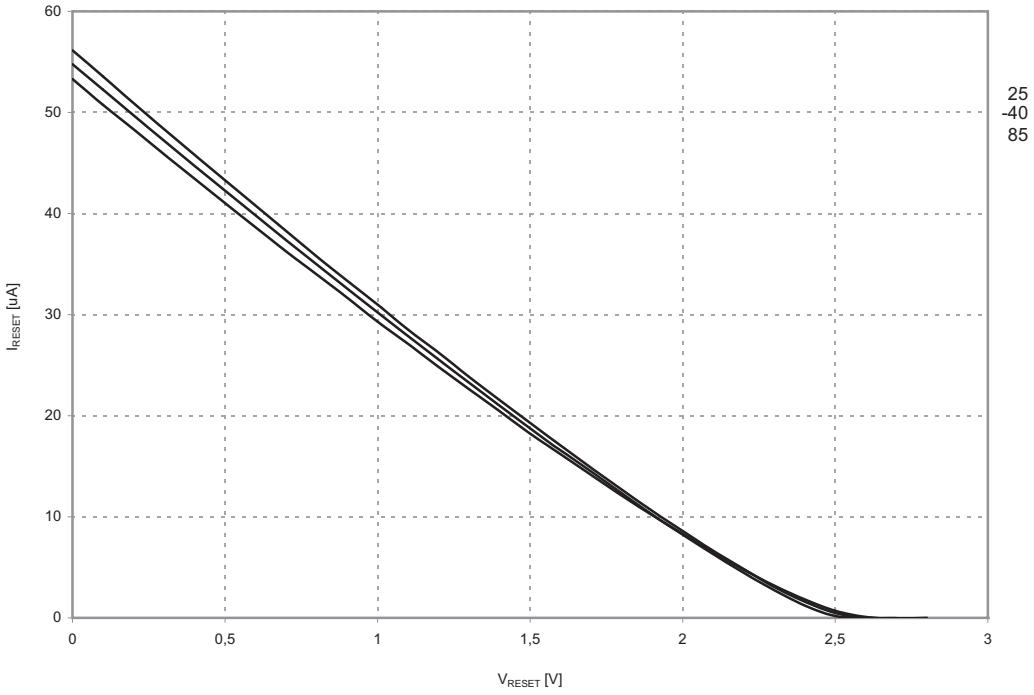
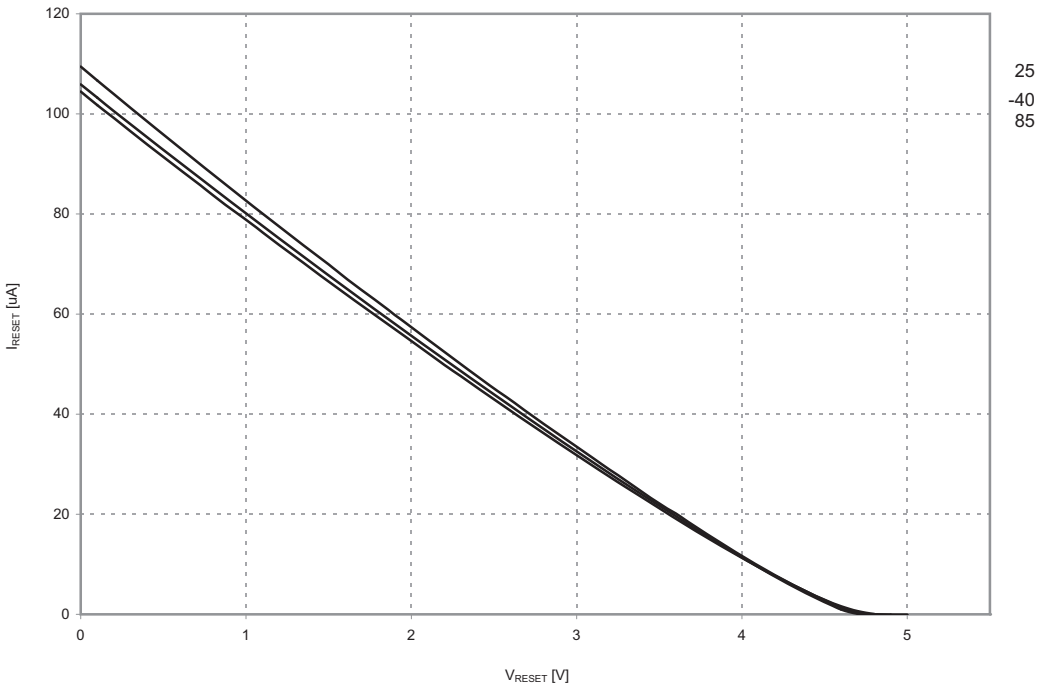


Figure 134. Reset Pull-up Resistor Current vs. Reset Pin Voltage (V_{CC} = 5V)



25.7 Input Thresholds

25.7.1 I/O Pins

Figure 135. V_{IH}: Input Threshold Voltage vs. V_{CC} (I/O Pin, Read as '1')

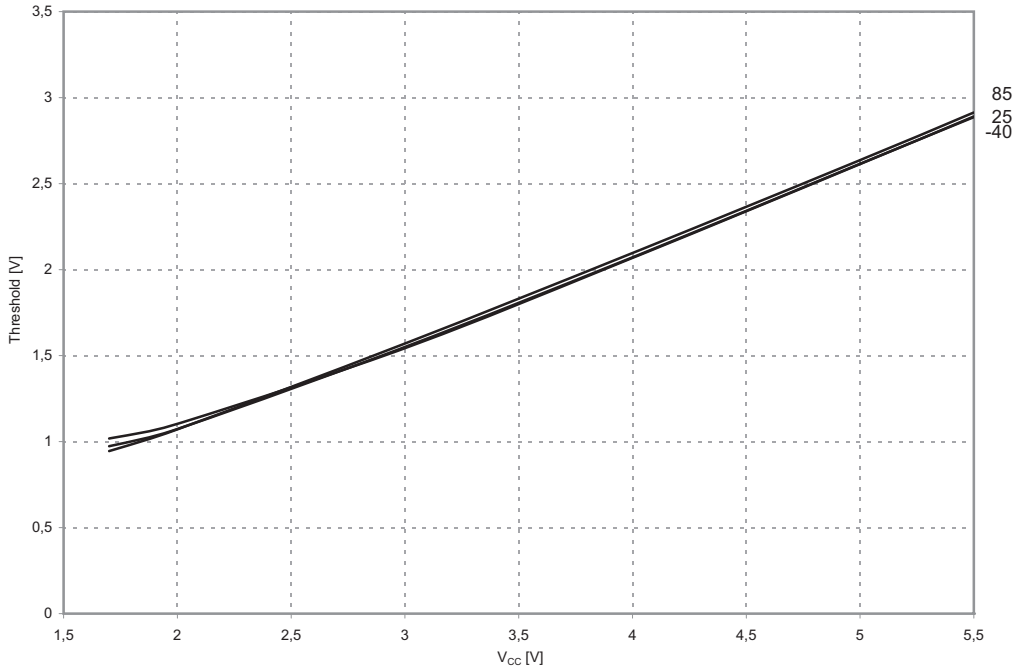


Figure 136. V_{IL}: Input Threshold Voltage vs. V_{CC} (I/O Pin, Read as '0')

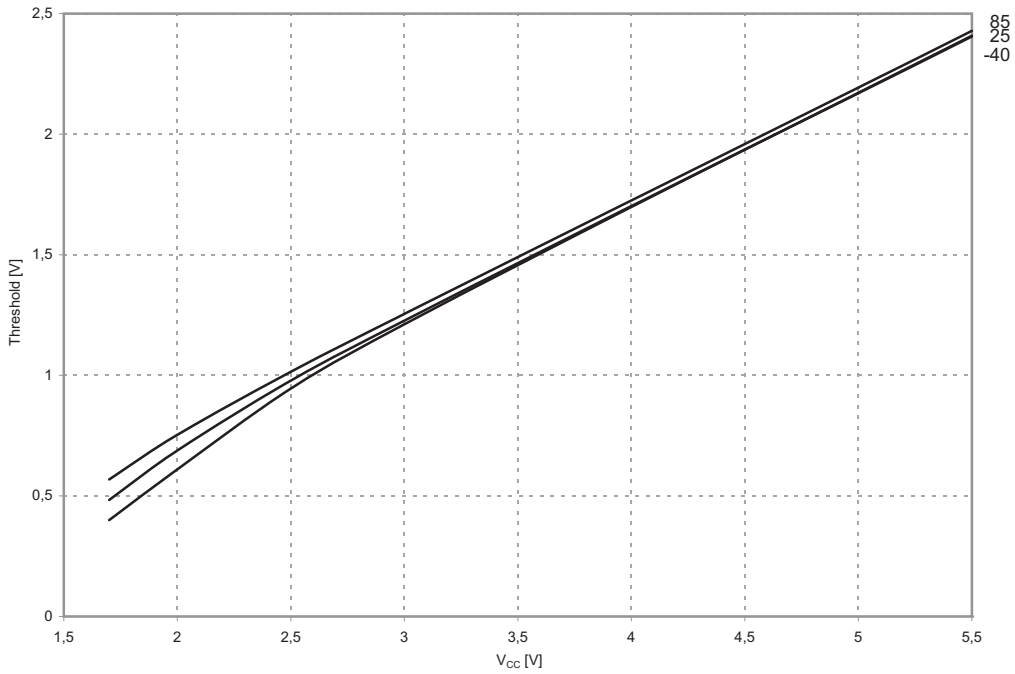
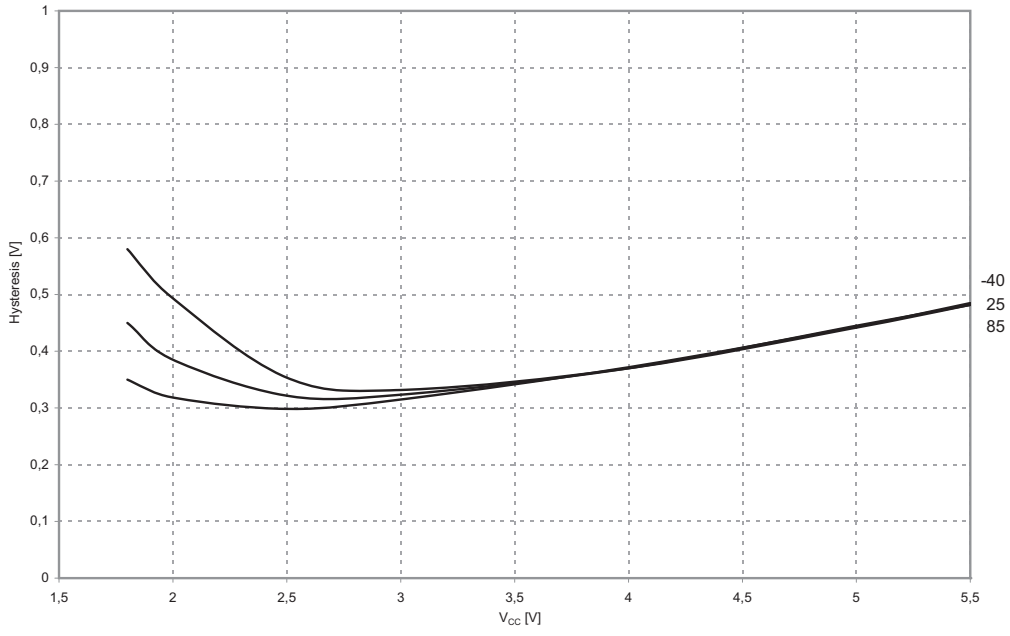


Figure 137. V_{IH}-V_{IL}: Input Hysteresis vs. V_{CC} (I/O Pin)



25.7.2 TWI Pins

Figure 138. V_{IH}: Input Threshold Voltage vs. V_{CC} (I/O Pin, Read as '1')

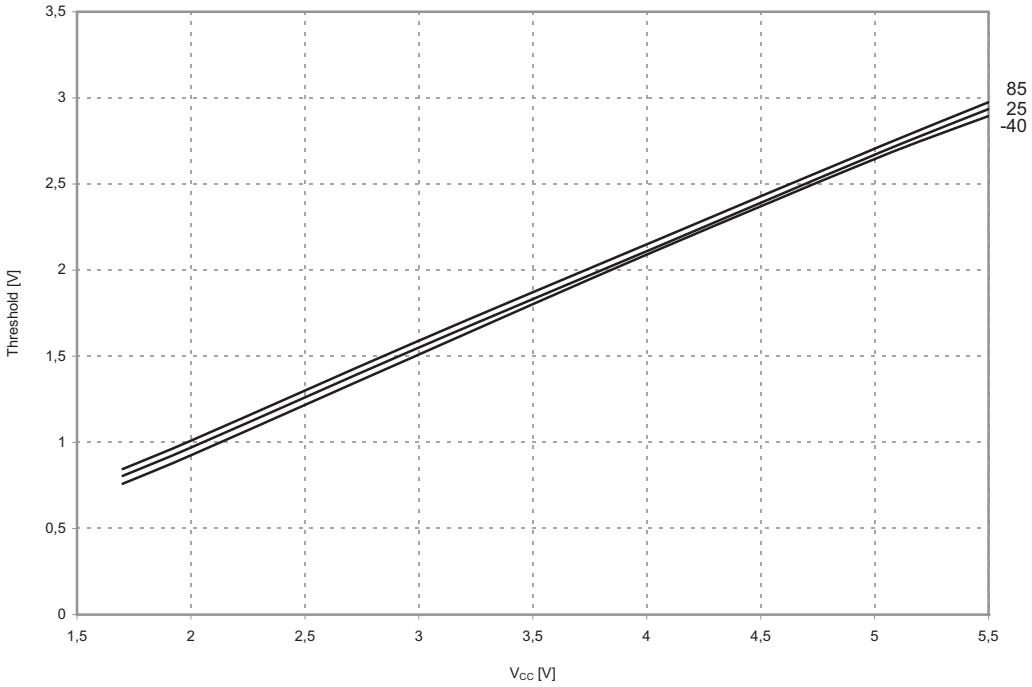


Figure 139. V_{IL}: Input Threshold Voltage vs. V_{CC} (I/O Pin, Read as '0')

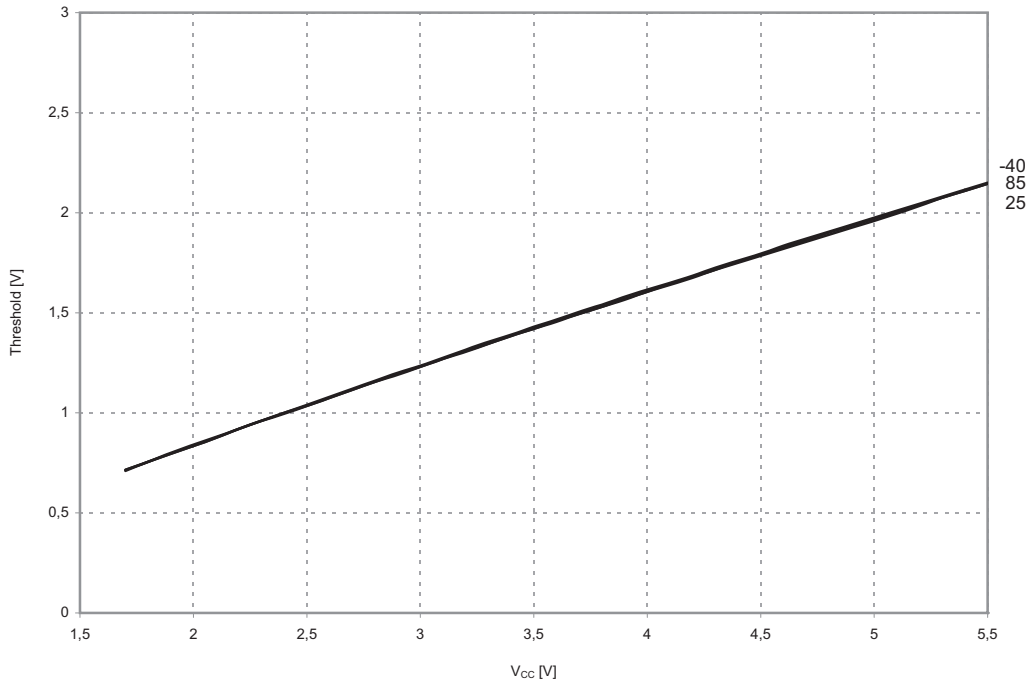
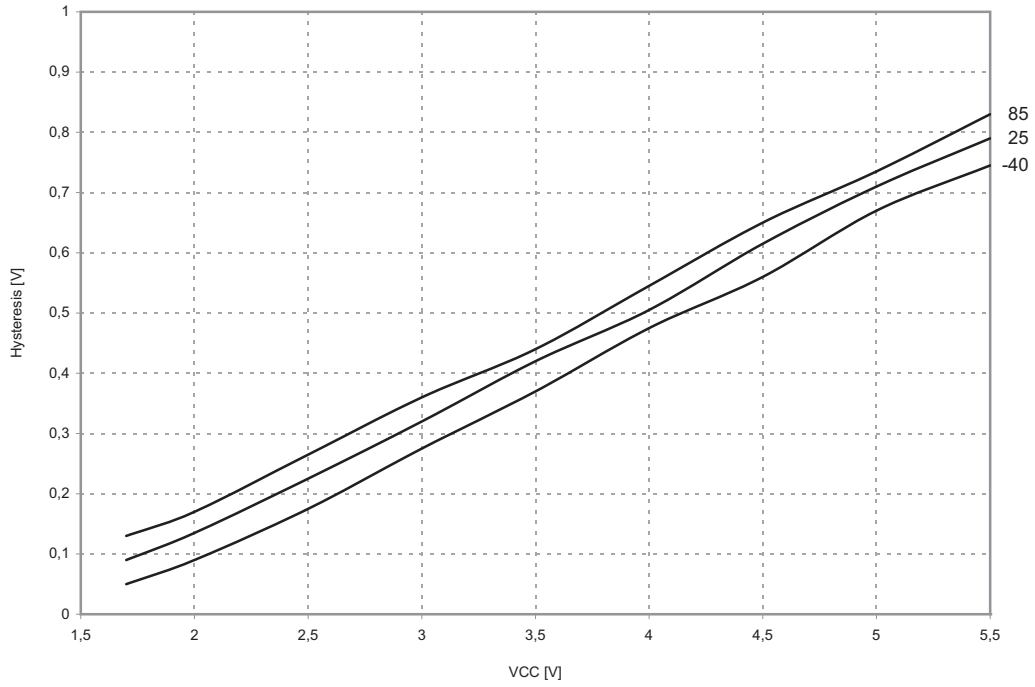


Figure 140. V_{IH}-V_{IL}: Input Hysteresis vs. V_{CC} (I/O Pin)



25.7.3 Reset Pin as I/O

Figure 141. V_{IH}: Input Threshold Voltage vs. V_{CC} (Reset Pin as I/O, Read as '1')

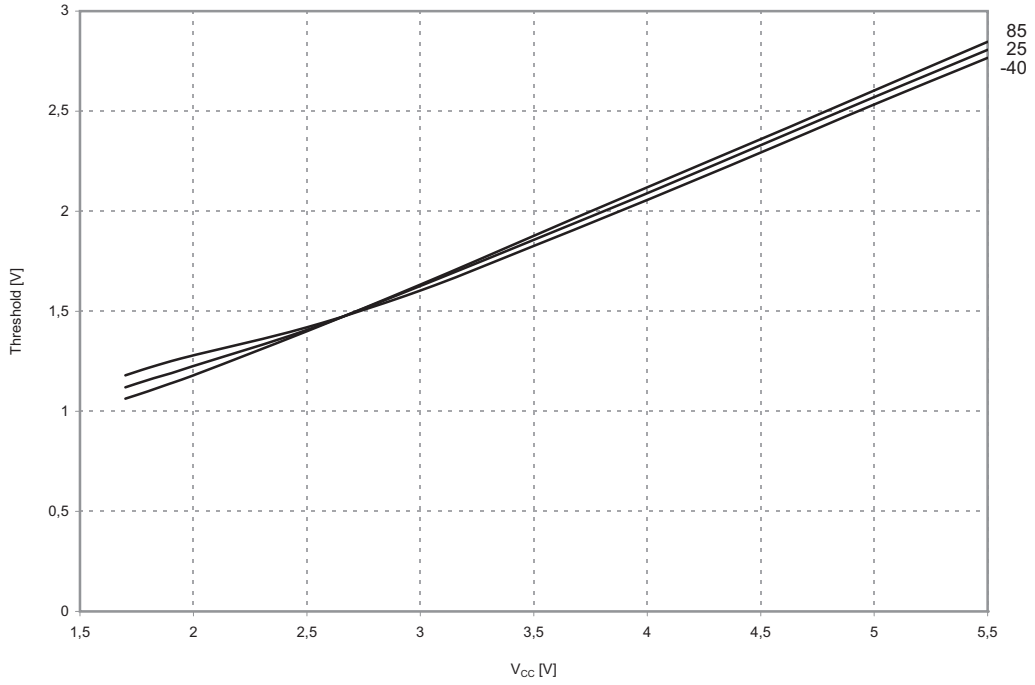


Figure 142. V_{IL}: Input Threshold Voltage vs. V_{CC} (Reset Pin as I/O, Read as '0')

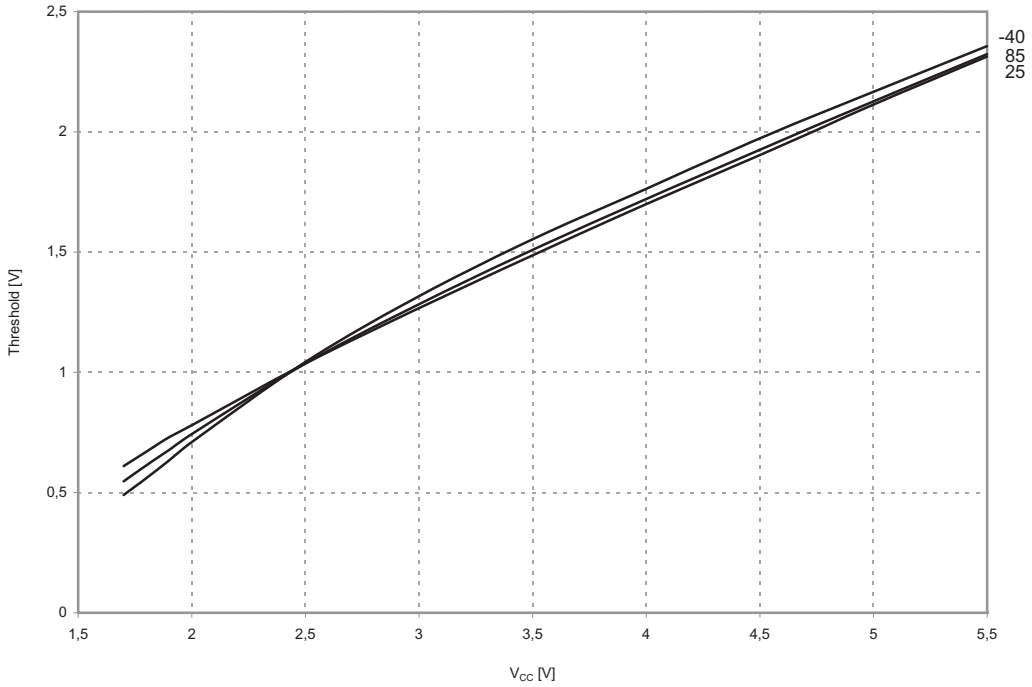
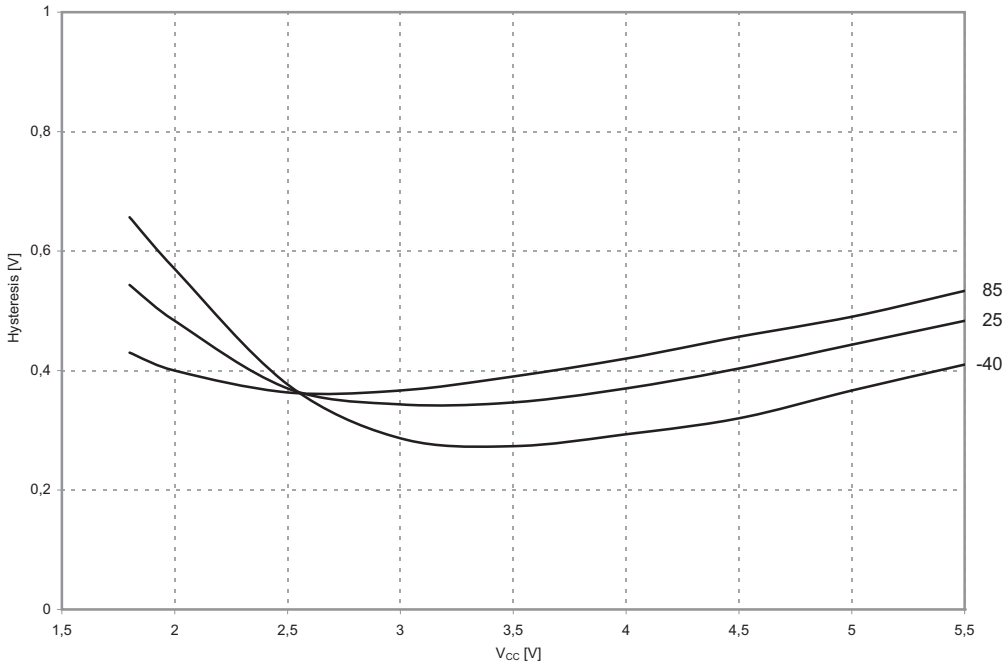


Figure 143. V_{IH}-V_{IL}: Input Hysteresis vs. V_{CC} (Reset Pin as I/O)



25.7.4 Reset Pin

Figure 144. V_{IH}: Input Threshold Voltage vs. V_{CC} (Reset Pin, Read as '1')

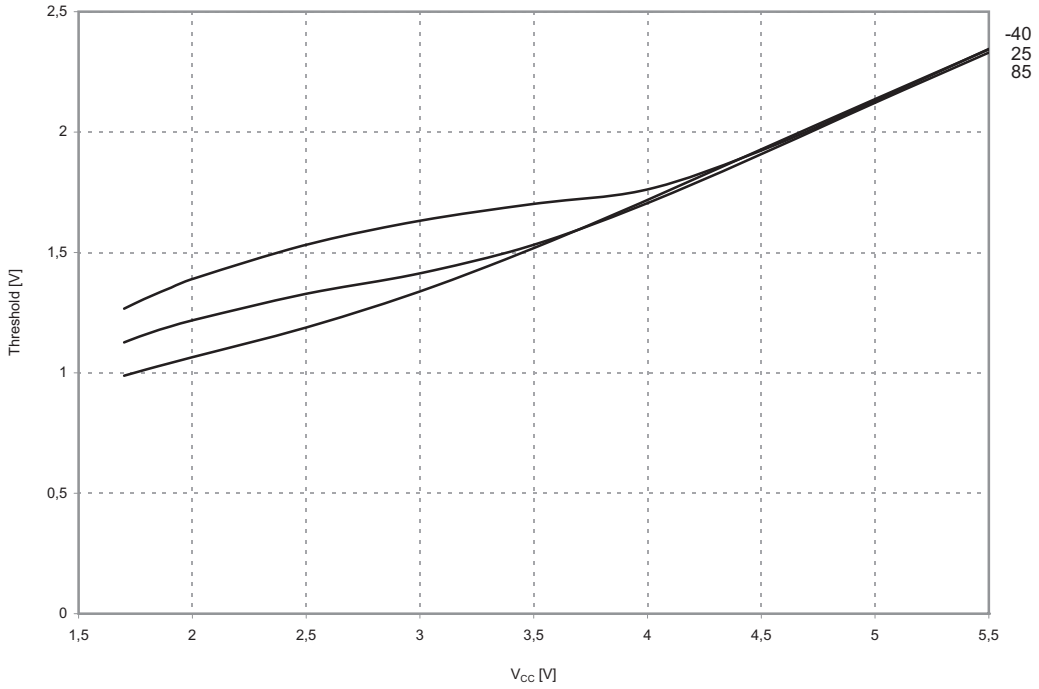


Figure 145. V_{IL}: Input Threshold Voltage vs. V_{CC} (Reset Pin, Read as '0')

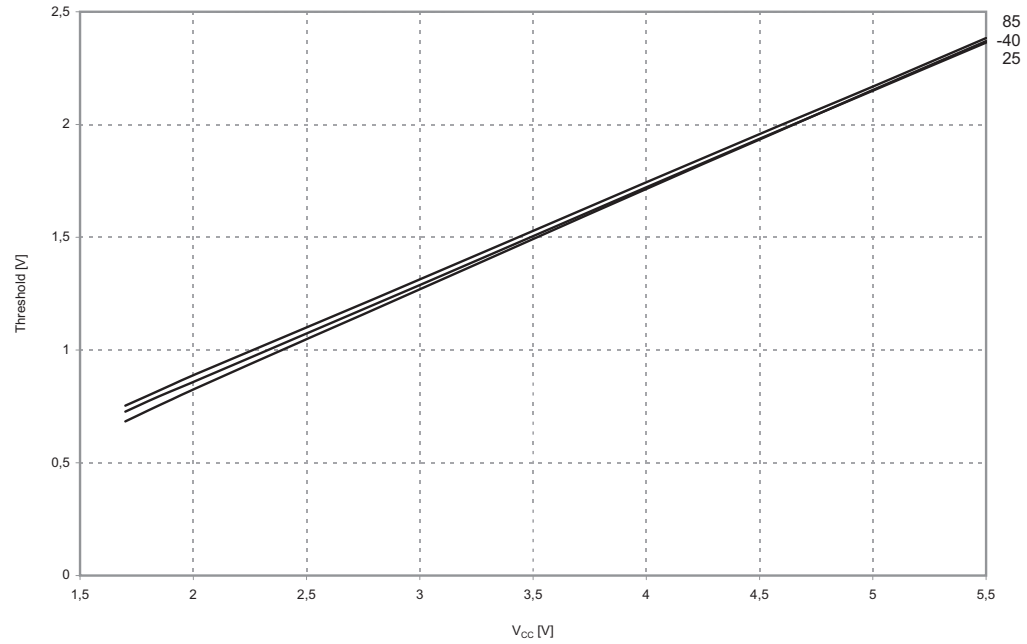
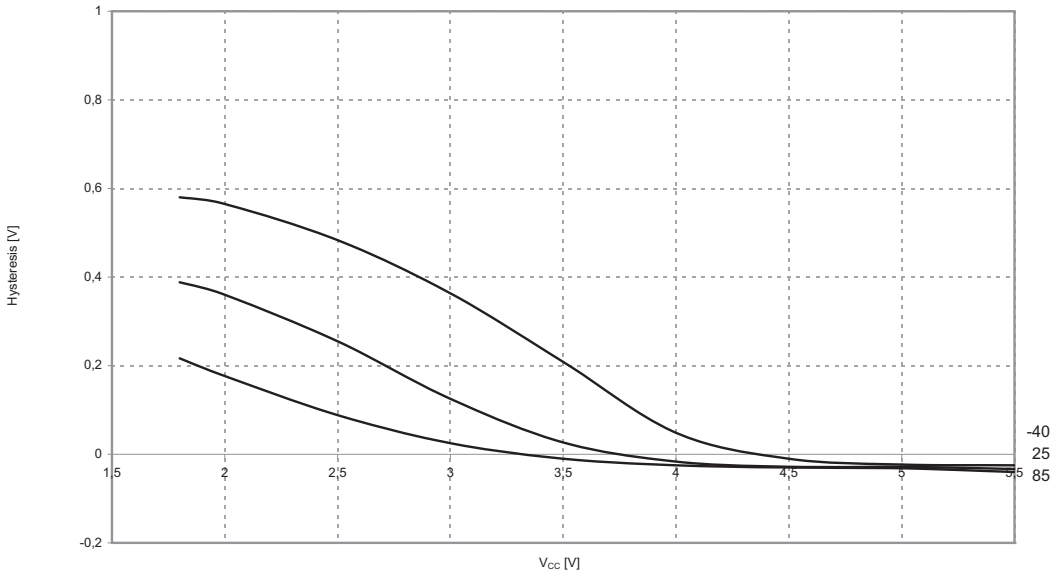


Figure 146. V_{IH}-V_{IL}: Input Hysteresis vs. V_{CC} (Reset Pin)



25.8 Current Source Strength

25.8.1 I/O Pins

Figure 147. V_{OH}: Output Voltage vs. Source Current (I/O Pin, V_{CC} = 1.8V)

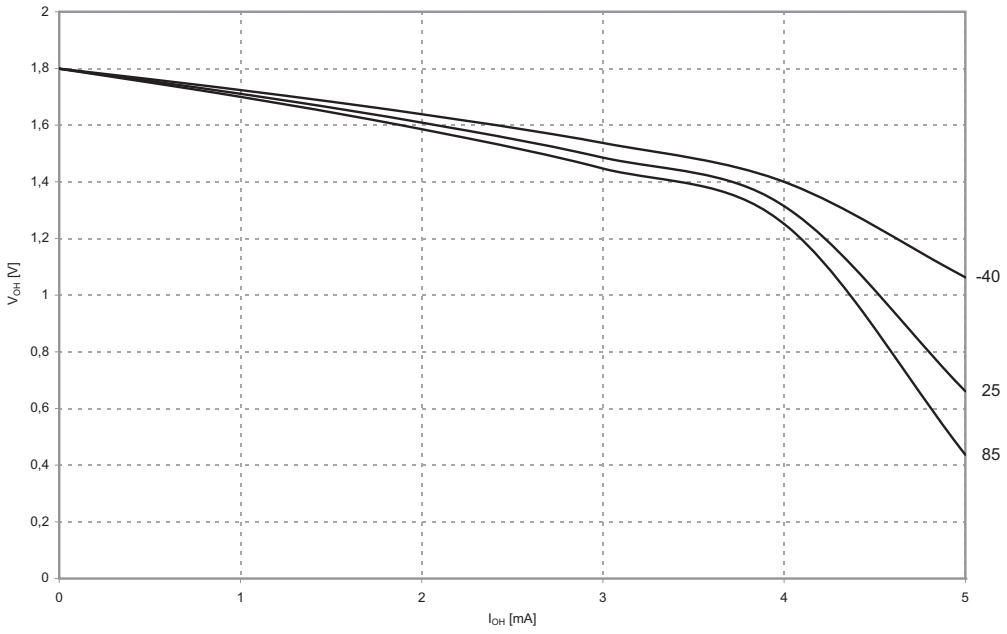


Figure 148. V_{OH}: Output Voltage vs. Source Current (I/O Pin, V_{CC} = 3V)

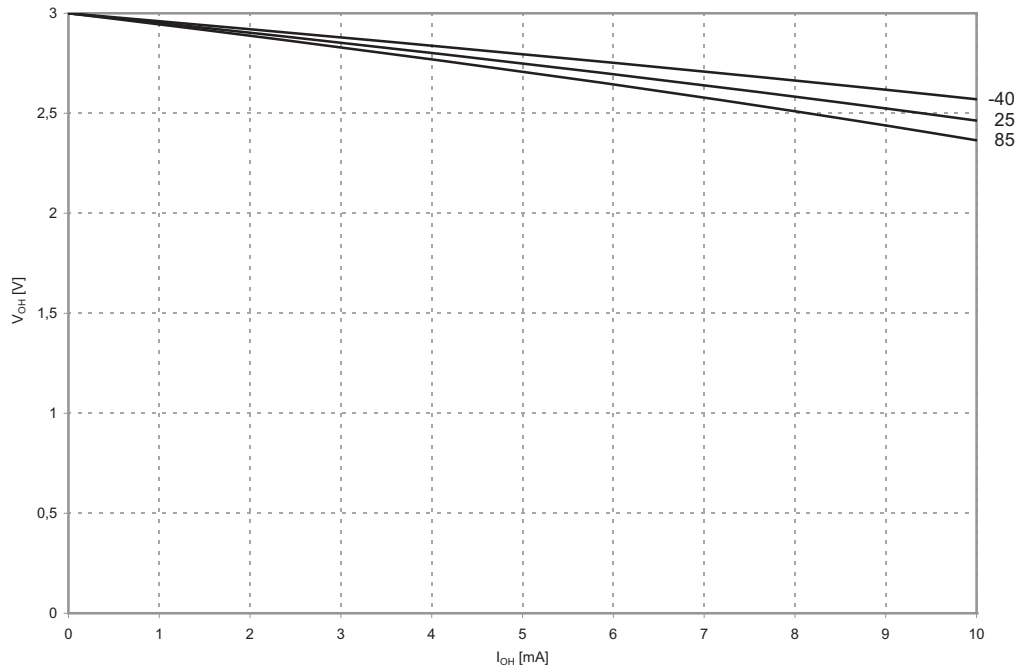
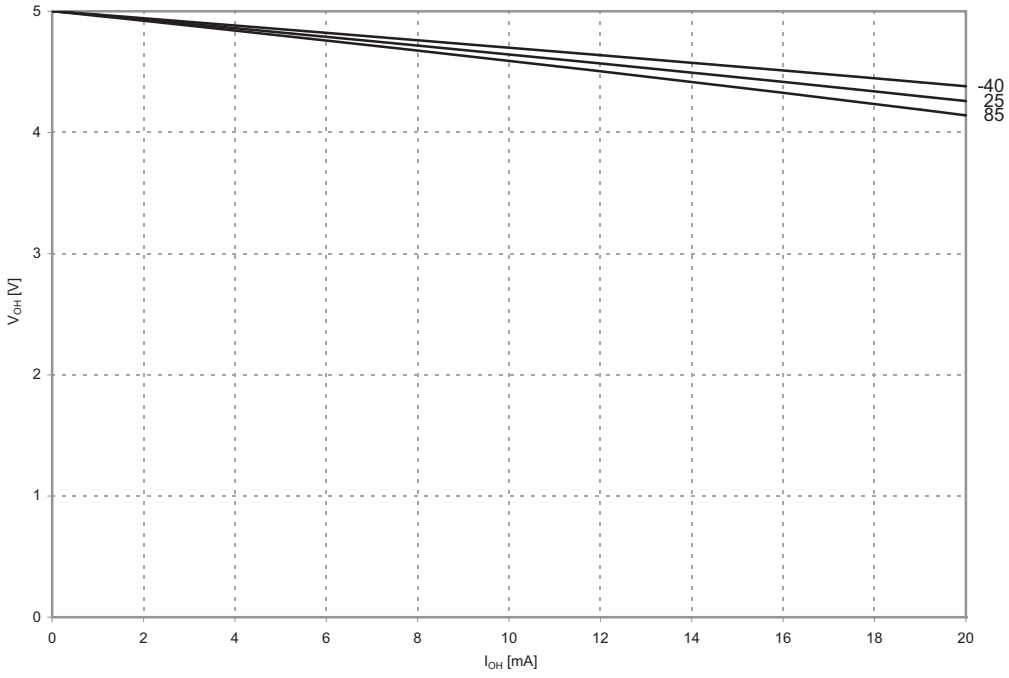


Figure 149. V_{OH}: Output Voltage vs. Source Current (I/O Pin, V_{CC} = 5V)



25.8.2 Reset Pin as I/O

Figure 150. V_{OH}: Output Voltage vs. Source Current (Reset Pin as I/O, V_{CC} = 1.8V

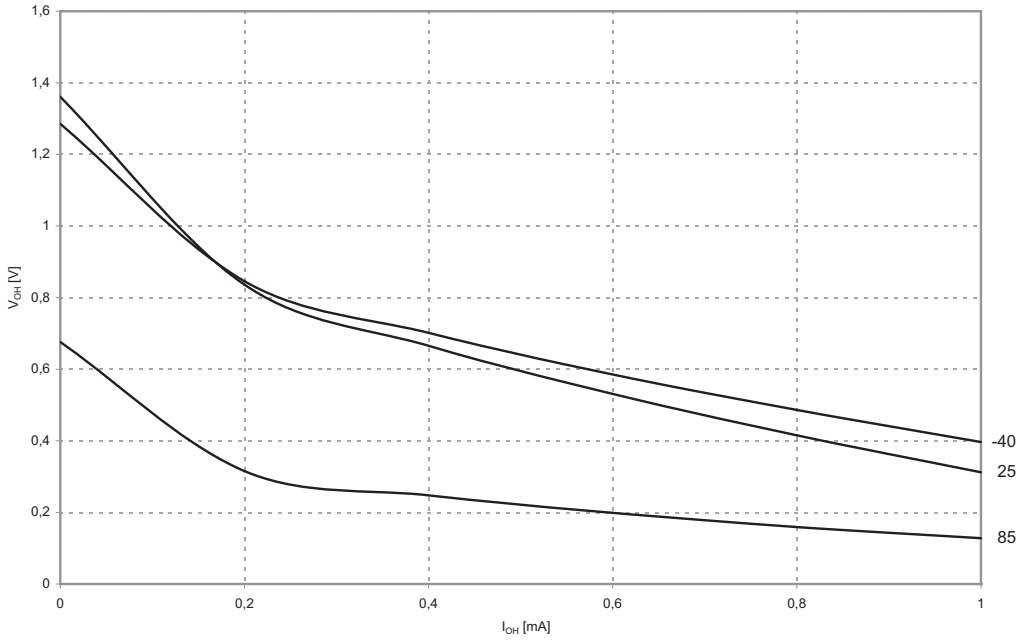


Figure 151. V_{OH}: Output Voltage vs. Source Current (Reset Pin as I/O, V_{CC} = 3V

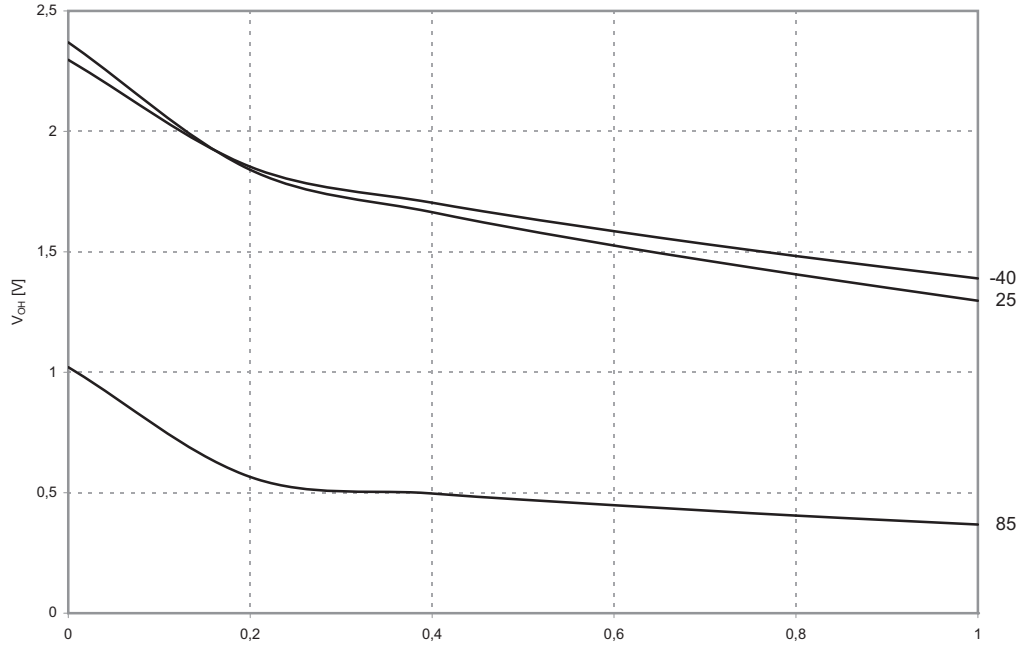
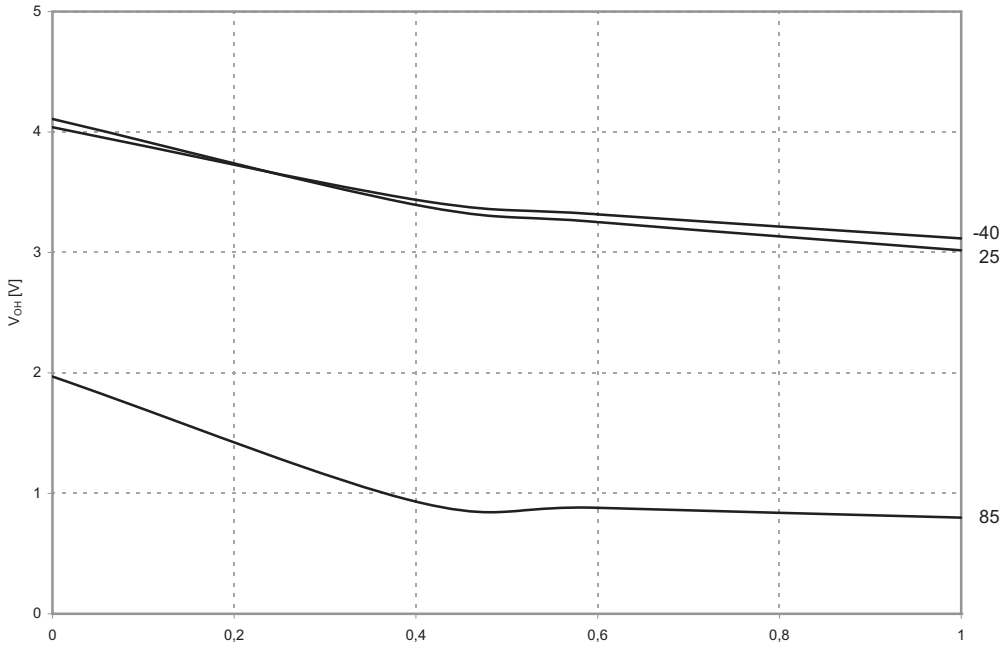


Figure 152. V_{OH}: Output Voltage vs. Source Current (Reset Pin as I/O, V_{CC} = 5V



25.9 Current Sink Capability

25.9.1 I/O Pins with Standard Sink Capability

Figure 153. V_{OL}: Output Voltage vs. Sink Current (Standard I/O Pin, V_{CC} = 1.8V)

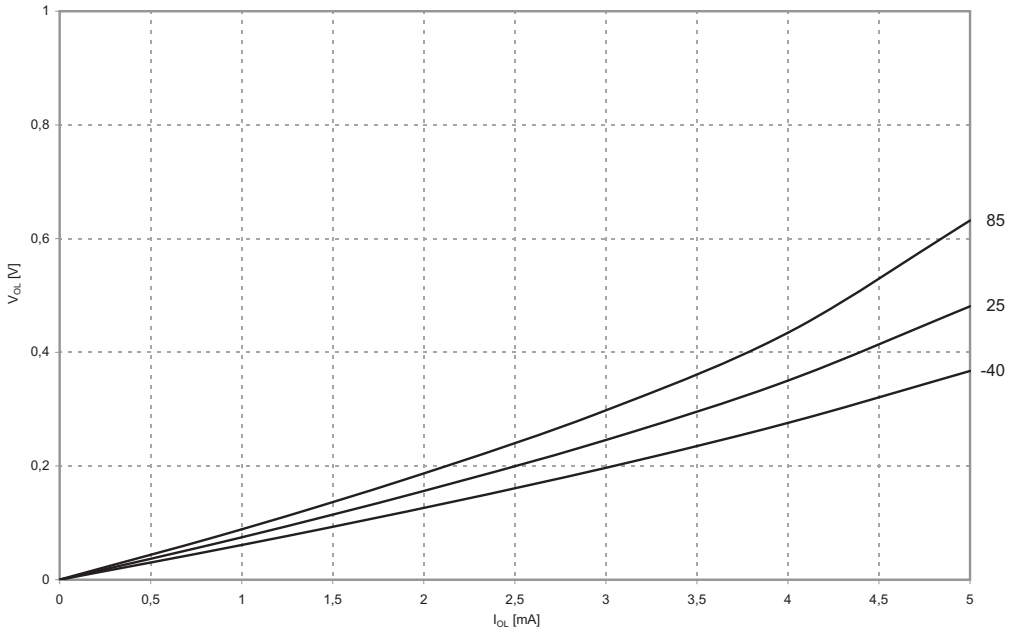


Figure 154. V_{OL}: Output Voltage vs. Sink Current (Standard I/O Pin, V_{CC} = 3V)

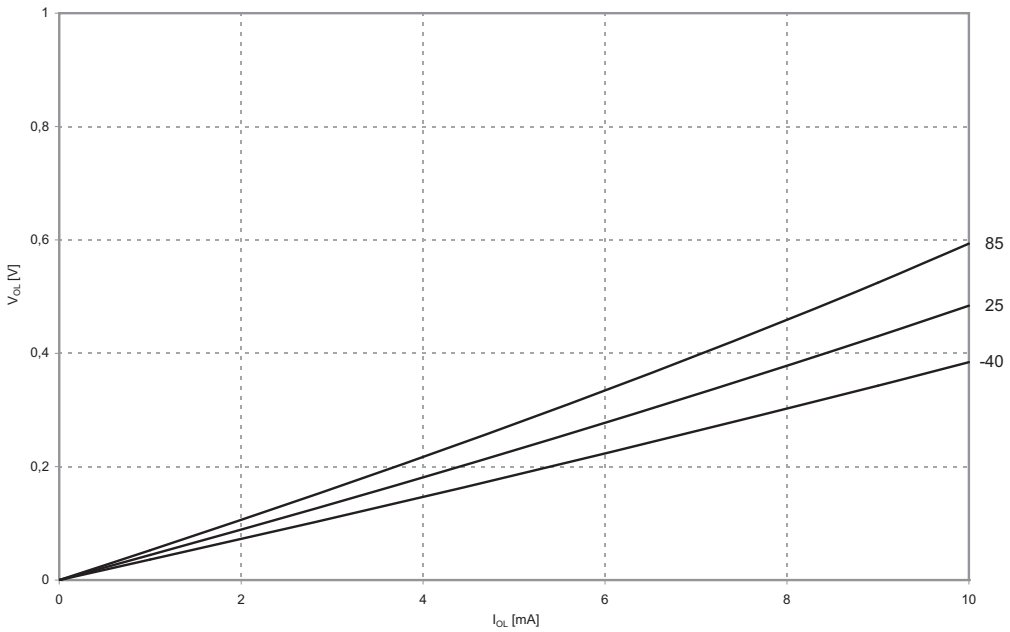
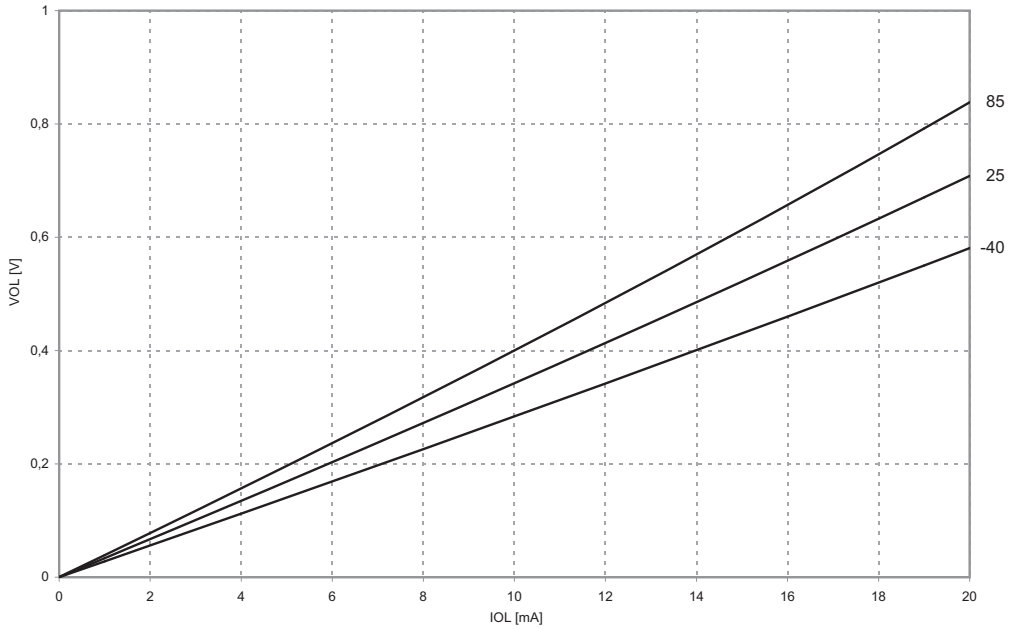


Figure 155. V_{OL}: Output Voltage vs. Sink Current (Standard I/O Pin, V_{CC} = 5V)



25.9.2 I/O Pins with High Sink Capability

Figure 156. V_{OL}: Output Voltage vs. Sink Current (High Sink I/O Pin, V_{CC} = 1.8V)

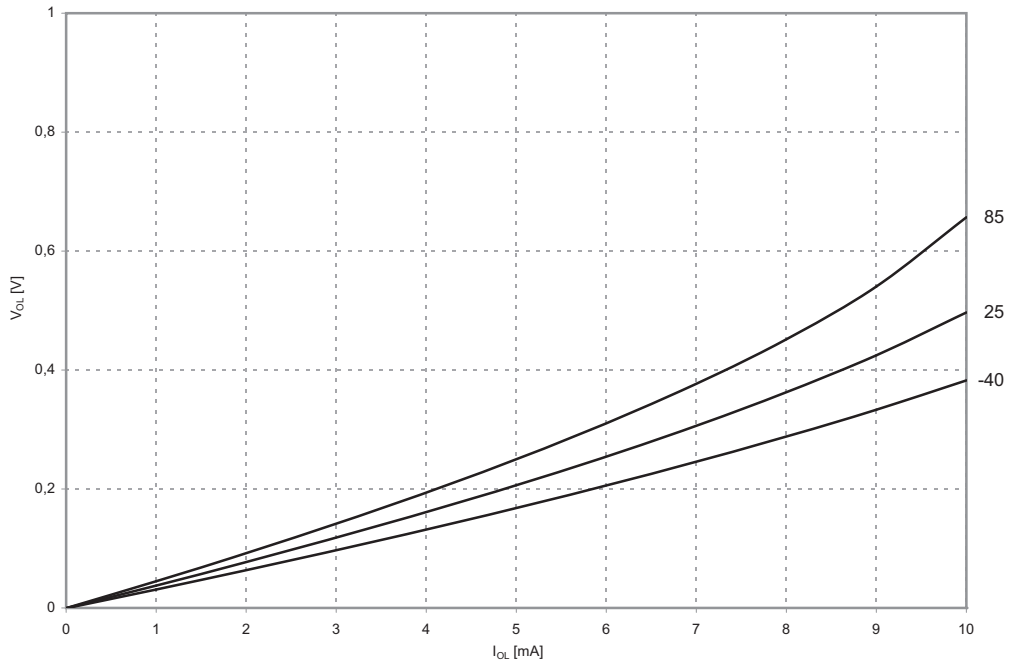


Figure 157. V_{OL}: Output Voltage vs. Sink Current (High Sink I/O Pin, V_{CC} = 3V)

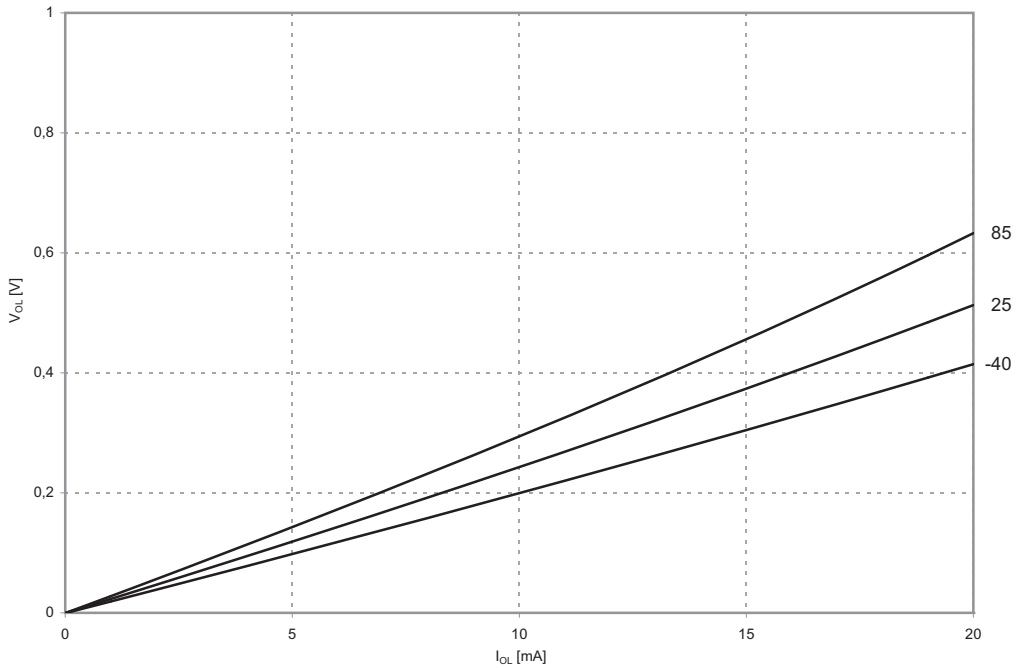
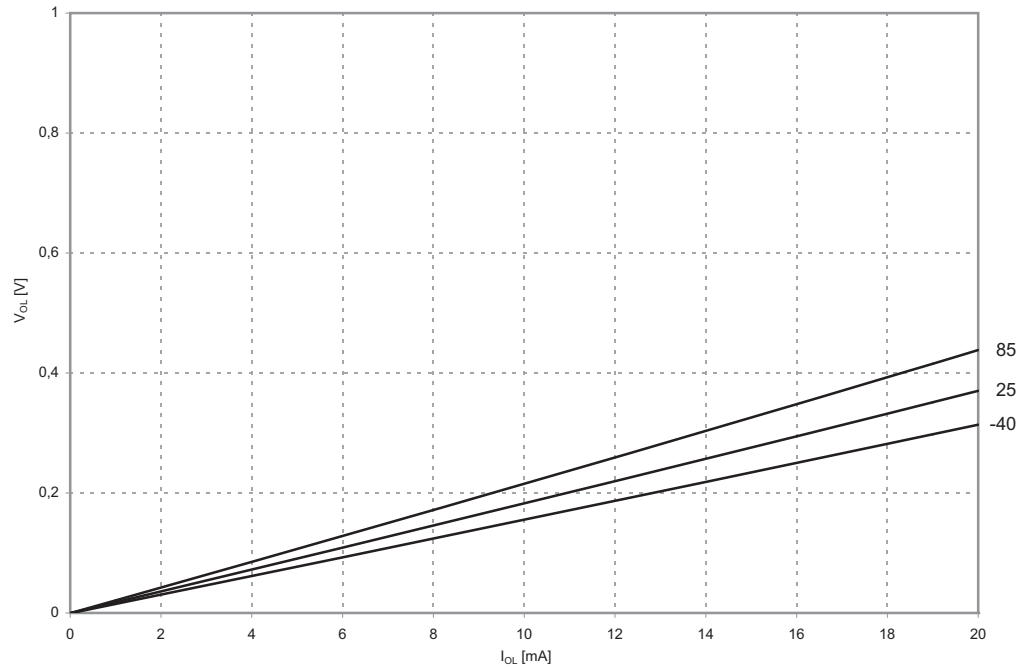


Figure 158. V_{OL}: Output Voltage vs. Sink Current (High Sink I/O Pin, V_{CC} = 5V)



25.9.3 I/O Pins with Extra High Sink Capability

Figure 159. V_{OL}: Output Voltage vs. Sink Current (Extra High Sink I/O Pin, V_{CC} = 1.8V)

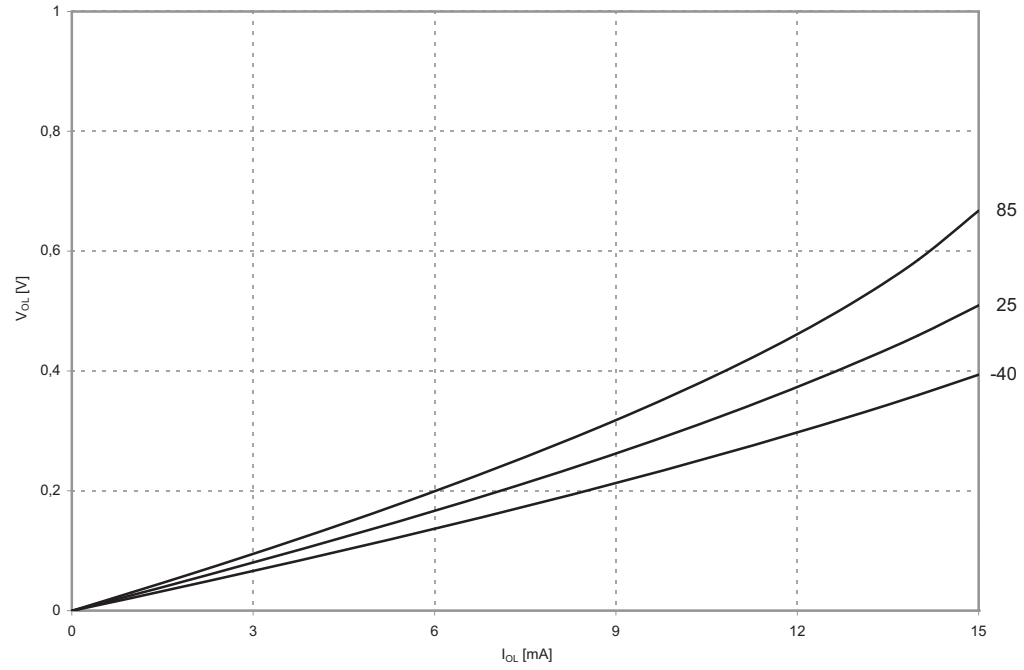


Figure 160. V_{OL}: Output Voltage vs. Sink Current (Extra High Sink I/O Pin, V_{CC} = 3V)

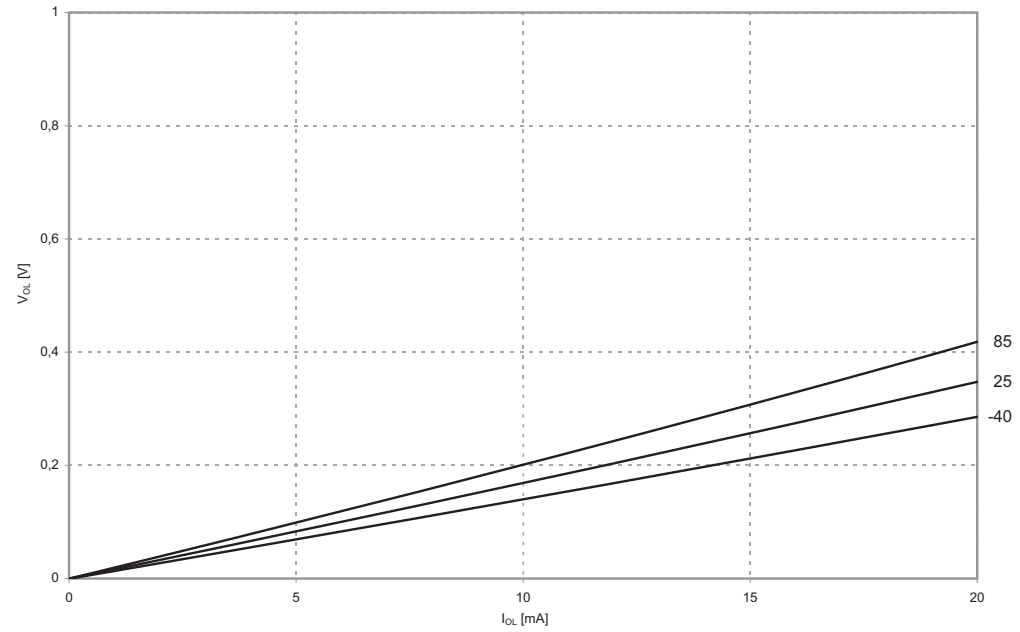
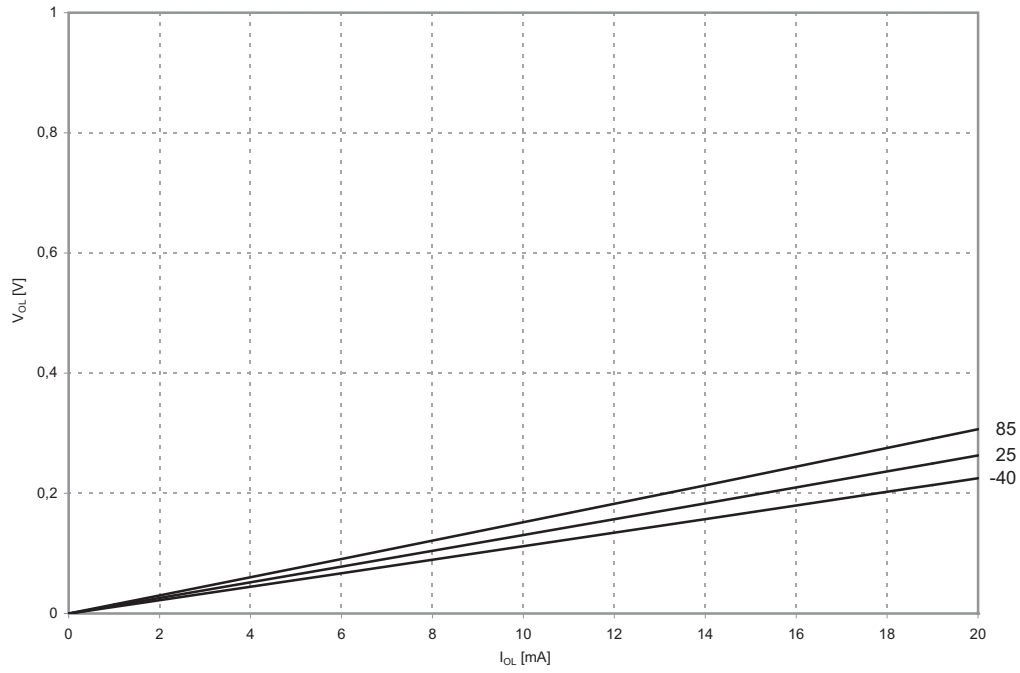


Figure 161. V_{OL}: Output Voltage vs. Sink Current (Extra High Sink I/O Pin, V_{CC} = 5V)



25.9.4 Reset Pin as I/O

Figure 162. V_{OL}: Output Voltage vs. Sink Current (Reset Pin as I/O, V_{CC} = 1.8V)

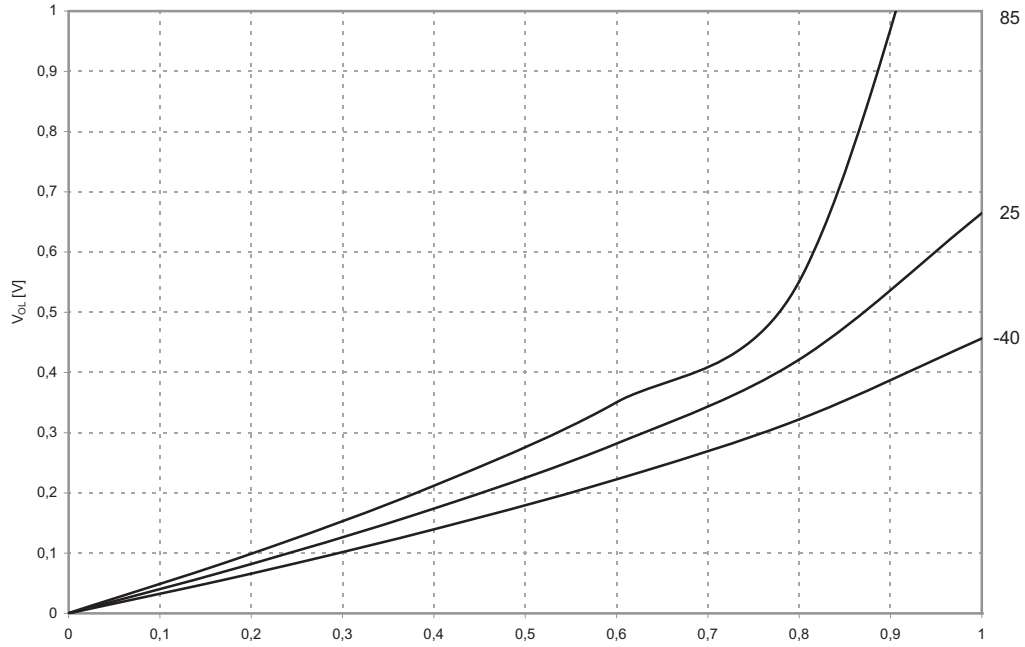


Figure 163. V_{OL}: Output Voltage vs. Sink Current (Reset Pin as I/O, V_{CC} = 3V)

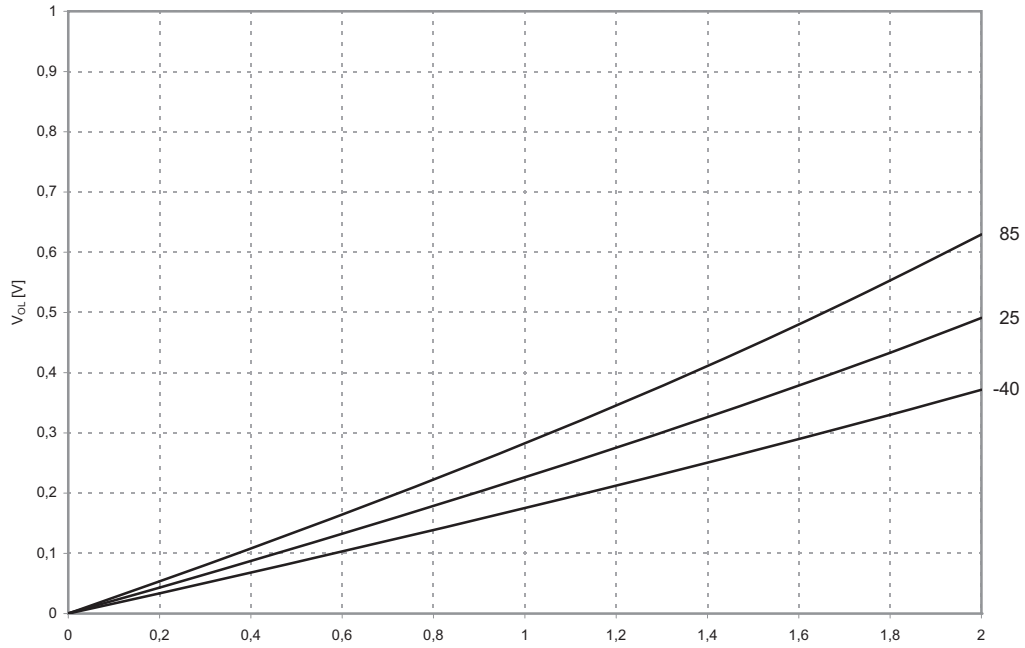
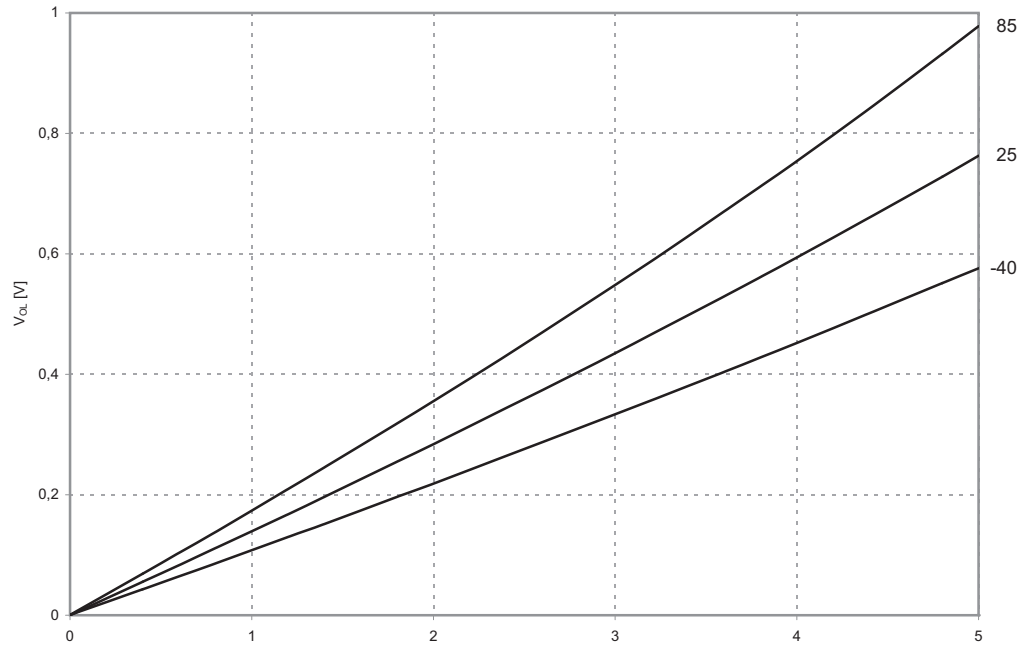


Figure 164. V_{OL}: Output Voltage vs. Sink Current (Reset Pin as I/O, V_{CC} = 5V)



25.10 BOD

Figure 165. BOD Threshold vs Temperature (BODLEVEL = 4.3V)

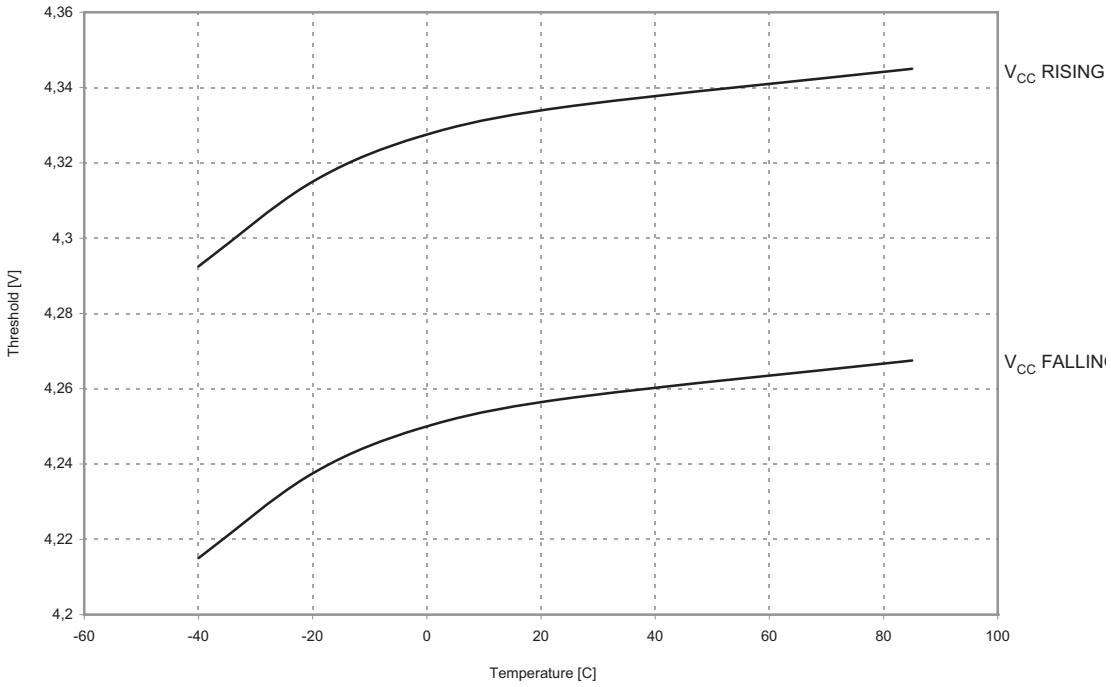


Figure 166. BOD Threshold vs Temperature (BODLEVEL = 2.7V)

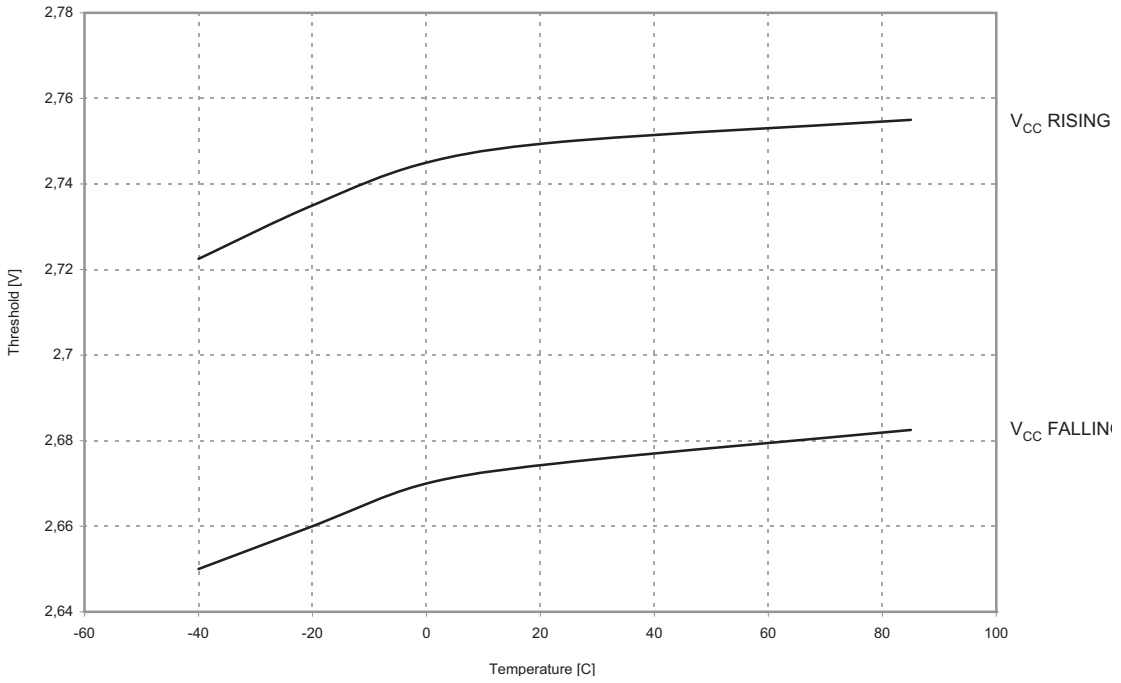


Figure 167. BOD Threshold vs Temperature (BODLEVEL = 1.8V)

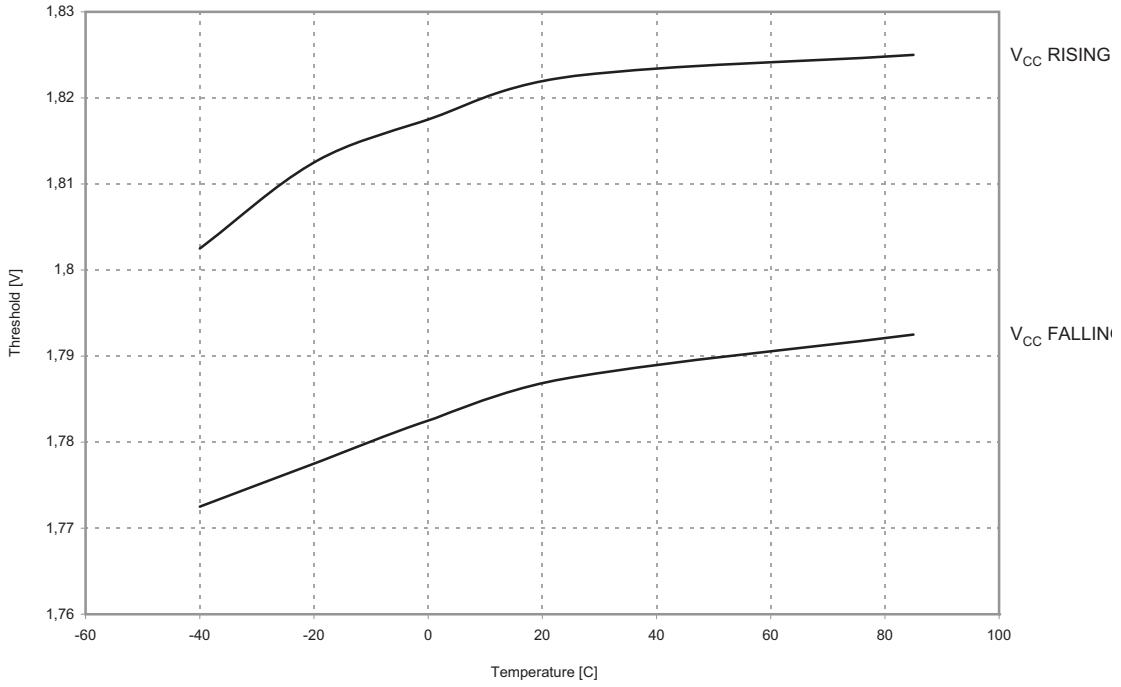


Figure 168. Sampled BOD Threshold vs Temperature (BODLEVEL = 4.3V)

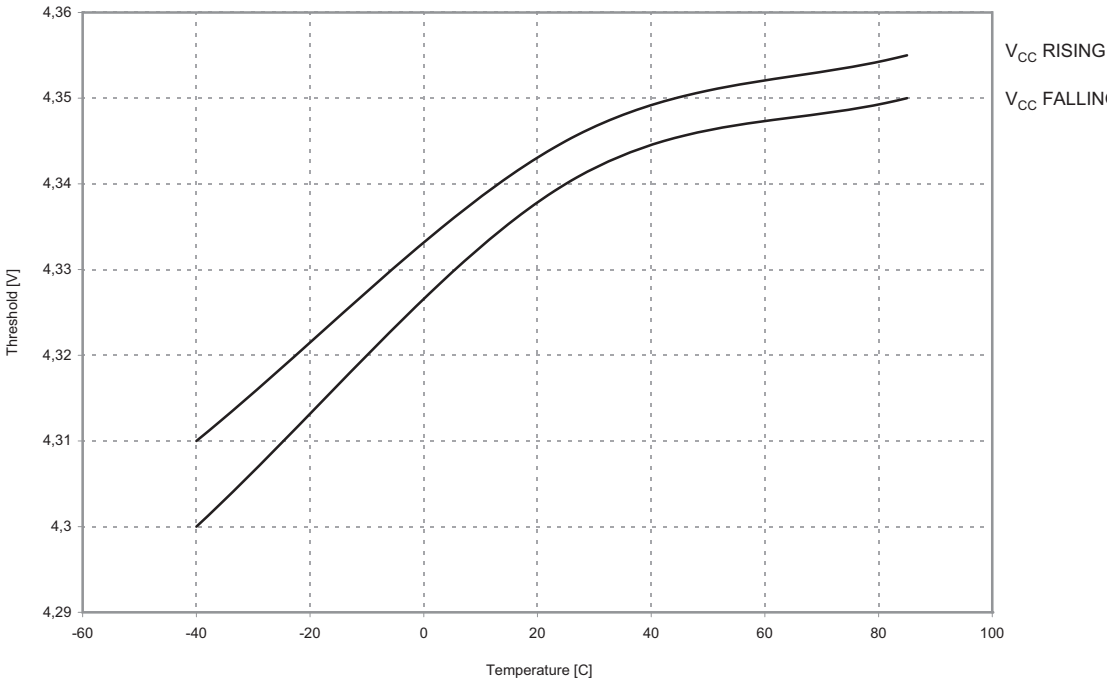


Figure 169. Sampled BOD Threshold vs Temperature (BODLEVEL = 2.7V)

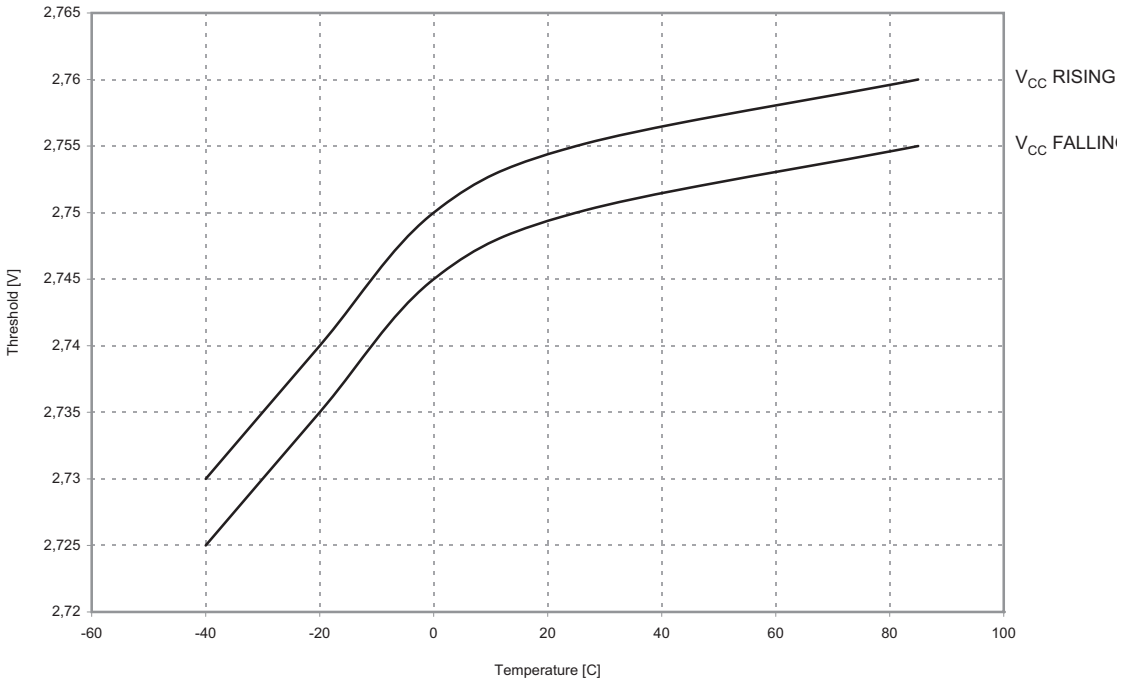
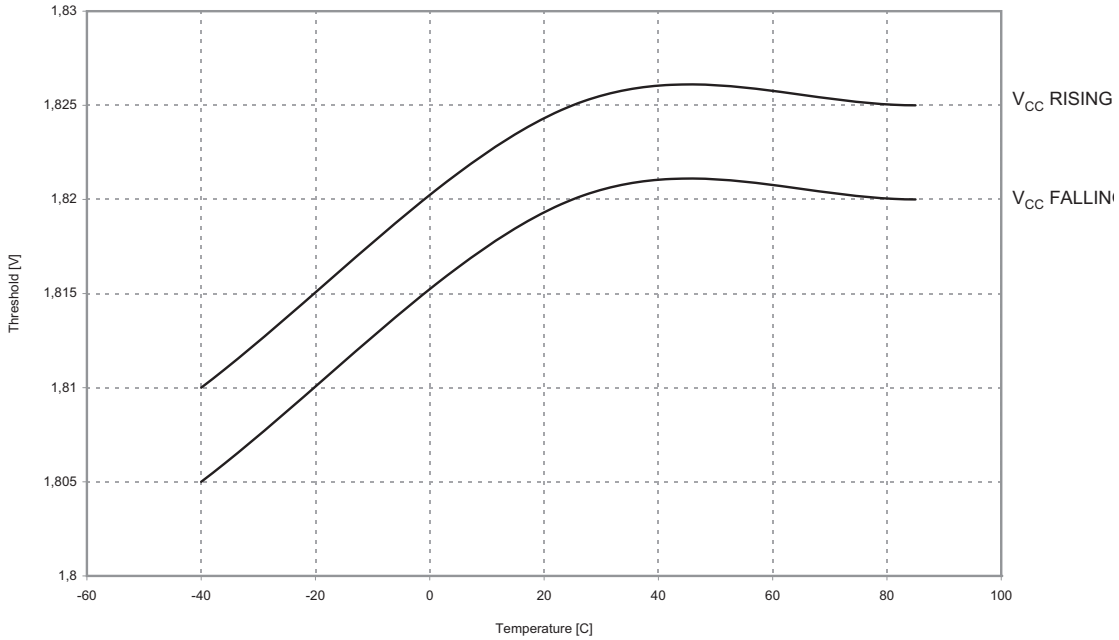


Figure 170. Sampled BOD Threshold vs Temperature (BODLEVEL = 1.8V)



25.11 Bandgap Voltage

Figure 171. Bandgap Voltage vs. Supply Voltage

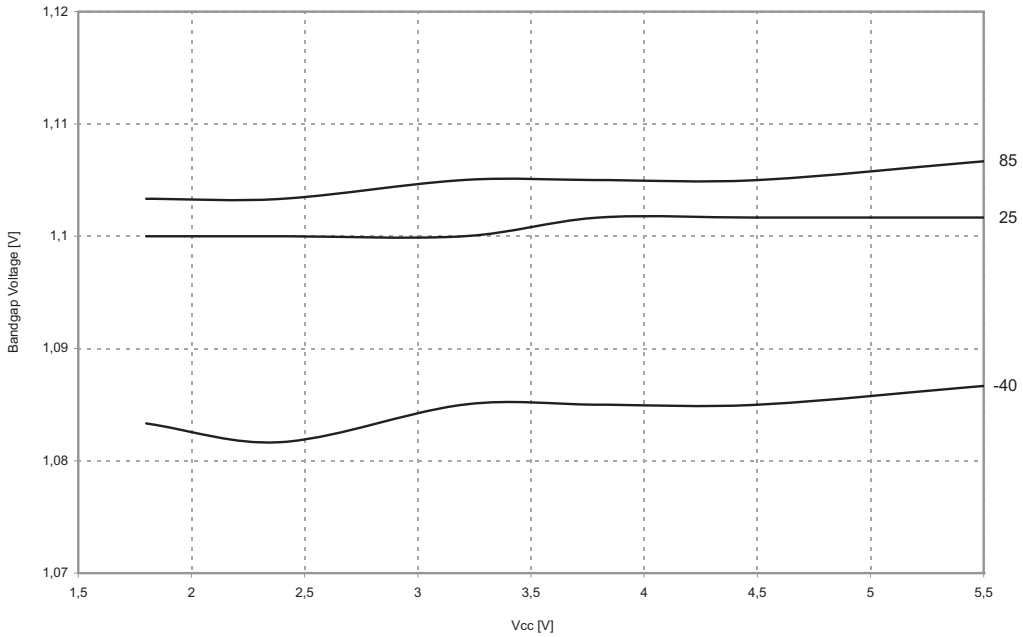
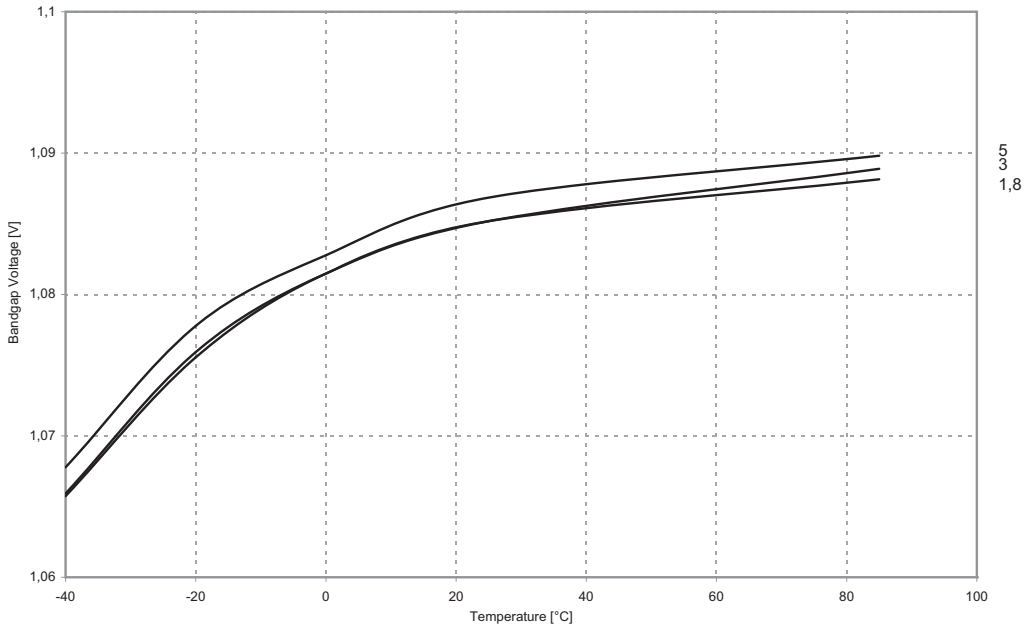


Figure 172. Bandgap Voltage vs. Temperature (V_{CC} = 3.3V)



25.12 Reset

Figure 173. POR Trigger Levels

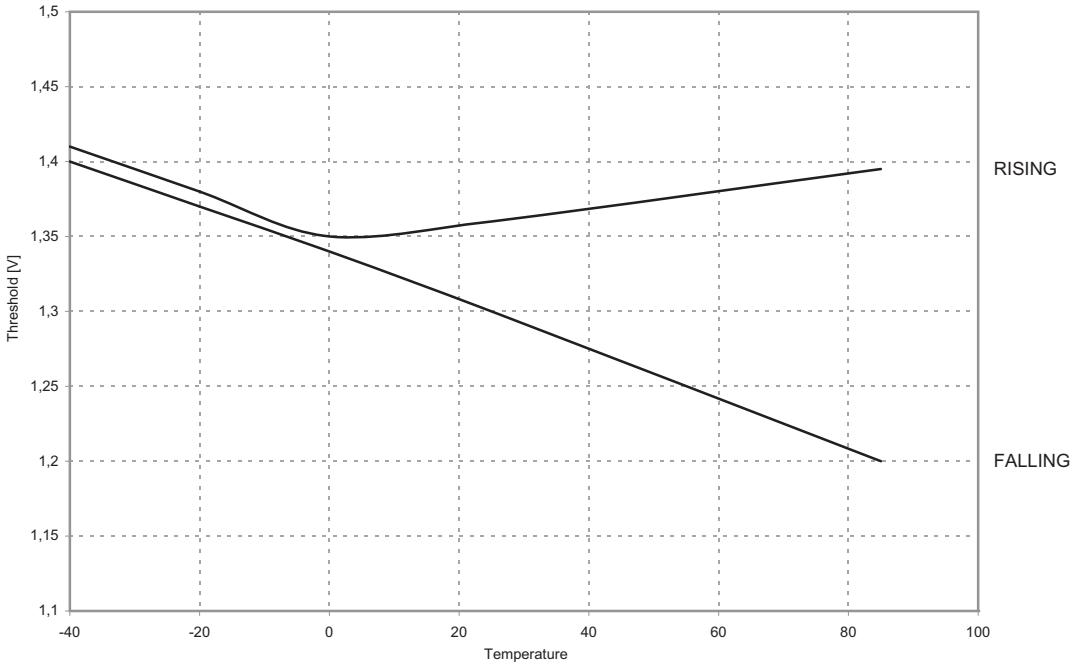
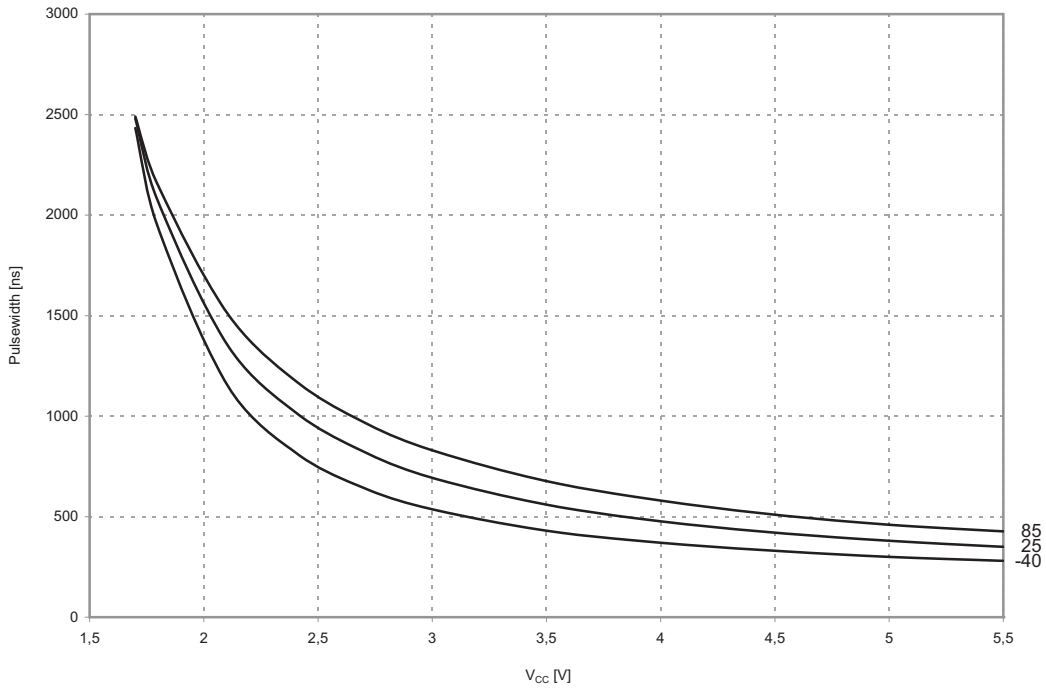


Figure 174. Minimum Reset Pulse Width vs. V_{CC}



25.13 Analog Comparator Offset

Figure 175. Analog Comparator Offset vs. V_{IN} (V_{CC} = 5V)

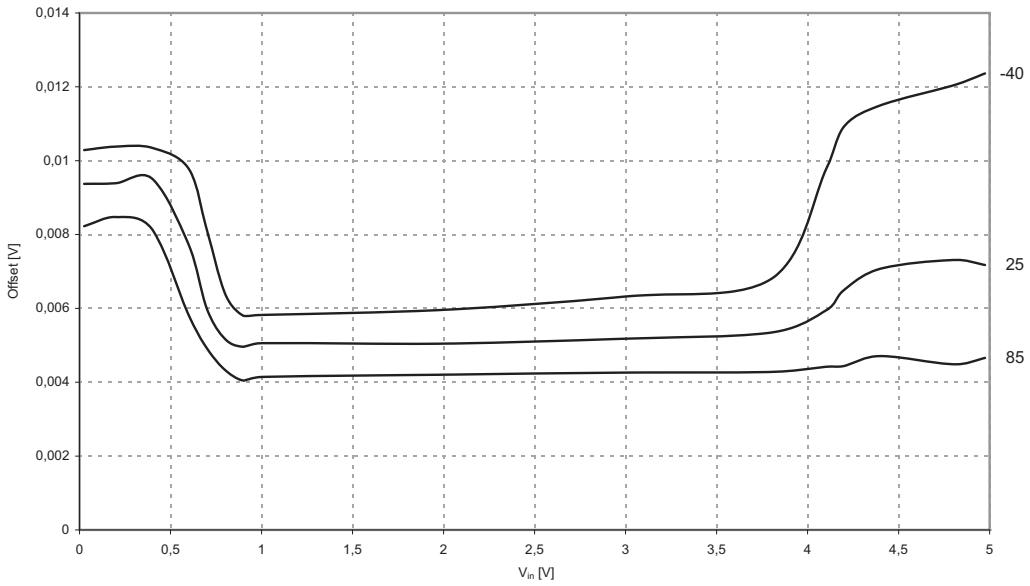


Figure 176. Analog Comparator Offset vs. V_{CC} (V_{IN} = 1.1V)

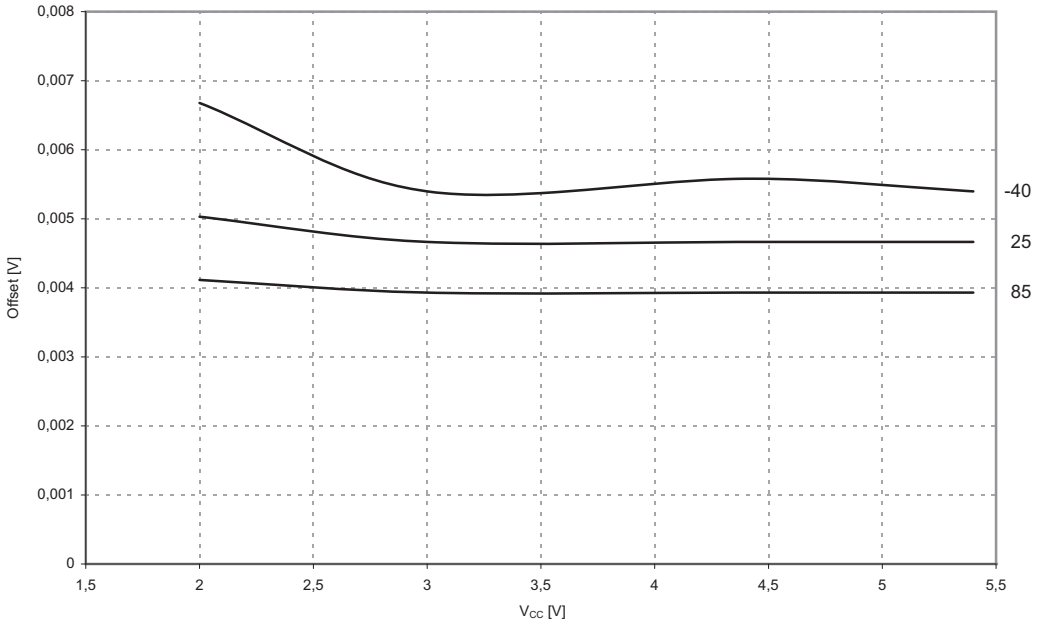
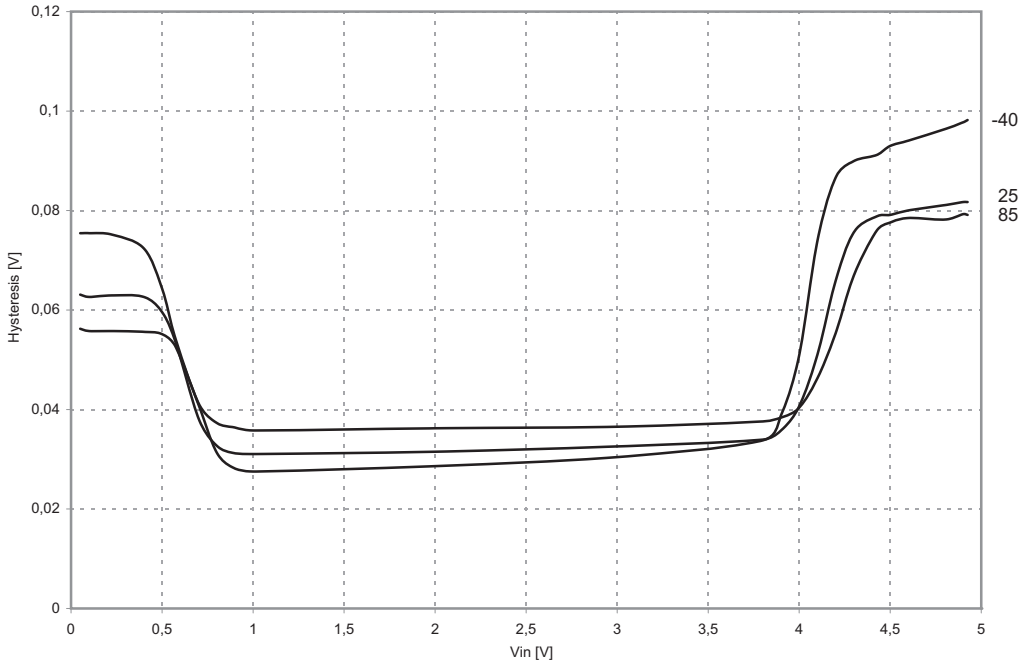


Figure 177. Analog Comparator Hysteresis vs. V_{IN} (V_{CC} = 5.0V)



25.14 Internal Oscillator Speed

25.14.1 8MHz Oscillator with CKDIV8 Enabled

Figure 178. Calibrated Oscillator Frequency vs. V_{CC} (One-point Calibration)

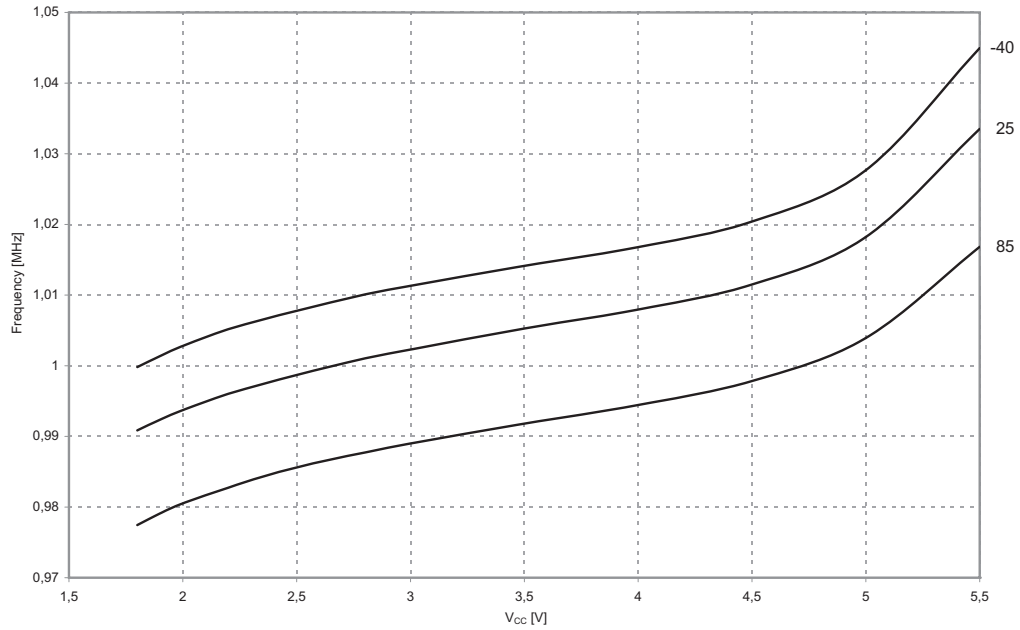


Figure 179. Calibrated Oscillator Frequency vs. V_{CC} (Two-point Calibration)

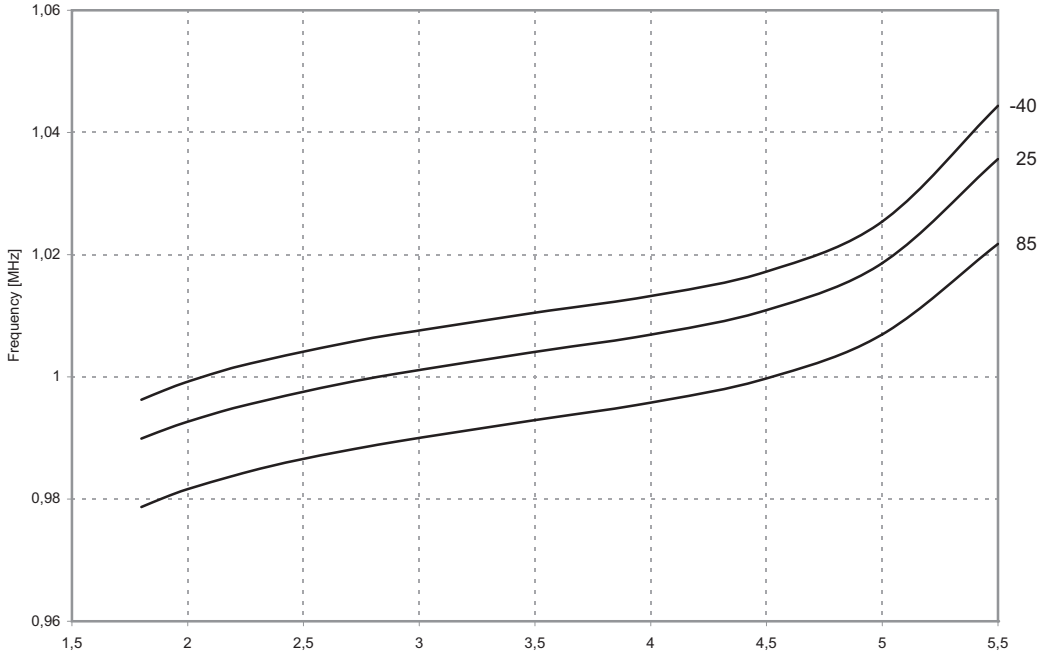


Figure 180. Calibrated Oscillator Frequency vs. Temperature (One-point Calibration)

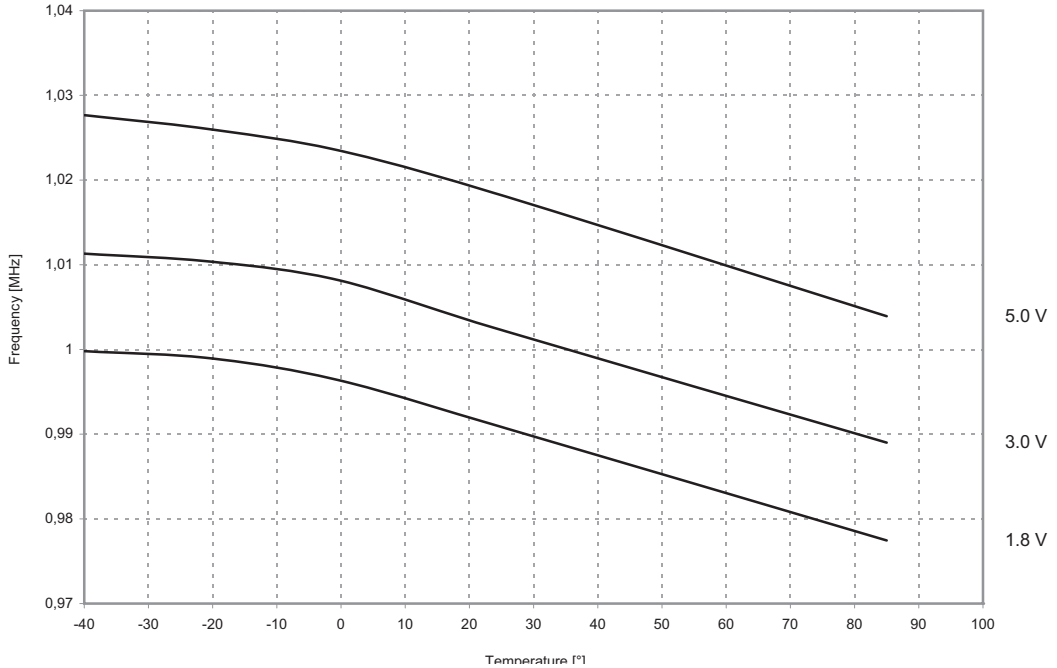


Figure 181. Calibrated Oscillator Frequency vs. Temperature (Two-point Calibration)

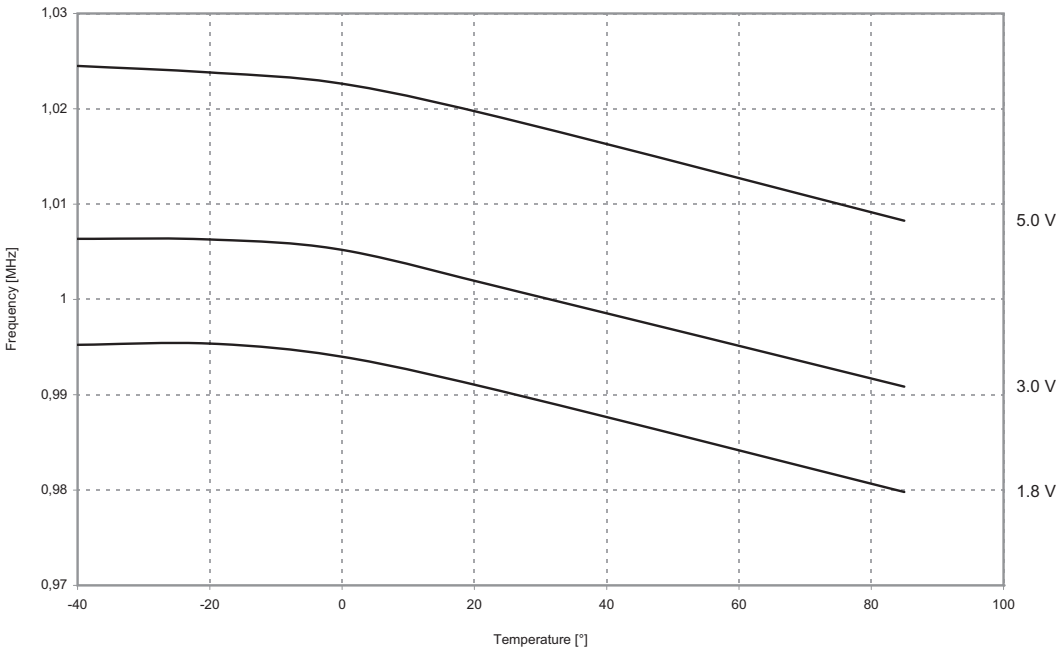
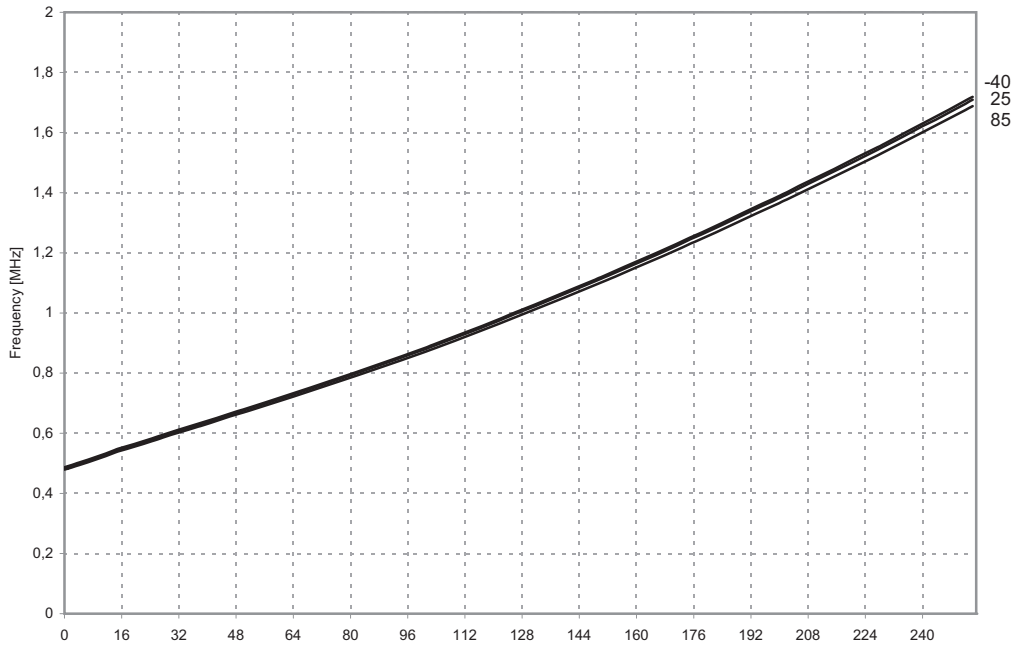


Figure 182. Calibrated Oscillator Frequency vs. OSCCAL0 Value



25.14.2 32kHz ULP Oscillator

Figure 183. ULP Oscillator Frequency vs. V_{CC}

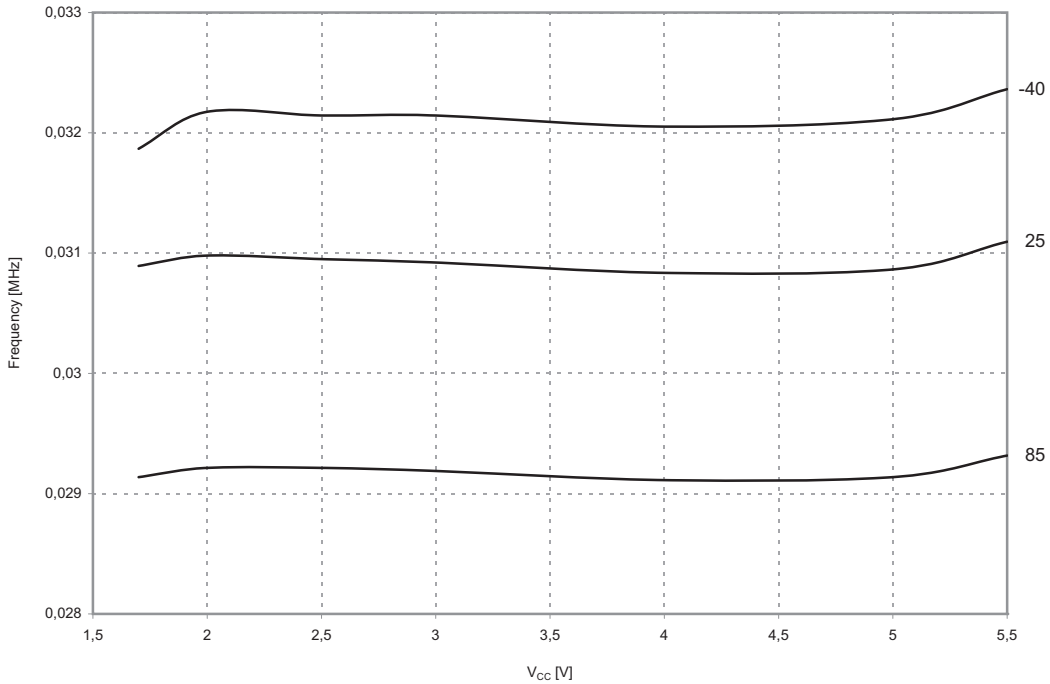


Figure 184. ULP Oscillator Frequency vs. Temperature

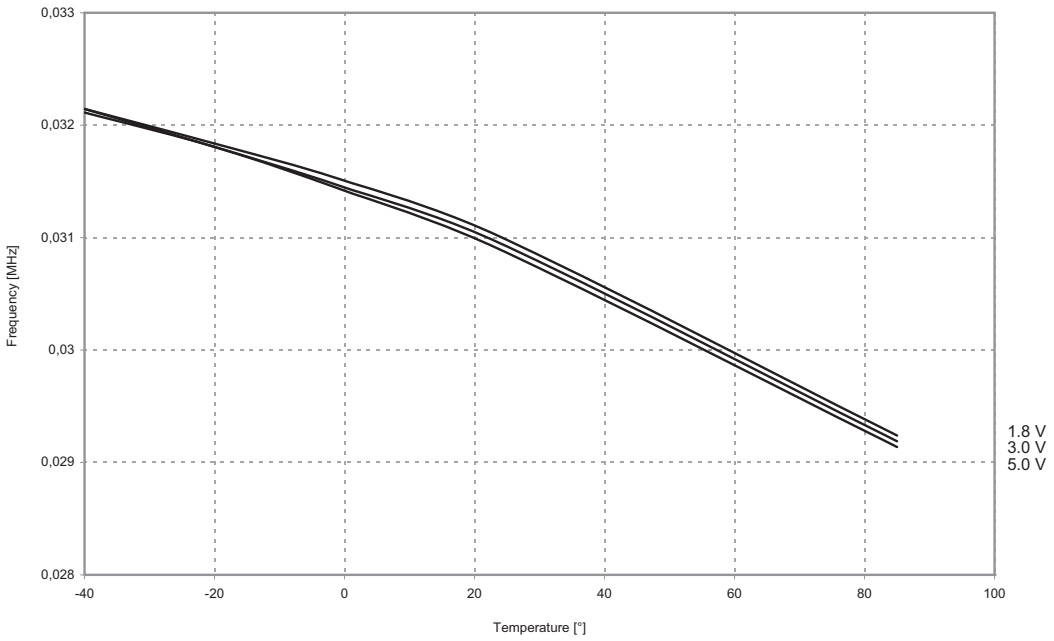
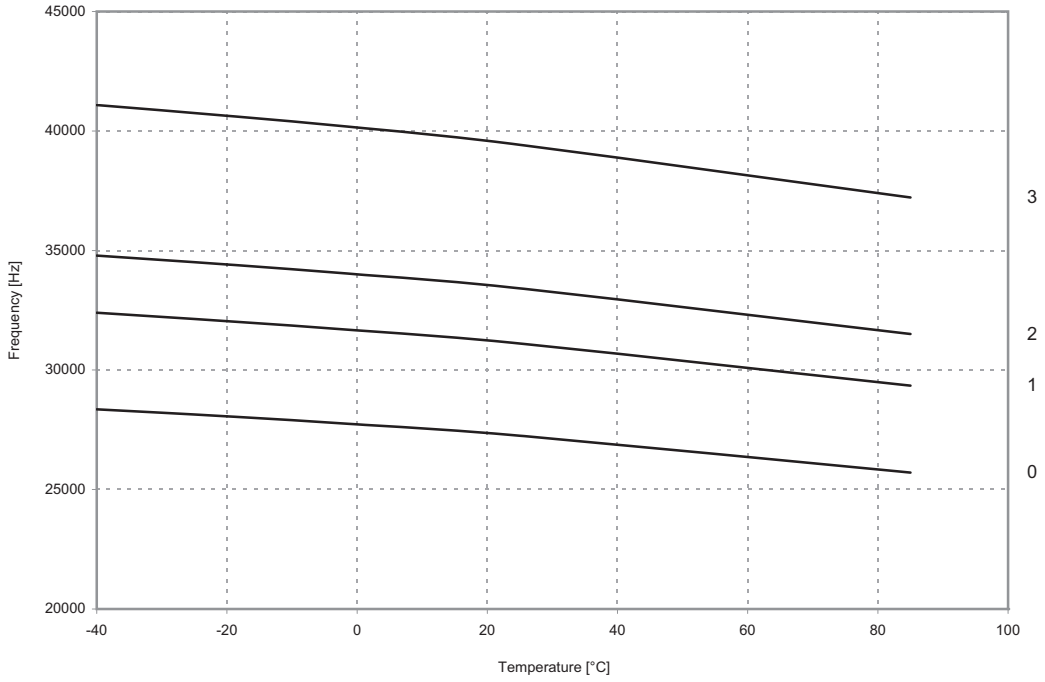


Figure 185. ULP Oscillator Frequency vs. OSCCAL1 Value



26. Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page(s)
(0xFF)	Reserved	–	–	–	–	–	–	–	–	
(0xFE)	Reserved	–	–	–	–	–	–	–	–	
(0xFD)	Reserved	–	–	–	–	–	–	–	–	
(0xFC)	Reserved	–	–	–	–	–	–	–	–	
(0xFB)	Reserved	–	–	–	–	–	–	–	–	
(0xFA)	Reserved	–	–	–	–	–	–	–	–	
(0xF9)	Reserved	–	–	–	–	–	–	–	–	
(0xF8)	Reserved	–	–	–	–	–	–	–	–	
(0xF7)	Reserved	–	–	–	–	–	–	–	–	
(0xF6)	Reserved	–	–	–	–	–	–	–	–	
(0xF5)	Reserved	–	–	–	–	–	–	–	–	
(0xF4)	Reserved	–	–	–	–	–	–	–	–	
(0xF3)	Reserved	–	–	–	–	–	–	–	–	
(0xF2)	Reserved	–	–	–	–	–	–	–	–	
(0xF1)	OSCTCAL0B	Oscillator Temperature Compensation Register B								Page 33
(0xF0)	OSCTCAL0A	Oscillator Temperature Compensation Register A								Page 33
(0xEF)	Reserved	–	–	–	–	–	–	–	–	
(0xEE)	Reserved	–	–	–	–	–	–	–	–	
(0xED)	Reserved	–	–	–	–	–	–	–	–	
(0xEC)	Reserved	–	–	–	–	–	–	–	–	
(0xEB)	Reserved	–	–	–	–	–	–	–	–	
(0xEA)	Reserved	–	–	–	–	–	–	–	–	
(0xE9)	TOCPMSA1	TOCC7S1	TOCC7S0	TOCC6S1	TOCC6S0	TOCC5S1	TOCC5S0	TOCC4S1	TOCC4S0	Page 127
(0xE8)	TOCPMSA0	TOCC3S1	TOCC3S0	TOCC2S1	TOCC2S0	TOCC1S1	TOCC1S0	TOCC0S1	TOCC0S0	Page 127
(0xE7)	Reserved	–	–	–	–	–	–	–	–	
(0xE6)	Reserved	–	–	–	–	–	–	–	–	
(0xE5)	Reserved	–	–	–	–	–	–	–	–	
(0xE4)	Reserved	–	–	–	–	–	–	–	–	
(0xE3)	Reserved	–	–	–	–	–	–	–	–	
(0xE2)	TOCPMCOE	TOCC7OE	TOCC6OE	TOCC5OE	TOCC4OE	TOCC3OE	TOCC2OE	TOCC1OE	TOCC0OE	Page 128
(0xE1)	Reserved	–	–	–	–	–	–	–	–	
(0xE0)	Reserved	–	–	–	–	–	–	–	–	
(0xDF)	DIDR3	–	–	–	–	ADC27D	ADC26D	ADC25D	ADC24D	Page 154
(0xDE)	DIDR2	ADC23D	ADC22D	ADC21D	ADC20D	ADC19D	ADC18D	ADC17D	ADC16D	Page 154
(0xDD)	Reserved	–	–	–	–	–	–	–	–	
(0xDC)	Reserved	–	–	–	–	–	–	–	–	
(0xDB)	Reserved	–	–	–	–	–	–	–	–	
(0xDA)	Reserved	–	–	–	–	–	–	–	–	
(0xD9)	Reserved	–	–	–	–	–	–	–	–	
(0xD8)	Reserved	–	–	–	–	–	–	–	–	
(0xD7)	Reserved	–	–	–	–	–	–	–	–	
(0xD6)	Reserved	–	–	–	–	–	–	–	–	
(0xD5)	Reserved	–	–	–	–	–	–	–	–	
(0xD4)	Reserved	–	–	–	–	–	–	–	–	
(0xD3)	Reserved	–	–	–	–	–	–	–	–	
(0xD2)	Reserved	–	–	–	–	–	–	–	–	
(0xD1)	Reserved	–	–	–	–	–	–	–	–	
(0xD0)	Reserved	–	–	–	–	–	–	–	–	
(0xCF)	Reserved	–	–	–	–	–	–	–	–	
(0xCE)	Reserved	–	–	–	–	–	–	–	–	
(0xCD)	Reserved	–	–	–	–	–	–	–	–	
(0xCC)	Reserved	–	–	–	–	–	–	–	–	
(0xCB)	Reserved	–	–	–	–	–	–	–	–	
(0xCA)	Reserved	–	–	–	–	–	–	–	–	
(0xC9)	Reserved	–	–	–	–	–	–	–	–	
(0xC8)	Reserved	–	–	–	–	–	–	–	–	
(0xC7)	Reserved	–	–	–	–	–	–	–	–	
(0xC6)	UDR	USART Data Register								Pages 184, 195
(0xC5)	UBRRH	–	–	–	–	USART Baud Register High				Page 189, 198
(0xC4)	UBRRL	USART Baud Rate Register Low								Page 189, 198
(0xC3)	UCSRD	RXSIE	RXS	SFDE	–	–	–	–	–	Page 188
(0xC2)	UCSRC	UMSEL1	UMSEL0	UPM1	UPM0	USBS	UCSZ1/UDO	UCSZ0/UCP	UCPOL	Page 186, 197
(0xC1)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	Page 185, 196
(0xC0)	UCSRA	RXC	TXC	UDRE	FE	DOR	UPE	U2X	MPCM	Page 184, 196
(0xBF)	Reserved	–	–	–	–	–	–	–	–	
(0xBE)	Reserved	–	–	–	–	–	–	–	–	

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page(s)
(0xBD)	TWSD	TWI Slave Data Register								Page 211
(0xBC)	TWSA	TWI Slave Address Register								Page 210
(0xBB)	TWSAM	TWI Slave Address Mask Register							TWAE	Page 211
(0xBA)	TWSSRA	TWDIF	TWASIF	TWCH	TWRA	TWC	TWBE	TWDIR	TWAS	Page 209
(0xB9)	TWSCRB	–	–	–	–	–	TWAA	TWCMD1	TWCMD0	Page 208
(0xB8)	TWSCRA	TWSHE	–	TWDIE	TWASIE	TWEN	TWSIE	TWPME	TWSME	Page 207
(0xB7)	Reserved	–	–	–	–	–	–	–	–	
(0xB6)	Reserved	–	–	–	–	–	–	–	–	
(0xB5)	Reserved	–	–	–	–	–	–	–	–	
(0xB4)	Reserved	–	–	–	–	–	–	–	–	
(0xB3)	Reserved	–	–	–	–	–	–	–	–	
(0xB2)	Reserved	–	–	–	–	–	–	–	–	
(0xB1)	Reserved	–	–	–	–	–	–	–	–	
(0xB0)	Reserved	–	–	–	–	–	–	–	–	
(0xAF)	Reserved	–	–	–	–	–	–	–	–	
(0xAE)	Reserved	–	–	–	–	–	–	–	–	
(0xAD)	Reserved	–	–	–	–	–	–	–	–	
(0xAC)	Reserved	–	–	–	–	–	–	–	–	
(0xAB)	Reserved	–	–	–	–	–	–	–	–	
(0xAA)	Reserved	–	–	–	–	–	–	–	–	
(0xA9)	Reserved	–	–	–	–	–	–	–	–	
(0xA8)	Reserved	–	–	–	–	–	–	–	–	
(0xA7)	Reserved	–	–	–	–	–	–	–	–	
(0xA6)	Reserved	–	–	–	–	–	–	–	–	
(0xA5)	Reserved	–	–	–	–	–	–	–	–	
(0xA4)	Reserved	–	–	–	–	–	–	–	–	
(0xA3)	Reserved	–	–	–	–	–	–	–	–	
(0xA2)	Reserved	–	–	–	–	–	–	–	–	
(0xA1)	Reserved	–	–	–	–	–	–	–	–	
(0xA0)	Reserved	–	–	–	–	–	–	–	–	
(0x9F)	Reserved	–	–	–	–	–	–	–	–	
(0x9E)	Reserved	–	–	–	–	–	–	–	–	
(0x9D)	Reserved	–	–	–	–	–	–	–	–	
(0x9C)	Reserved	–	–	–	–	–	–	–	–	
(0x9B)	Reserved	–	–	–	–	–	–	–	–	
(0x9A)	Reserved	–	–	–	–	–	–	–	–	
(0x99)	Reserved	–	–	–	–	–	–	–	–	
(0x98)	Reserved	–	–	–	–	–	–	–	–	
(0x97)	Reserved	–	–	–	–	–	–	–	–	
(0x96)	Reserved	–	–	–	–	–	–	–	–	
(0x95)	Reserved	–	–	–	–	–	–	–	–	
(0x94)	Reserved	–	–	–	–	–	–	–	–	
(0x93)	Reserved	–	–	–	–	–	–	–	–	
(0x92)	Reserved	–	–	–	–	–	–	–	–	
(0x91)	Reserved	–	–	–	–	–	–	–	–	
(0x90)	Reserved	–	–	–	–	–	–	–	–	
(0x8F)	Reserved	–	–	–	–	–	–	–	–	
(0x8E)	Reserved	–	–	–	–	–	–	–	–	
(0x8D)	Reserved	–	–	–	–	–	–	–	–	
(0x8C)	Reserved	–	–	–	–	–	–	–	–	
(0x8B)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte								Page 128
(0x8A)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte								Page 128
(0x89)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte								Page 128
(0x88)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte								Page 128
(0x87)	ICR1H	Timer/Counter1 – Input Capture Register High Byte								Page 129
(0x86)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte								Page 129
(0x85)	TCNT1H	Timer/Counter1 – Counter Register High Byte								Page 128
(0x84)	TCNT1L	Timer/Counter1 – Counter Register Low Byte								Page 128
(0x83)	Reserved	–	–	–	–	–	–	–	–	
(0x82)	TCCR1C	FOC1A	FOC1B	–	–	–	–	–	–	Page 127
(0x81)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	Page 125
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	Page 123
(0x7F)	DIDR1	ADC15D	ADC14D	ADC13D	ADC12D	ADC11D	ADC10D	ADC9D	ADC8D	Page 154
(0x7E)	DIDR0	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	Pages 136, 154
(0x7D)	ADMUXB	–	–	REFS	–	–	–	–	MUX5	Page 150
(0x7C)	ADMUXA	–	–	–	MUX4	MUX3	MUX2	MUX1	MUX0	Page 149
(0x7B)	ADCSRB	–	–	–	–	ADLAR	ADTS2	ADTS1	ADTS0	Page 153
(0x7A)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	Page 151

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page(s)
(0x79)	ADCH	ADC – Conversion Result High Byte								Page 151
(0x78)	ADCL	ADC – Conversion Result Low Byte								Page 151
(0x77)	Reserved	–	–	–	–	–	–	–	–	
(0x76)	Reserved	–	–	–	–	–	–	–	–	
(0x75)	Reserved	–	–	–	–	–	–	–	–	
(0x74)	Reserved	–	–	–	–	–	–	–	–	
(0x73)	PCMSK3	–	–	–	–	PCINT27	PCINT26	PCINT25	PCINT24	Page 54
(0x72)	Reserved	–	–	–	–	–	–	–	–	
(0x71)	Reserved	–	–	–	–	–	–	–	–	
(0x70)	Reserved	–	–	–	–	–	–	–	–	
(0x6F)	TIMSK1	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	Page 129
(0x6E)	TIMSK0	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	Page 102
(0x6D)	PCMSK2	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	Page 54
(0x6C)	PCMSK1	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	Page 54
(0x6B)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	Page 55
(0x6A)	Reserved	–	–	–	–	–	–	–	–	
(0x69)	EICRA	–	–	–	–	ISC11	ISC10	ISC01	ISC00	Page 55
(0x68)	PCICR	–	–	–	–	PCIE3	PCIE2	PCIE1	PCIE0	Page 56
(0x67)	OSCCAL1	–	–	–	–	–	–	CAL11	CAL10	Page 33
(0x66)	OSCCAL0	CAL07	CAL06	CAL05	CAL04	CAL03	CAL02	CAL01	CAL00	Page 32
(0x65)	Reserved	–	–	–	–	–	–	–	–	
(0x64)	PRR	PRTWI	–	PRTIM0	–	PRTIM1	PRSPI	PRUSART0	PRADC	Page 37
(0x63)	Reserved	–	–	–	–	–	–	–	–	
(0x62)	Reserved	–	–	–	–	–	–	–	–	
(0x61)	CLKPR	–	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0	Page 31
(0x60)	WDTCSR	WDIF	WDIE	WDP3	–	WDE	WDP2	WDP1	WDP0	Page 46
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	Page 15
0x3E (0x5E)	SPH	–	–	–	–	–	–	SP9	SP8	Page 14
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	Page 14
0x3C (0x5C)	Reserved	–	–	–	–	–	–	–	–	
0x3B (0x5B)	Reserved	–	–	–	–	–	–	–	–	
0x3A (0x5A)	Reserved	–	–	–	–	–	–	–	–	
0x39 (0x59)	Reserved	–	–	–	–	–	–	–	–	
0x38 (0x58)	Reserved	–	–	–	–	–	–	–	–	
0x37 (0x57)	SPMCSR	SPMIE	RWWSB	RSIG	RWWSRE	RWFLB	PGWRT	PGERS	SPMEN	Page 223
0x36 (0x56)	CCP	CPU Change Protection Register								Page 14
0x35 (0x55)	MCUCR	–	–	–	–	–	–	IVSEL	–	Page 53
0x34 (0x54)	MCUSR	–	–	–	–	WDRF	BORF	EXTRF	PORF	Page 45
0x33 (0x53)	SMCR	–	–	–	–	–	SM1	SM0	SE	Page 37
0x32 (0x52)	Reserved	–	–	–	–	–	–	–	–	
0x31 (0x51)	DWDR	debugWire Data Register								Page 213
0x30 (0x50)	ACSRA	ACD	ACPMUX2	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	Page 134
0x2F (0x4F)	ACSRB	HSEL	HLEV	ACLP		ACNMUX1	ACNMUX0	ACPMUX1	ACPMUX0	Page 135
0x2E (0x4E)	SPDR	SPI Data Register								Page 163
0x2D (0x4D)	SPSR	SPIF	WCOL	–	–	–	–	SPI2X		Page 162
0x2C (0x4C)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	Page 161
0x2B (0x4B)	GPIOR2	General Purpose I/O Register 2								Page 25
0x2A (0x4A)	GPIOR1	General Purpose I/O Register 1								Page 25
0x29 (0x49)	Reserved									
0x28 (0x48)	OCR0B	Timer/Counter0 – Output Compare Register B								Page 102
0x27 (0x47)	OCR0A	Timer/Counter0 – Output Compare Register A								Page 102
0x26 (0x46)	TCNT0	Timer/Counter0 – Counter Register								Page 101
0x25 (0x45)	TCCR0B	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	Page 100
0x24 (0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	Page 97
0x23 (0x43)	GTCCR	TSM	–	–	–	–	–	–	PSR	Page 132
0x22 (0x42)	Reserved									
0x21 (0x41)	EEARL	EEPROM Address Register Low Byte								Page 23
0x20 (0x40)	EEDR	EEPROM Data Register								Page 24
0x1F (0x3F)	EECR	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	Page 24
0x1E (0x3E)	GPIOR0	General Purpose I/O register 0								Page 26
0x1D (0x3D)	EIMSK	–	–	–	–	–	–	INT1	INT0	Page 56
0x1C (0x3C)	EIFR	–	–	–	–	–	–	INT1	INTF0	Page 57
0x1B (0x3B)	PCIFR	–	–	–	–	PCIF3	PCIF2	PCIF1	PCIF0	Page 57
0x1A (0x3A)	Reserved	–	–	–	–	–	–	–	–	
0x19 (0x39)	Reserved	–	–	–	–	–	–	–	–	
0x18 (0x38)	Reserved	–	–	–	–	–	–	–	–	
0x17 (0x37)	Reserved	–	–	–	–	–	–	–	–	
0x16 (0x36)	TIFR1	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	Page 130

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page(s)
0x15 (0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0	Page 103
0x14 (0x34)	PHDE	–	–	–	–	–	PHDEC	–	–	Page 81
0x13 (0x33)	Reserved	–	–	–	–	–	–	–	–	
0x12 (0x32)	Reserved	–	–	–	–	–	–	–	–	
0x11 (0x31)	Reserved	–	–	–	–	–	–	–	–	
0x10 (0x30)	Reserved	–	–	–	–	–	–	–	–	
0x0F (0x2F)	PUED	–	–	–	–	PUED3	PUED2	PUED1	PUED0	Page 82
0x0E (0x2E)	PORTD	–	–	–	–	PORTD3	PORTD2	PORTD1	PORTD0	Page 82
0x0D (0x2D)	DDRD	–	–	–	–	DDD3	DDD2	DDD1	DDD0	Page 82
0x0C (0x2C)	PIND	–	–	–	–	PIND3	PIND2	PIND1	PIND0	Page 83
0x0B (0x2B)	PUEC	PUEC7	PUEC6	PUEC5	PUEC4	PUEC3	PUEC2	PUEC1	PUEC0	Page 83
0x0A (0x2A)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	Page 83
0x09 (0x29)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	Page 83
0x08 (0x28)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	Page 84
0x07 (0x27)	PUEB	PUEB7	PUEB6	PUEB5	PUEB4	PUEB3	PUEB2	PUEB1	PUEB0	Page 84
0x06 (0x26)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	Page 84
0x05 (0x25)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	Page 84
0x04 (0x24)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	Page 85
0x03 (0x23)	PUEA	PUEA7	PUEA6	PUEA5	PUEA4	PUEA3	PUEA2	PUEA1	PUEA0	Page 85
0x02 (0x22)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	Page 85
0x01 (0x21)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	Page 85
0x00 (0x20)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	Page 86

Note:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O Registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the Status Flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operation the specified bit, and can therefore be used on registers containing such Status Flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.

27. Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
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, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,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 is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1/2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A

28. Ordering Information

28.1 ATtiny828

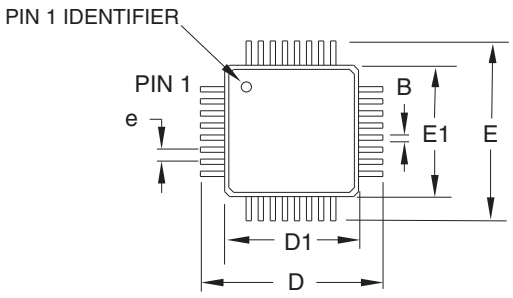
Speed (MHz) [1]	Supply Voltage (V) [1]	Temperature Range	Package [2]	Accuracy [3]	Ordering Code [4]
20 MHz	1.7 – 5.5V	Industrial [5] (-40°C to +85°C)	32A	±10%	ATtiny828-AU
				±2%	ATtiny828R-AU
				±10%	ATtiny828-AUR
				±2%	ATtiny828R-AUR
			32M1-A	±10%	ATtiny828-MU
				±2%	ATtiny828R-MU
				±10%	ATtiny828-MUR
				±2%	ATtiny828R-MUR

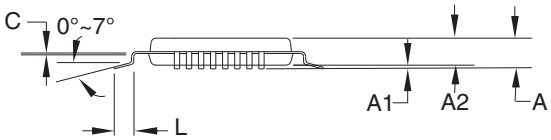
Notes:
1. For speed vs. supply voltage, see section “Speed” on page 249.
2. All packages are Pb-free, halide-free and fully green and they comply with the European directive for Restriction of Hazardous Substances (RoHS).
3. Indicates accuracy of internal oscillator. See “Accuracy of Calibrated Internal Oscillator” on page 249.
4. Code indicators:
   - U: matte tin
   - R: tape & reel
5. These devices can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.

Package Type	
32A	32-lead, Thin (1.0 mm) Plastic Quad Flat Package (TQFP)
32M1-A	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50 mm, Quad Flat No-Lead (QFN)

29. Packaging Information

29.1 32A





COMMON DIMENSIONS
(Unit of measure = mm)

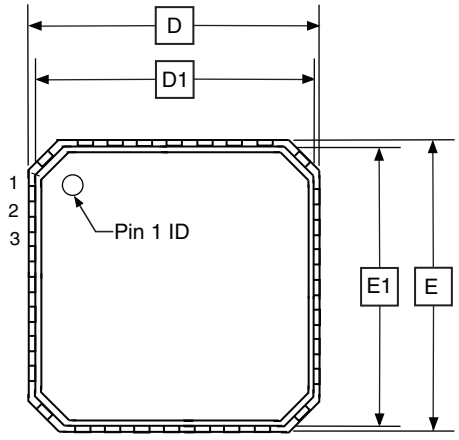
SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	1.20	
A1	0.05	–	0.15	
A2	0.95	1.00	1.05	
D	8.75	9.00	9.25	
D1	6.90	7.00	7.10	Note 2
E	8.75	9.00	9.25	
E1	6.90	7.00	7.10	Note 2
B	0.30	–	0.45	
C	0.09	–	0.20	
L	0.45	–	0.75	
e	0.80 TYP			

Notes:
1. This package conforms to JEDEC reference MS-026, Variation ABA.
2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
3. Lead coplanarity is 0.10mm maximum.

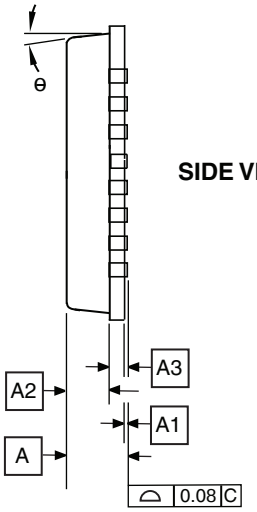
2010-10-20

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**32A,** 32-lead, 7 x 7mm body size, 1.0mm body thickness, 0.8mm lead pitch, thin profile plastic quad flat package (TQFP)	32A	C

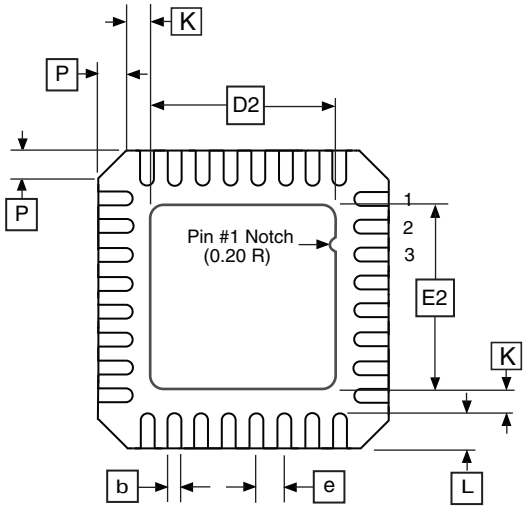
29.2 32M1-A



TOP VIEW



SIDE VIEW



BOTTOM VIEW

Note: JEDEC Standard MO-220, Fig. 2 (Anvil Singulation), VHHD-2.

COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	0.80	0.90	1.00	
A1	–	0.02	0.05	
A2	–	0.65	1.00	
A3		0.20 REF		
b	0.18	0.23	0.30	
D	4.90	5.00	5.10	
D1	4.70	4.75	4.80	
D2	2.95	3.10	3.25	
E	4.90	5.00	5.10	
E1	4.70	4.75	4.80	
E2	2.95	3.10	3.25	
e		0.50 BSC		
L	0.30	0.40	0.50	
P	–	–	0.60	
θ	–	–	12^{o}	
K	0.20	–	–	

5/25/06

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**32M1-A**, 32-pad, 5 x 5 x 1.0mm Body, Lead Pitch 0.50mm, 3.10mm Exposed Pad, Micro Lead Frame Package (MLF)	32M1-A	E

30. Errata

The revision letters in this section refer to the revision of the corresponding ATtiny828 device.

30.1 Rev. A

- Port Pin Restrictions When ULP Oscillator Is Disabled

1. Port Pin Restrictions When ULP Oscillator Is Disabled

Port pin PD3 is not guaranteed to perform as a reliable input when the Ultra Low Power (ULP) oscillator is not running. In addition, the pin is pulled down internally when ULP oscillator is disabled. TWI and SPI use may be limited when ULP is not running since pin PD3 is used by SCL and SCK signals.

Problem Fix / Workaround

The ULP oscillator is automatically activated when required. To use PD3 as an input or clock signal of TWI/SPI, activate the watchdog timer. The watchdog timer automatically enables the ULP oscillator.

31. Revision History

Doc. Rev.	Date	Comments
8371A	08/2012	Initial document release.

Table of Contents

Features 1

1. Pin Configurations 2
   1.1 Pin Description 3

2. Overview 5

3. General Information 7
   3.1 Resources 7
   3.2 Code Examples 7
   3.3 Data Retention 7

4. CPU Core 8
   4.1 Architectural Overview 8
   4.2 ALU – Arithmetic Logic Unit 9
   4.3 Status Register 9
   4.4 General Purpose Register File 9
   4.5 Stack Pointer 11
   4.6 Instruction Execution Timing 11
   4.7 Reset and Interrupt Handling 12
   4.8 Register Description 14

5. Memories 16
   5.1 Program Memory (Flash) 16
   5.2 Data Memory (SRAM) and Register Files 17
   5.3 Data Memory (EEPROM) 19
   5.4 Register Description 23

6. Clock System 27
   6.1 Clock Subsystems 27
   6.2 Clock Sources 28
   6.3 System Clock Prescaler 30
   6.4 Clock Output Buffer 30
   6.5 Start-Up Time 30
   6.6 Register Description 31

7. Power Management and Sleep Modes 34
   7.1 Sleep Modes 34
   7.2 Power Reduction Register 35
   7.3 Minimizing Power Consumption 35
   7.4 Register Description 37

8. System Control and Reset 39
   8.1 Resetting the AVR 39
   8.2 Reset Sources 39
   8.3 Internal Voltage Reference 43
   8.4 Watchdog Timer 43
   8.5 Register Description 45

9. Interrupts 48
   9.1 Interrupt Vectors 48

9.2 External Interrupts . . . 51
9.3 Register Description . . . 53

10. I/O Ports . . . 59
10.1 Overview . . . 59
10.2 Ports as General Digital I/O . . . 59
10.3 Alternative Port Functions . . . 63
10.4 Register Description . . . 81

11. 8-bit Timer/Counter0 with PWM . . . 87
11.1 Features . . . 87
11.2 Overview . . . 87
11.3 Clock Sources . . . 88
11.4 Counter Unit . . . 88
11.5 Output Compare Unit . . . 89
11.6 Compare Match Output Unit . . . 91
11.7 Modes of Operation . . . 92
11.8 Timer/Counter Timing Diagrams . . . 96
11.9 Register Description . . . 97

12. 16-bit Timer/Counter1 . . . 104
12.1 Features . . . 104
12.2 Overview . . . 104
12.3 Timer/Counter Clock Sources . . . 105
12.4 Counter Unit . . . 106
12.5 Input Capture Unit . . . 106
12.6 Output Compare Units . . . 108
12.7 Compare Match Output Unit . . . 110
12.8 Modes of Operation . . . 111
12.9 Timer/Counter Timing Diagrams . . . 118
12.10 Accessing 16-bit Registers . . . 120
12.11 Register Description . . . 123

13. Timer/Counter Prescaler . . . 131
13.1 Prescaler Reset . . . 131
13.2 External Clock Source . . . 131
13.3 Register Description . . . 132

14. Analog Comparator . . . 133
14.1 Register Description . . . 134

15. Analog to Digital Converter . . . 137
15.1 Features . . . 137
15.2 Overview . . . 137
15.3 Operation . . . 138
15.4 Starting a Conversion . . . 139
15.5 Prescaling and Conversion Timing . . . 140
15.6 Changing Channel or Reference Selection . . . 143
15.7 ADC Noise Canceler . . . 144
15.8 Analog Input Circuitry . . . 144
15.9 Noise Canceling Techniques . . . 145
15.10 ADC Accuracy Definitions . . . 145

15.11 ADC Conversion Result . . . 148
15.12 Temperature Measurement . . . 148
15.13 Register Description . . . 149

16. SPI – Serial Peripheral Interface . . . 155
16.1 Features . . . 155
16.2 Overview . . . 155
16.3 $\overline{SS}$ Pin Functionality . . . 159
16.4 Data Modes . . . 160
16.5 Register Description . . . 161

17. USART . . . 164
17.1 Features . . . 164
17.2 Overview . . . 164
17.3 Clock Generation . . . 165
17.4 Frame Formats . . . 168
17.5 USART Initialization . . . 169
17.6 Data Transmission – The USART Transmitter . . . 170
17.7 Data Reception – The USART Receiver . . . 173
17.8 Asynchronous Data Reception . . . 177
17.9 Multi-processor Communication Mode . . . 180
17.10 Examples of Baud Rate Setting . . . 180
17.11 Register Description . . . 184

18. USART in SPI Mode . . . 190
18.1 Features . . . 190
18.2 Overview . . . 190
18.3 Clock Generation . . . 190
18.4 SPI Data Modes and Timing . . . 191
18.5 Frame Formats . . . 191
18.6 Data Transfer . . . 193
18.7 AVR USART MSPIM vs. AVR SPI . . . 195
18.8 Register Description . . . 195

19. I2C Compatible, Two-Wire Slave Interface . . . 199
19.1 Features . . . 199
19.2 Overview . . . 199
19.3 General TWI Bus Concepts . . . 199
19.4 TWI Slave Operation . . . 205
19.5 Register Description . . . 207

20. debugWIRE On-chip Debug System . . . 212
20.1 Features . . . 212
20.2 Overview . . . 212
20.3 Physical Interface . . . 212
20.4 Software Break Points . . . 213
20.5 Limitations of debugWIRE . . . 213
20.6 Register Description . . . 213

21. Self-Programming with Boot Loader and Read-While-Write . . . 214
21.1 Features . . . 214
21.2 Overview . . . 214

21.3 Application and Boot Loader Flash Sections . . . 214
21.4 Read-While-Write and No Read-While-Write Flash Sections . . . 214
21.5 Entering the Boot Loader Program . . . 216
21.6 Configuring the Boot Loader . . . 216
21.7 Boot Loader Lock Bits . . . 218
21.8 Self-Programming the Flash . . . 218
21.9 Preventing Flash Corruption . . . 222
21.10 Programming Time for Flash when Using SPM . . . 223
21.11 Register Description . . . 223

22. Lock Bits, Fuse Bits and Device Signature . . . 225
22.1 Lock Bits . . . 225
22.2 Fuse Bits . . . 226
22.3 Device Signature Imprint Table . . . 228
22.4 Reading Lock, Fuse and Signature Data from Software . . . 229

23. External Programming . . . 232
23.1 Memory Parametrics . . . 232
23.2 Parallel Programming . . . 232
23.3 Serial Programming . . . 241
23.4 Programming Time for Flash and EEPROM . . . 245

24. Electrical Characteristics . . . 247
24.1 Absolute Maximum Ratings* . . . 247
24.2 DC Characteristics . . . 247
24.3 Speed . . . 249
24.4 Clock Characteristics . . . 249
24.5 System and Reset Characteristics . . . 250
24.6 Temperature Sensor . . . 251
24.7 Two-Wire Serial Interface Characteristics . . . 252
24.8 ADC Characteristics . . . 253
24.9 Analog Comparator Characteristics . . . 254
24.10 Parallel Programming Characteristics . . . 254
24.11 Serial Programming Characteristics . . . 257

25. Typical Characteristics . . . 258
25.1 Current Consumption in Active Mode . . . 258
25.2 Current Consumption in Idle Mode . . . 261
25.3 Current Consumption in Power-down Mode . . . 263
25.4 Current Consumption in Reset . . . 264
25.5 Current Consumption of Peripheral Units . . . 266
25.6 Pull-up Resistors . . . 268
25.7 Input Thresholds . . . 271
25.8 Current Source Strength . . . 277
25.9 Current Sink Capability . . . 280
25.10 BOD . . . 286
25.11 Bandgap Voltage . . . 289
25.12 Reset . . . 290
25.13 Analog Comparator Offset . . . 291
25.14 Internal Oscillator Speed . . . 293

26. Register Summary . . . 297

27. Instruction Set Summary 301

28. Ordering Information 303

28.1 ATtiny828 303

29. Packaging Information 304

29.1 32A 304

29.2 32M1-A 305

30. Errata 306

30.1 Rev. A 306

31. Revision History 307

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation
1600 Technology Drive
San Jose, CA 95110
USA
Tel: (+1) (408) 441-0311
Fax: (+1) (408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Roa
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan G.K.
16F Shin-Osaki Kangyo Bldg
1-6-4 Osaki, Shinagawa-ku
Tokyo 141-0032
JAPAN
Tel: (+81) (3) 6417-0300
Fax: (+81) (3) 6417-0370

© 2012 Atmel Corporation. All rights reserved. / Rev.: 8371A–AVR–08/12

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.