8-bit AVR Microcontrollers

# ATmega48A/88A/168A

DATASHEET COMPLETE

## Introduction

The Atmel® ATmega48A/88A/168A is a low-power CMOS 8-bit microcontroller based on the AVR® enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega48A/88A/168A achieves throughputs close to 1MIPS per MHz. This empowers system designer to optimize the device for power consumption versus processing speed.

## Feature

High Performance, Low Power Atmel®AVR® 8-Bit Microcontroller Family

- Advanced RISC Architecture
  - 131 Powerful Instructions
  - Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4K/8K/16KBytes of In-System Self-Programmable Flash program Memory
  - 256/512/512Bytes EEPROM
  - 512/1K/1KBytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data Retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Atmel® QTouch® Library Support
  - Capacitive Touch Buttons, Sliders and Wheels
  - QTouch and QMatrix® Acquisition
  - Up to 64 sense channels

- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Two Master/Slave SPI Serial Interface
  - One Programmable Serial USART
  - One Byte-oriented 2-wire Serial Interface (Philips $I^2C$ compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - One On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 4MHz @ 1.8 - 5.5V
  - 0 - 10MHz @ 2.7 - 5.5V
  - 0 - 20MHz @ 4.5 - 5.5V
- Power Consumption at 1MHz, 1.8V, 25°C
  - Active Mode: 0.2mA
  - Power-down Mode: 0.1µA
  - Power-save Mode: 0.75µA (Including 32kHz RTC)

# Table of Contents


Introduction....1

Feature....1

1. Description....9

2. Configuration Summary....10

3. Ordering Information....11

3.1. ATmega48A....11
3.2. ATmega88A....12
3.3. ATmega168A....13

4. Block Diagram....14

5. Pin Configurations....15

5.1. Pin-out....15
5.2. Pin Descriptions....19

6. I/O Multiplexing....21

7. Resources....23

8. Data Retention....24

9. About Code Examples....25

10. Capacitive Touch Sensing....26

10.1. QTouch Library....26

11. AVR CPU Core....27

11.1. Overview....27
11.2. ALU – Arithmetic Logic Unit....28
11.3. Status Register....28
11.4. General Purpose Register File....30
11.5. Stack Pointer....31
11.6. Instruction Execution Timing....33
11.7. Reset and Interrupt Handling....34

12. AVR Memories....36

12.1. Overview....36
12.2. In-System Reprogrammable Flash Program Memory....36
12.3. SRAM Data Memory....37
12.4. EEPROM Data Memory....39
12.5. I/O Memory....40
12.6. Register Description....41

13. System Clock and Clock Options ..... 51
13.1. Clock Systems and Their Distribution ..... 51
13.2. Clock Sources ..... 52
13.3. Low Power Crystal Oscillator ..... 54
13.4. Full Swing Crystal Oscillator ..... 55
13.5. Low Frequency Crystal Oscillator ..... 56
13.6. Calibrated Internal RC Oscillator ..... 57
13.7. 128kHz Internal Oscillator ..... 58
13.8. External Clock ..... 59
13.9. Timer/Counter Oscillator ..... 60
13.10. Clock Output Buffer ..... 60
13.11. System Clock Prescaler ..... 60
13.12. Register Description ..... 61

14. PM - Power Management and Sleep Modes ..... 65
14.1. Overview ..... 65
14.2. Sleep Modes ..... 65
14.3. Idle Mode ..... 65
14.4. ADC Noise Reduction Mode ..... 66
14.5. Power-Down Mode ..... 66
14.6. Power-save Mode ..... 67
14.7. Standby Mode ..... 67
14.8. Extended Standby Mode ..... 67
14.9. Power Reduction Register ..... 67
14.10. Minimizing Power Consumption ..... 68
14.11. Register Description ..... 69

15. SCRST - System Control and Reset ..... 74
15.1. Resetting the AVR ..... 74
15.2. Reset Sources ..... 74
15.3. Power-on Reset ..... 75
15.4. External Reset ..... 76
15.5. Brown-out Detection ..... 76
15.6. Watchdog System Reset ..... 77
15.7. Internal Voltage Reference ..... 77
15.8. Watchdog Timer ..... 78
15.9. Register Description ..... 80

16. Interrupts ..... 84
16.1. Interrupt Vectors in ATmega48A ..... 84
16.2. Interrupt Vectors in ATmega88A ..... 85
16.3. Interrupt Vectors in ATmega168A ..... 88
16.4. Register Description ..... 91

17. EXINT - External Interrupts ..... 94
17.1. Pin Change Interrupt Timing ..... 94
17.2. Register Description ..... 95

18. I/O-Ports ..... 104

18.1. Overview....104
18.2. Ports as General Digital I/O....105
18.3. Alternate Port Functions....108
18.4. Register Description....120

19. TC0 - 8-bit Timer/Counter0 with PWM....132
19.1. Features....132
19.2. Overview....132
19.3. Timer/Counter Clock Sources....134
19.4. Counter Unit....134
19.5. Output Compare Unit....135
19.6. Compare Match Output Unit....137
19.7. Modes of Operation....138
19.8. Timer/Counter Timing Diagrams....142
19.9. Register Description....144

20. TC1 - 16-bit Timer/Counter1 with PWM....156
20.1. Overview....156
20.2. Features....156
20.3. Block Diagram....156
20.4. Definitions....157
20.5. Registers....158
20.6. Accessing 16-bit Registers....158
20.7. Timer/Counter Clock Sources....161
20.8. Counter Unit....161
20.9. Input Capture Unit....162
20.10. Output Compare Units....164
20.11. Compare Match Output Unit....166
20.12. Modes of Operation....167
20.13. Timer/Counter Timing Diagrams....175
20.14. Register Description....176

21. Timer/Counter 0, 1 Prescalers....193
21.1. Internal Clock Source....193
21.2. Prescaler Reset....193
21.3. External Clock Source....193
21.4. Register Description....194

22. TC2 - 8-bit Timer/Counter2 with PWM and Asynchronous Operation....196
22.1. Features....196
22.2. Overview....196
22.3. Timer/Counter Clock Sources....198
22.4. Counter Unit....198
22.5. Output Compare Unit....199
22.6. Compare Match Output Unit....201
22.7. Modes of Operation....202
22.8. Timer/Counter Timing Diagrams....206
22.9. Asynchronous Operation of Timer/Counter2....207
22.10. Timer/Counter Prescaler....209

22.11. Register Description ..... 209

23. SPI – Serial Peripheral Interface ..... 222

23.1. Features ..... 222
23.2. Overview ..... 222
23.3. $\overline{SS}$ Pin Functionality ..... 226
23.4. Data Modes ..... 226
23.5. Register Description ..... 227

24. USART - Universal Synchronous Asynchronous Receiver Transceiver ..... 232

24.1. Features ..... 232
24.2. Overview ..... 232
24.3. Block Diagram ..... 232
24.4. Clock Generation ..... 233
24.5. Frame Formats ..... 236
24.6. USART Initialization ..... 237
24.7. Data Transmission – The USART Transmitter ..... 238
24.8. Data Reception – The USART Receiver ..... 240
24.9. Asynchronous Data Reception ..... 244
24.10. Multi-Processor Communication Mode ..... 246
24.11. Examples of Baud Rate Setting ..... 247
24.12. Register Description ..... 250

25. USARTSPI - USART in SPI Mode ..... 261

25.1. Features ..... 261
25.2. Overview ..... 261
25.3. Clock Generation ..... 261
25.4. SPI Data Modes and Timing ..... 262
25.5. Frame Formats ..... 262
25.6. Data Transfer ..... 264
25.7. AVR USART MSPIM vs. AVR SPI ..... 265
25.8. Register Description ..... 266

26. TWI - 2-wire Serial Interface ..... 267

26.1. Features ..... 267
26.2. Two-Wire Serial Interface Bus Definition ..... 267
26.3. Data Transfer and Frame Format ..... 268
26.4. Multi-master Bus Systems, Arbitration and Synchronization ..... 271
26.5. Overview of the TWI Module ..... 273
26.6. Using the TWI ..... 275
26.7. Transmission Modes ..... 278
26.8. Multi-master Systems and Arbitration ..... 296
26.9. Register Description ..... 298

27. AC - Analog Comparator ..... 306

27.1. Overview ..... 306
27.2. Analog Comparator Multiplexed Input ..... 306
27.3. Register Description ..... 307

28. ADC - Analog to Digital Converter....312
28.1. Features....312
28.2. Overview....312
28.3. Starting a Conversion....314
28.4. Prescaling and Conversion Timing....315
28.5. Changing Channel or Reference Selection....317
28.6. ADC Noise Canceler....319
28.7. ADC Conversion Result....322
28.8. Temperature Measurement....323
28.9. Register Description....323

29. DBG - debugWIRE On-chip Debug System....334
29.1. Features....334
29.2. Overview....334
29.3. Physical Interface....334
29.4. Software Break Points....335
29.5. Limitations of debugWIRE....335
29.6. Register Description....335

30. Self-Programming the Flash....337
30.1. Overview....337
30.2. Addressing the Flash During Self-Programming....338
30.3. Register Description....343

31. BTLDR - Boot Loader Support – Read-While-Write Self-Programming....346
31.1. Features....346
31.2. Overview....346
31.3. Application and Boot Loader Flash Sections....346
31.4. Read-While-Write and No Read-While-Write Flash Sections....347
31.5. Boot Loader Lock Bits....349
31.6. Entering the Boot Loader Program....350
31.7. Addressing the Flash During Self-Programming....351
31.8. Self-Programming the Flash....352
31.9. Register Description....361

32. MEMPROG- Memory Programming....364
32.1. Program And Data Memory Lock Bits....364
32.2. Fuse Bits....365
32.3. Signature Bytes....368
32.4. Calibration Byte....368
32.5. Page Size....368
32.6. Parallel Programming Parameters, Pin Mapping, and Commands....368
32.7. Parallel Programming....370
32.8. Serial Downloading....377

33. Electrical Characteristics....383
33.1. Absolute Maximum Ratings....383
33.2. Common DC Characteristics....383

33.3. Speed Grades ..... 387
33.4. Clock Characteristics ..... 387
33.5. System and Reset Characteristics ..... 388
33.6. SPI Timing Characteristics ..... 390
33.7. Two-wire Serial Interface Characteristics ..... 391
33.8. ADC Characteristics ..... 393
33.9. Parallel Programming Characteristics ..... 394

34. Typical Characteristics ($T_A$ = -40°C to 85°C) ..... 397
34.1. ATmega48A Typical Characteristics ..... 397
34.2. ATmega88A: Typical Characteristics ..... 418
34.3. ATmega168A Typical Characteristics ..... 442

35. Register Summary ..... 465
35.1. Note ..... 467

36. Instruction Set Summary ..... 469

37. Packaging Information ..... 473
37.1. 32-pin 32A ..... 473
37.2. 32-pin 32M1-A ..... 474
37.3. 32-pin 32CC1 ..... 475
37.4. 28-pin 28M1 ..... 476
37.5. 28-pin 28P3 ..... 477

38. Errata ..... 478
38.1. Errata ATmega48A ..... 478
38.2. Errata ATmega88A ..... 478
38.3. Errata ATmega168A ..... 479

39. Datasheet Revision History ..... 480
39.1. Rev. B – 11/2016 ..... 480
39.2. Rev. A – 06/2016 ..... 480

# 1. Description

The Atmel AVR® core combines a rich instruction set with 32 general purpose working registers. All the 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 more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega48A/88A/168A provides the following features: 4K/8K/16Kbytes of In-System Programmable Flash with Read-While-Write capabilities, 256/512/512bytes EEPROM, 512/1K/1Kbytes SRAM, 23 general purpose I/O lines, 32 general purpose working registers, Real Time Counter (RTC), three flexible Timer/Counters with compare modes and PWM, 1 serial programmable USARTs , 1 byte-oriented 2-wire Serial Interface (I2C), a 6-channel 10-bit ADC (8 channels in TQFP and QFN/MLF packages) , a programmable Watchdog Timer with internal Oscillator, an SPI serial port, and six software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption. In Extended Standby mode, both the main oscillator and the asynchronous timer continue to run.

Atmel offers the QTouch® library for embedding capacitive touch buttons, sliders and wheels functionality into AVR microcontrollers. The patented charge-transfer signal acquisition offers robust sensing and includes fully debounced reporting of touch keys and includes Adjacent Key Suppression® (AKS™) technology for unambiguous detection of key events. The easy-to-use QTouch Suite toolchain allows you to explore, develop and debug your own touch applications.

The device is manufactured using Atmel’s high density non-volatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The Boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega48A/88A/168A is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega48A/88A/168A is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, Program Debugger/Simulators, In-Circuit Emulators, and Evaluation kits.

## 2. Configuration Summary

| Features | ATmega48A/88A/168A |
|---|---|
| Pin Count | 28/32 |
| Flash (Bytes) | 4K/8K/16K |
| SRAM (Bytes) | 512/1K/1K |
| EEPROM (Bytes) | 256/512/512 |
| Interrupt Vector Size (instruction word/vector) | 1/1/2 |
| General Purpose I/O Lines | 23 |
| SPI | 2 |
| TWI ($I^2C$) | 1 |
| USART | 1 |
| ADC | 10-bit 15kSPS |
| ADC Channels | 8 |
| 8-bit Timer/Counters | 2 |
| 16-bit Timer/Counters | 1 |

ATmega88A and ATmega168A support a real Read-While-Write Self-Programming mechanism. There is a separate Boot Loader Section, and the SPM instruction can only execute from there. In ATmega48A, there is no Read-While-Write support and no separate Boot Loader Section. The SPM instruction can execute from the entire Flash.

# 3. Ordering Information

## 3.1. ATmega48A

| Speed [MHz][3] | Power Supply [V] | Ordering Code[2] | Package[1] | Operational Range |
|---|---|---|---|---|
| 20 | 1.8 - 5.5 | ATmega48A-AU<br>ATmega48A-AUR[4]<br>ATmega48A-CCU<br>ATmega48A-CCUR[4]<br>ATmega48A-MMH[5]<br>ATmega48A-MMHR[4][5]<br>ATmega48A-MU<br>ATmega48A-MUR[4]<br>ATmega48A-PU | 32A<br>32A<br>32CC1<br>32CC1<br>28M1<br>28M1<br>32M1-A<br>32M1-A<br>28P3 | Industrial<br>(-40°C to 85°C) |

**Note:**

1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
3. Please refer to *Speed Grades* for Speed vs. $V_{CC}$
4. Tape & Reel.
5. NiPdAu Lead Finish.

| Package Type | |
|---|---|
| 28M1 | 28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF) |
| 28P3 | 28-lead, 0.300” Wide, Plastic Dual Inline Package (PDIP) |
| 32M1-A | 32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF) |
| 32A | 32-lead, Thin (1.0mm) Plastic Quad Flat Package (TQFP) |
| 32CC1 | 32-ball, 4 x 4 x 0.6mm package, ball pitch 0.5mm, Ultra Thin, Fine-Pitch Ball Grill Array (UFBGA) |

## 3.2. ATmega88A

| Speed [MHz][3] | Power Supply [V] | Ordering Code[2] | Package[1] | Operational Range |
|---|---|---|---|---|
| 20 | 1.8 - 5.5 | ATmega88A-AU<br>ATmega88A-AUR[4]<br>ATmega88A-CCU<br>ATmega88A-CCUR[4]<br>ATmega88A-MMH[5]<br>ATmega88A-MMHR[4][5]<br>ATmega88A-MU<br>ATmega88A-MUR[4]<br>ATmega88A-PU | 32A<br>32A<br>32CC1<br>32CC1<br>28M1<br>28M1<br>32M1-A<br>32M1-A<br>28P3 | Industrial<br>(-40°C to 85°C) |

**Note:**

1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
3. Please refer to *Speed Grades* for Speed vs. $V_{CC}$
4. Tape & Reel.
5. NiPdAu Lead Finish.

| Package Type | |
|---|---|
| 28M1 | 28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF) |
| 28P3 | 28-lead, 0.300” Wide, Plastic Dual Inline Package (PDIP) |
| 32M1-A | 32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF) |
| 32A | 32-lead, Thin (1.0mm) Plastic Quad Flat Package (TQFP) |
| 32CC1 | 32-ball, 4 x 4 x 0.6mm package, ball pitch 0.5mm, Ultra Thin, Fine-Pitch Ball Grill Array (UFBGA) |

## 3.3. ATmega168A

| Speed [MHz][3] | Power Supply [V] | Ordering Code[2] | Package[1] | Operational Range |
|---|---|---|---|---|
| 20 | 1.8 - 5.5 | ATmega168A-AU<br>ATmega168A-AUR[5]<br>ATmega168A-CCU<br>ATmega168A-CCUR[5]<br>ATmega168A-MMH[4]<br>ATmega168A-MMHR[4][5]<br>ATmega168A-MU<br>ATmega168A-MUR[5]<br>ATmega168A-PU | 32A<br>32A<br>32CC1<br>32CC1<br>28M1<br>28M1<br>32M1-A<br>32M1-A<br>28P3 | Industrial<br>(-40°C to 85°C) |

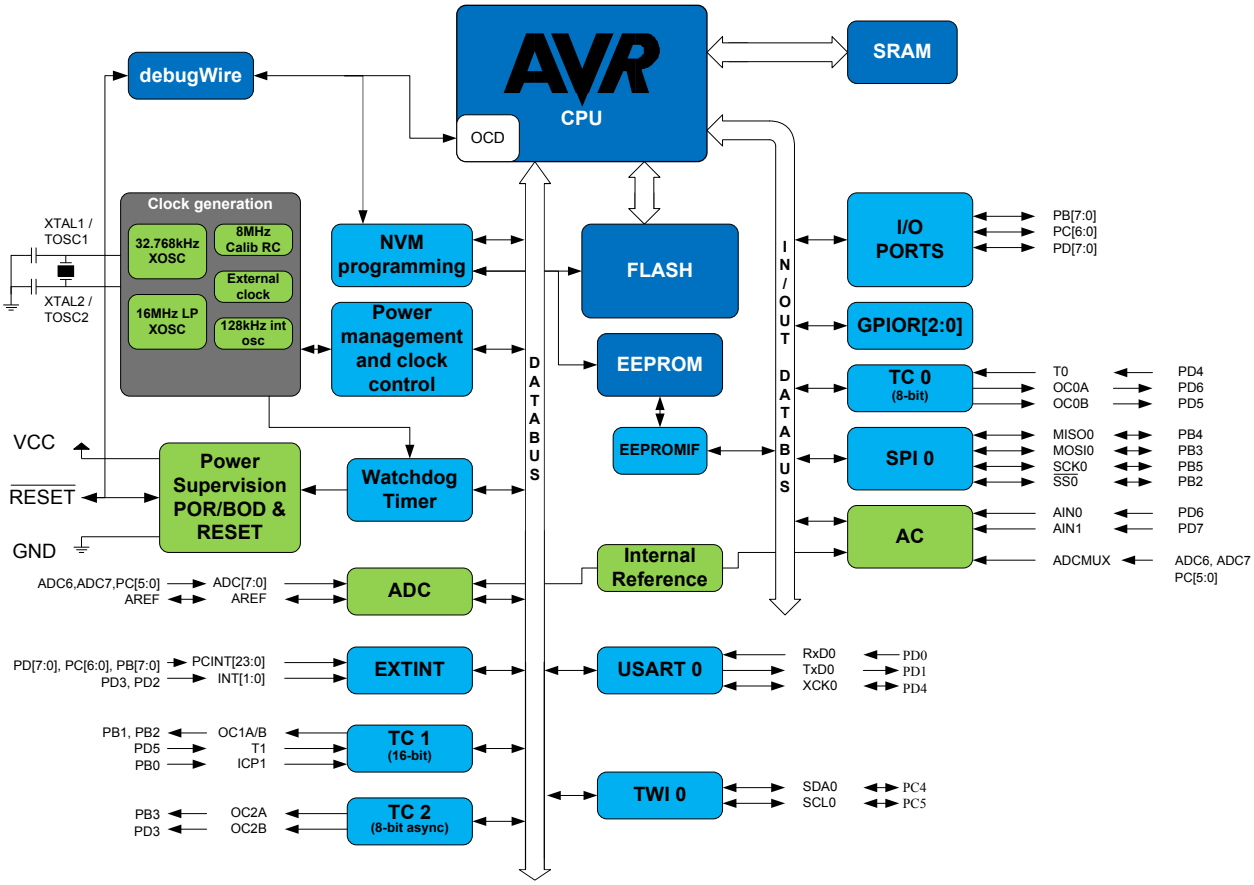
**Note:**

1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
3. Please refer to *Speed Grades* for Speed vs. $V_{CC}$
4. Tape & Reel.
5. NiPdAu Lead Finish.

| Package Type | |
|---|---|
| 28M1 | 28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF) |
| 28P3 | 28-lead, 0.300” Wide, Plastic Dual Inline Package (PDIP) |
| 32M1-A | 32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF) |
| 32A | 32-lead, Thin (1.0mm) Plastic Quad Flat Package (TQFP) |
| 32CC1 | 32-ball, 4 x 4 x 0.6mm package, ball pitch 0.5mm, Ultra Thin, Fine-Pitch Ball Grill Array (UFBGA) |

# 4. Block Diagram

**Figure 4-1. Block Diagram**



# 5. Pin Configurations

## 5.1. Pin-out

**Figure 5-1. 28-pin PDIP**

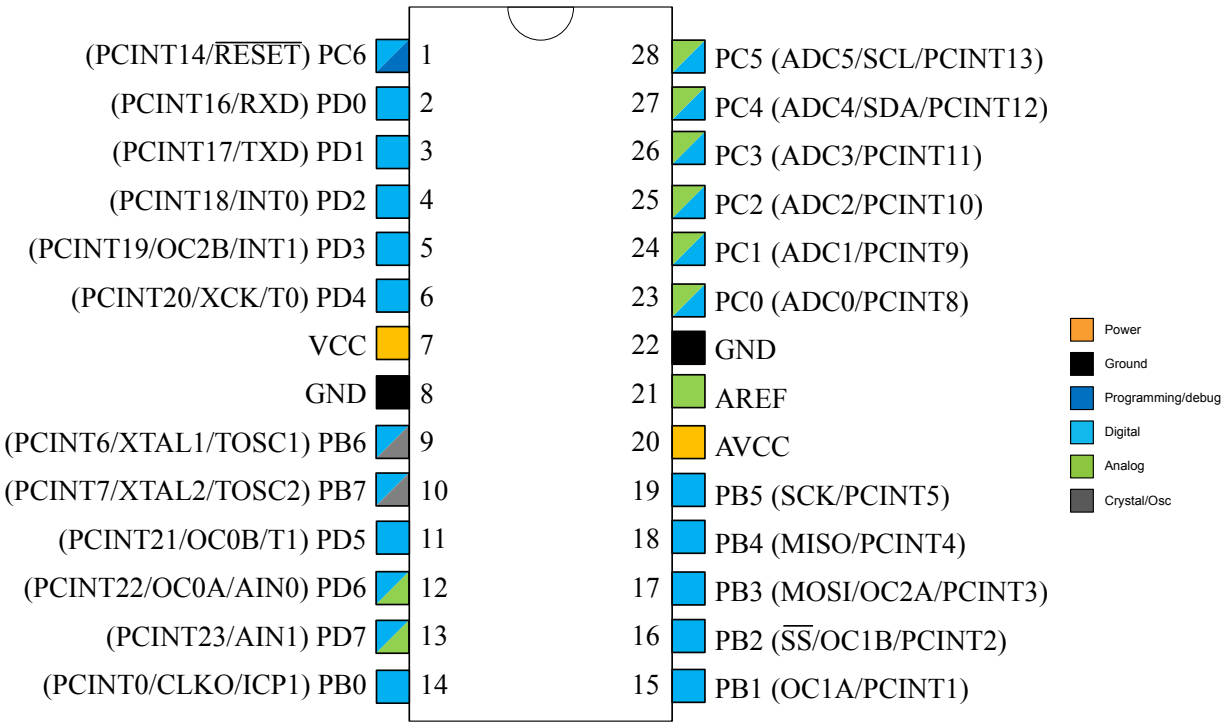


**Figure 5-2. 28-pin MLF Top View**

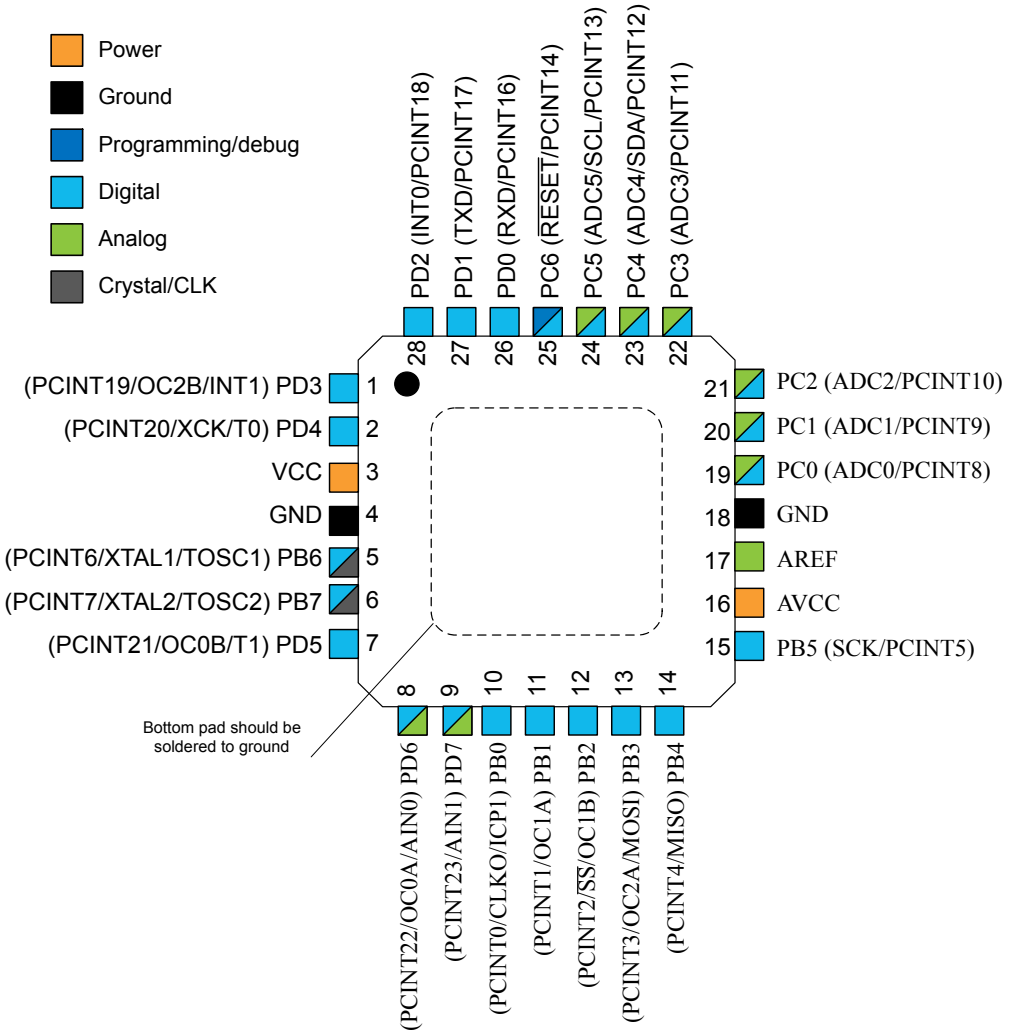


**Figure 5-3. 32-pin TQFP Top View**

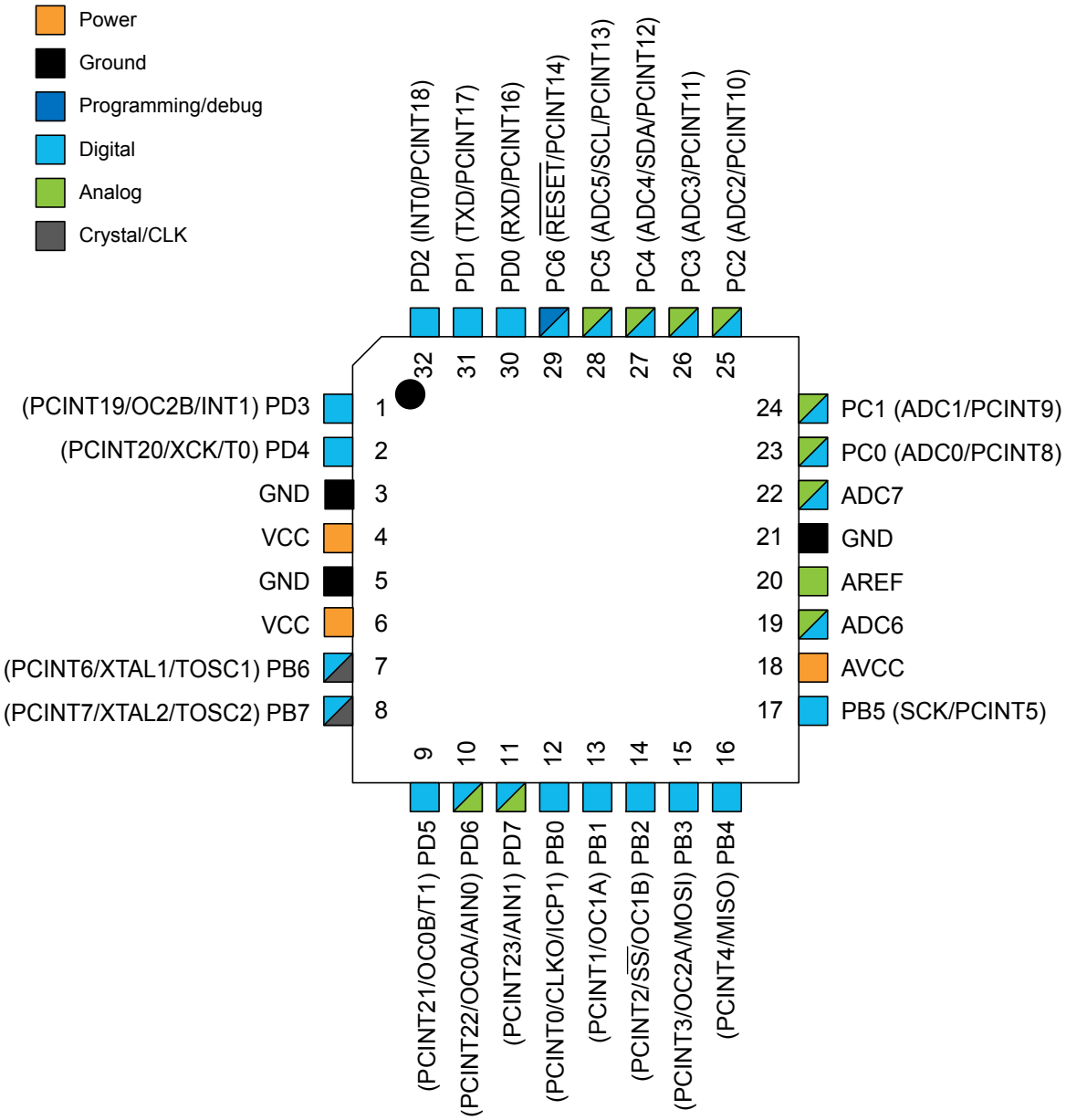


**Figure 5-4. 32-pin MLF Top View**

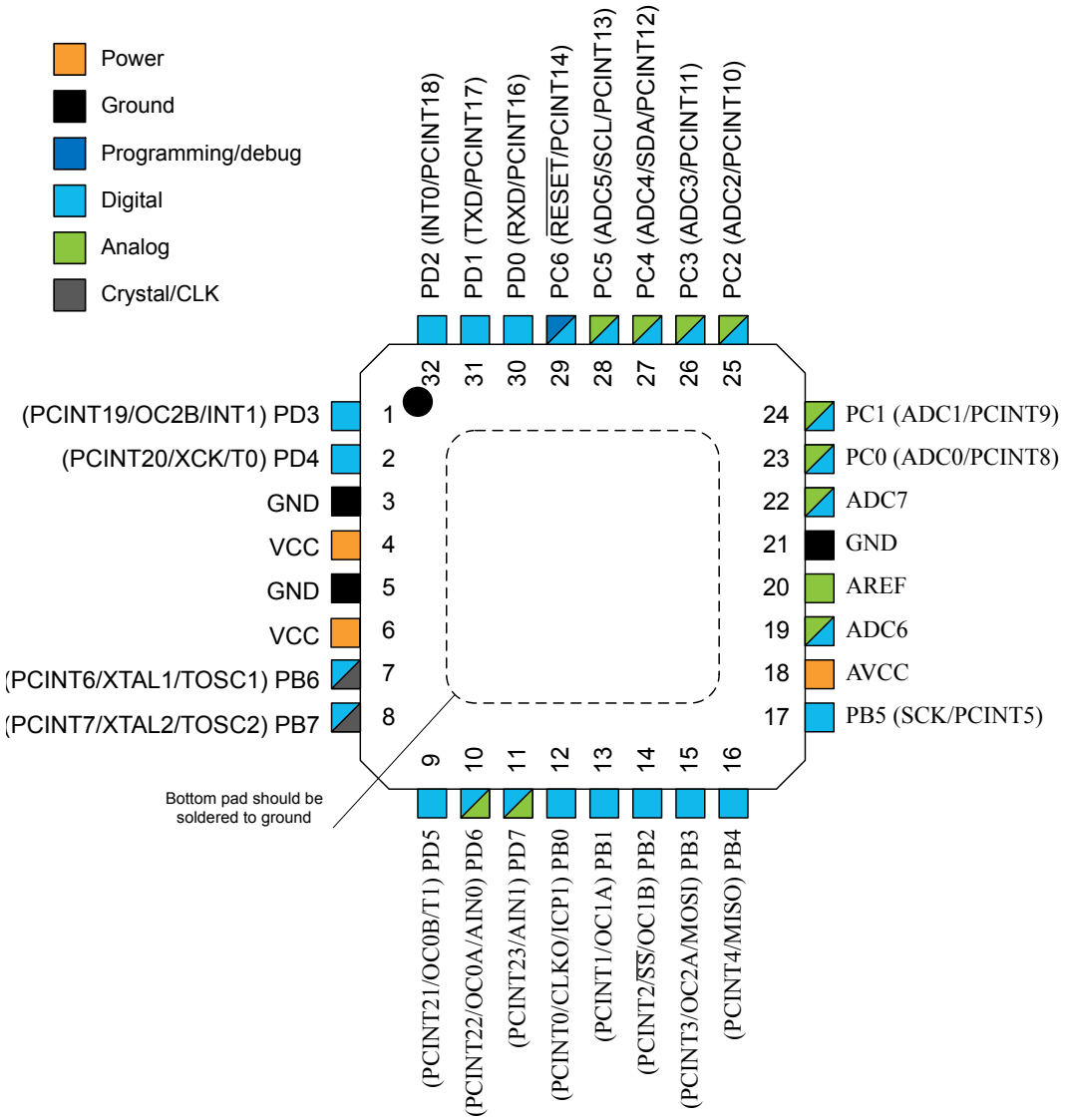



**Table 5-1. 32UFBGA**

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| A | PD2 | PD1 | PC6 | PC4 | PC2 | PC1 |
| B | PD3 | PD4 | PD0 | PC5 | PC3 | PC0 |
| C | GND | GND | - | - | ADC7 | GND |
| D | VCC | VCC | - | - | AREF | ADC6 |
| E | PB6 | PD6 | PB0 | PB2 | AVCC | PB5 |
| F | PB7 | PD5 | PD7 | PB1 | PB3 | PB4 |

## 5.2. Pin Descriptions

### 5.2.1. VCC

Digital supply voltage.

### 5.2.2. GND

Ground.

### 5.2.3. Port B (PB[7:0]) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB[7:6] is used as TOSC[2:1] input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

### 5.2.4. Port C (PC[5:0])

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC[5:0] output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

### 5.2.5. PC6/$\overline{\text{RESET}}$

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a 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. Shorter pulses are not guaranteed to generate a Reset.

The various special features of Port C are elaborated in the *Alternate Functions of Port C* section.

### 5.2.6. Port D (PD[7:0])

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

### 5.2.7. $AV_{CC}$

$AV_{CC}$ is the supply voltage pin for the A/D Converter, PC[3:0], and PE[3:2]. It should be externally connected to $V_{CC}$, even if the ADC is not used. If the ADC is used, it should be connected to $V_{CC}$ through a low-pass filter. Note that PC[6:4] use digital supply voltage, $V_{CC}$.

### 5.2.8. AREF

AREF is the analog reference pin for the A/D Converter.

### 5.2.9. ADC[7:6] (TQFP and VFQFN Package Only)

In the TQFP and VFQFN package, ADC[7:6] serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

# 6. I/O Multiplexing

Each pin is by default controlled by the PORT as a general purpose I/O and alternatively it can be assigned to one of the peripheral functions.

The following table describes the peripheral signals multiplexed to the PORT I/O pins.

**Table 6-1. PORT Function Multiplexing**

| (32-pin 32UFBGA) Pin# | (32-pin MLF/ TQFP) Pin# | (28-pin MLF) Pin# | (28-pin PIPD) Pin# | PAD | EXTINT | PCINT | ADC/AC | OSC | T/C #0 | T/C #1 | USART 0 | I2C 0 | SPI 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| B1 | 1 | 1 | 5 | PD[3] | INT1 | PCINT19 | | | OC2B | | | | |
| B2 | 2 | 2 | 6 | PD[4] | | PCINT20 | | | T0 | | XCK0 | | |
| D1 | 4 | 3 | 7 | VCC | | | | | | | | | |
| C1 | 3 | 4 | 8 | GND | | | | | | | | | |
| D2 | 6 | - | - | VCC | | | | | | | | | |
| C2 | 5 | - | - | GND | | | | | | | | | |
| E1 | 7 | 5 | 9 | PB[6] | | PCINT6 | | XTAL1/ TOSC1 | | | | | |
| F1 | 8 | 6 | 10 | PB[7] | | PCINT7 | | XTAL2/ TOSC2 | | | | | |
| F2 | 9 | 7 | 11 | PD[5] | | PCINT21 | | | OC0B | T1 | | | |
| E2 | 10 | 8 | 12 | PD[6] | | PCINT22 | AIN0 | | OC0A | | | | |
| F3 | 11 | 9 | 13 | PD[7] | | PCINT23 | AIN1 | | | | | | |
| E3 | 12 | 10 | 14 | PB[0] | | PCINT0 | | CLKO | ICP1 | | | | |
| F4 | 13 | 11 | 15 | PB[1] | | PCINT1 | | | OC1A | | | | |
| E4 | 14 | 12 | 16 | PB[2] | | PCINT2 | | | OC1B | | | | $\overline{SS0}$ |
| F5 | 15 | 13 | 17 | PB[3] | | PCINT3 | | | OC2A | | | | MOSI0 |
| F6 | 16 | 14 | 18 | PB[4] | | PCINT4 | | | | | | | MISO0 |
| E6 | 17 | 15 | 19 | PB[5] | | PCINT5 | | | | | | | SCK0 |
| E5 | 18 | 16 | 20 | AVCC | | | | | | | | | |
| D6 | 19 | - | - | ADC6 | | | ADC6 | | | | | | |
| D5 | 20 | 17 | 21 | AREF | | | | | | | | | |
| C6 | 21 | 18 | 22 | GND | | | | | | | | | |
| C5 | 22 | - | - | ADC7 | | | ADC7 | | | | | | |
| B6 | 23 | 19 | 13 | PC[0] | | PCINT8 | ADC0 | | | | | | |
| A6 | 24 | 20 | 24 | PC[1] | | PCINT9 | ADC1 | | | | | | |
| A2 | 25 | 21 | 25 | PC[2] | | PCINT10 | ADC2 | | | | | | |
| B5 | 26 | 22 | 26 | PC[3] | | PCINT11 | ADC3 | | | | | | |
| A4 | 27 | 23 | 27 | PC[4] | | PCINT12 | ADC4 | | | | | SDA0 | |
| B4 | 28 | 24 | 28 | PC[5] | | PCINT13 | ADC5 | | | | | SCL0 | |

| (32-pin 32UFBGA) Pin# | (32-pin MLF/ TQFP) Pin# | (28-pin MLF) Pin# | (28-pin PIPD) Pin# | PAD | EXTINT | PCINT | ADC/AC | OSC | T/C #0 | T/C #1 | USART 0 | I2C 0 | SPI 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A3 | 29 | 25 | 1 | PC[6]/ $\overline{\text{RESET}}$ | | PCINT14 | | | | | | | |
| B3 | 30 | 26 | 2 | PD[0] | | PCINT16 | | | | | RXD0 | | |
| A2 | 31 | 27 | 3 | PD[1] | | PCINT17 | | | | | TXD0 | | |
| A1 | 32 | 28 | 4 | PD[2] | INT0 | PCINT18 | | | | | | | |

# 7. Resources

A comprehensive set of development tools, application notes, and datasheets are available for download on http://www.atmel.com/avr.

# 8. 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.

# 9. About 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. Confirm with the C compiler documentation for more details.

For I/O Registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

# 10. Capacitive Touch Sensing

## 10.1. QTouch Library

The Atmel® QTouch® Library provides a simple to use solution to realize touch sensitive interfaces on most Atmel AVR® microcontrollers. The QTouch Library includes support for the Atmel QTouch and Atmel QMatrix® acquisition methods.

Touch sensing can be added to any application by linking the appropriate Atmel QTouch Library for the AVR Microcontroller. This is done by using a simple set of APIs to define the touch channels and sensors, and then calling the touch sensing API's to retrieve the channel information and determine the touch sensor states.

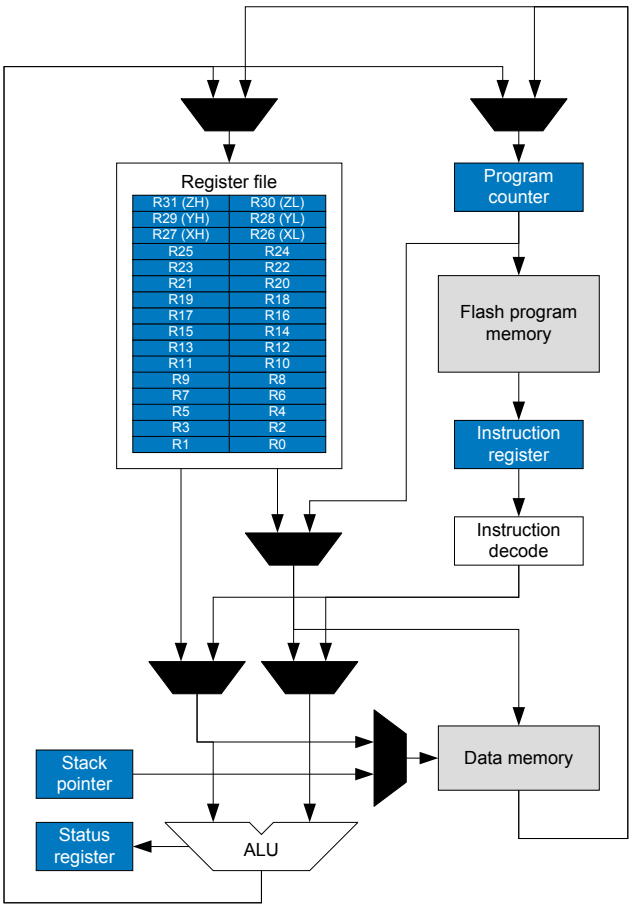
The QTouch Library is FREE and downloadable from the Atmel website at the following location: http://www.atmel.com/technologies/touch/. For implementation details and other information, refer to the Atmel QTouch Library User Guide - also available for download from the Atmel website.

# 11. AVR CPU Core

## 11.1. Overview

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.

**Figure 11-1. 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, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction.

Program Flash memory space is divided in two sections, the Boot Program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM instruction that writes into the Application Flash memory section must reside in the Boot Program section.

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

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

A flexible interrupt module has its control registers in the I/O space with an additional Global Interrupt Enable bit in the Status Register. All interrupts have a separate Interrupt Vector in the Interrupt Vector table. The interrupts have priority in accordance with their Interrupt Vector position. The lower the Interrupt Vector address, the higher 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, this device has Extended I/O space from 0x60 - 0xFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

## 11.2. 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 *Instruction Set Summary* section for a detailed description.

**Related Links**

Instruction Set Summary on page 469

## 11.3. 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. The Status Register is updated after all ALU operations, as specified in the Instruction Set Reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

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

### 11.3.1. Status Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** SREG
**Offset:** 0x5F
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x3F

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | I | T | H | S | V | N | Z | C |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 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: 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 Flag is useful in BCD arithmetic. See the *Instruction Set Description* for detailed information.

**Bit 4 – S: Sign Flag, 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 arithmetic. 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**

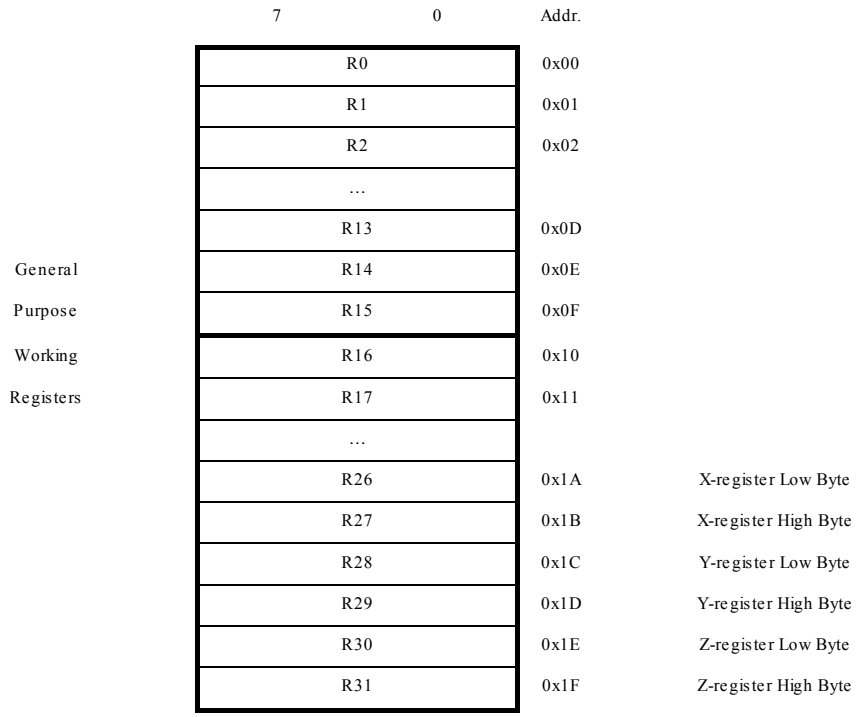
The Carry Flag C indicates a carry in an arithmetic or logic operation. See the *Instruction Set Description* for detailed information.

## 11.4. 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 11-2. AVR CPU General Purpose Working Registers**




Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions. As shown in the figure, 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.

### 11.4.1. The X-register, Y-register, and Z-register

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

**Figure 11-3. The X-, Y-, and Z-registers**

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

**Related Links**

Instruction Set Summary on page 469

## 11.5. Stack Pointer

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

The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. A Stack PUSH command will decrease the Stack Pointer. The Stack in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. Initial Stack Pointer value equals the last address of the internal SRAM and the Stack Pointer must be set to point above start of the SRAM. See the table for Stack Pointer details.

**Table 11-1. Stack Pointer Instructions**

| Instruction | Stack pointer | Description |
|---|---|---|
| PUSH | Decremented by 1 | Data is pushed onto the stack |
| CALL<br>ICALL<br>RCALL | Decremented by 2 | Return address is pushed onto the stack with a subroutine call or interrupt |
| POP | Incremented by 1 | Data is popped from the stack |
| RET<br>RETI | Incremented by 2 | Return address is popped from the stack with return from subroutine or return from interrupt |

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

### 11.5.1. Stack Pointer Register High byte

When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these offset addresses.

**Name:** SPH
**Offset:** 0x5E
**Reset:** RAMEND
**Property:** When addressing I/O Registers as data space the offset address is 0x3E

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | (SP[10:8]) SPH | | |
| Access | | | | | | RW | RW | RW |
| Reset | | | | | | 0 | 0 | 0 |

**Bits 2:0 – (SP[10:8]) SPH: Stack Pointer Register**
SPH and SPL are combined into SP. It means SPH[2:0] is SP[10:8].

### 11.5.2. Stack Pointer Register Low byte

When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these offset addresses.

**Name:** SPL
**Offset:** 0x5D
**Reset:** 0x11111111
**Property:** When addressing I/O Registers as data space the offset address is 0x3D

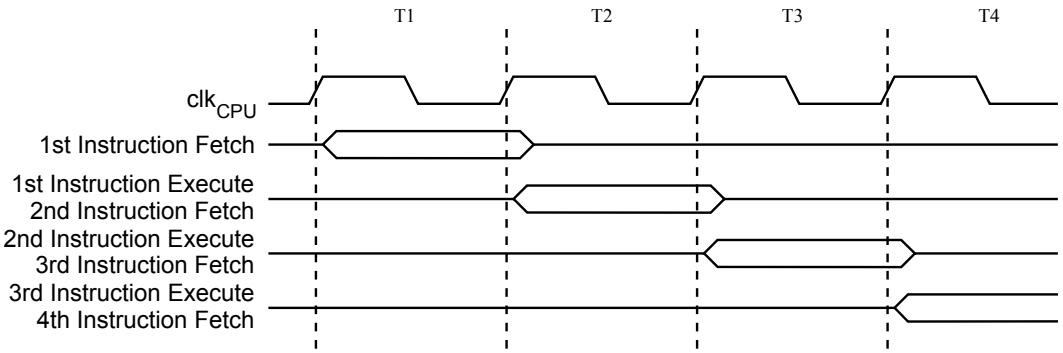
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | (SP[7:0]) SPL | | | | | | | |
| Access | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Bits 7:0 – (SP[7:0]) SPL: Stack Pointer Register**
SPH and SPL are combined into SP. It means SPL[7:0] is SP[7:0].

## 11.6. Instruction Execution Timing

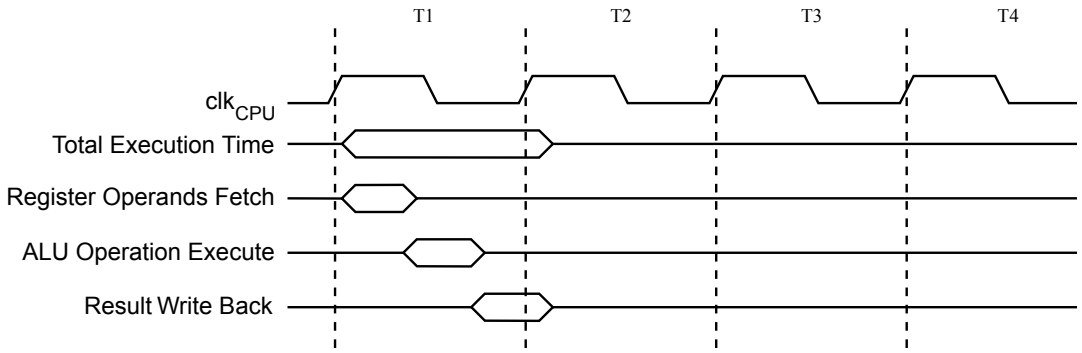
This section describes the general access timing concepts for instruction execution. The AVR CPU is driven by the CPU clock $clk_{CPU}$, directly generated from the selected clock source for the chip. No internal clock division is used. The Figure below shows the parallel instruction fetches and instruction 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 11-4. The Parallel Instruction Fetches and Instruction Executions**




The following Figure 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 11-5. Single Cycle ALU Operation**




## 11.7. 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 Program Counter value, interrupts may be automatically disabled when Boot Lock bits BLB02 or BLB12 are programmed. This feature improves software security.

The lowest addresses in the program memory space are by default defined as the Reset and Interrupt Vectors. They have determined priority levels: 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 Vectors can be moved to the start of the Boot Flash section by setting the IVSEL bit in the MCU Control Register (MCUCR). The Reset Vector can also be moved to the start of the Boot Flash section by programming the BOOTRST Fuse.

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.

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

**Note:** Please refer to *About Code Examples*.

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

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

**Note:** Please refer to *About Code Examples*.

**Related Links**

Memory Programming on page 364

Boot Loader Support – Read-While-Write Self-Programming on page 346

### 11.7.1. 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.

# 12. AVR Memories

## 12.1. Overview

This section describes the different memory types in the device. The AVR architecture has two main memory spaces, the Data Memory and the Program Memory space. In addition, the device features an EEPROM Memory for data storage. All memory spaces are linear and regular.

## 12.2. In-System Reprogrammable Flash Program Memory

The ATmega48A/88A/168A contains 4K/8K/16Kbytes On-chip In-System Reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 2K/4K/8K x 16. For software security, the Flash Program memory space is divided into two sections - Boot Loader Section and Application Program Section in the device (ATmega88A and ATmega168A).

The Flash memory has an endurance of at least 10,000 write/erase cycles. The ATmega48A/88A/168A Program Counter (PC) is 11/12/13 bits wide, thus addressing the 2K/4K/8K program memory locations. The operation of Boot Program section and associated Boot Lock bits for software protection are described in detail in *Self-Programming* for ATmega48A and *Boot Loader Support – Read-While-Write Self-Programming*. Refer to *Memory Programming* for the description on Flash data serial downloading using the SPI pins.

Constant tables can be allocated within the entire program memory address space, using the Load Program Memory (LPM) instruction.

Timing diagrams for instruction fetch and execution are presented in *Instruction Exectution Timing*.

**Figure 12-1. Program Memory Map ATmega48A**

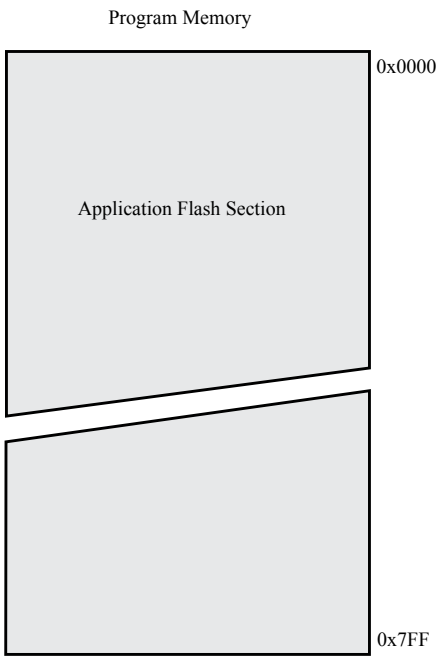


**Figure 12-2. Program Memory Map ATmega88A**

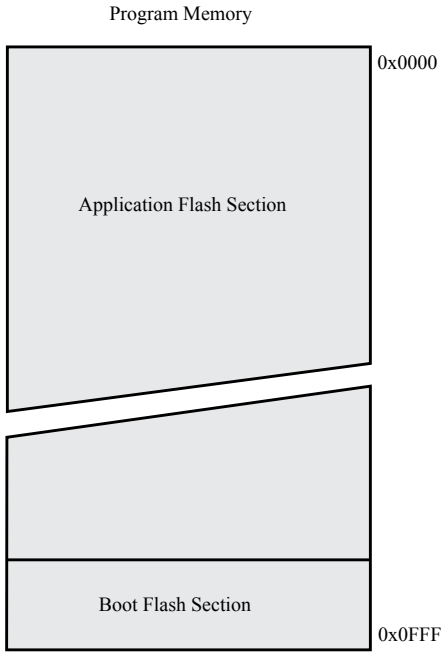
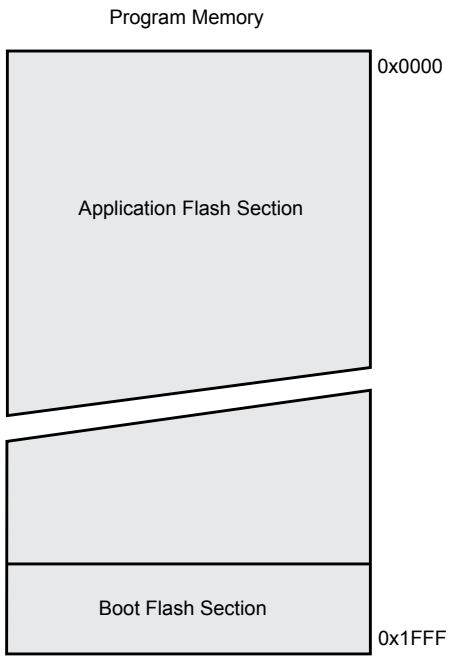



**Figure 12-3. Program Memory Map ATmega168A**




**Related Links**

BTLDR - Boot Loader Support – Read-While-Write Self-Programming on page 346

MEMPROG- Memory Programming on page 364

Instruction Execution Timing on page 33

## 12.3. SRAM Data Memory

The following figure shows how the device SRAM Memory is organized.

The device is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in the Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

The lower 768/1280/1280 data memory locations address both the Register File, the I/O memory, Extended I/O memory, and the internal data SRAM. The first 32 locations address the Register File, the next 64 location the standard I/O memory, then 160 locations of Extended I/O memory, and the next 512/1K/1K locations address the internal data SRAM.

The five different addressing modes for the data memory cover:

- Direct
  - The direct addressing reaches the entire data space.
- Indirect with Displacement
  - The Indirect with Displacement mode reaches 63 address locations from the base address given by the Y- or Z-register.
- Indirect
  - In the Register File, registers R26 to R31 feature the indirect addressing pointer registers.
- Indirect with Pre-decrement
  - The address registers X, Y, and Z are decremented.
- Indirect with Post-increment
  - The address registers X, Y, and Z are incremented.

The 32 general purpose working registers, 64 I/O Registers, 160 Extended I/O Registers, and the 512/1K/1K bytes of internal data SRAM in the device are all accessible through all these addressing modes.

**Figure 12-4. Data Memory Map with 512 byte internal data SRAM**

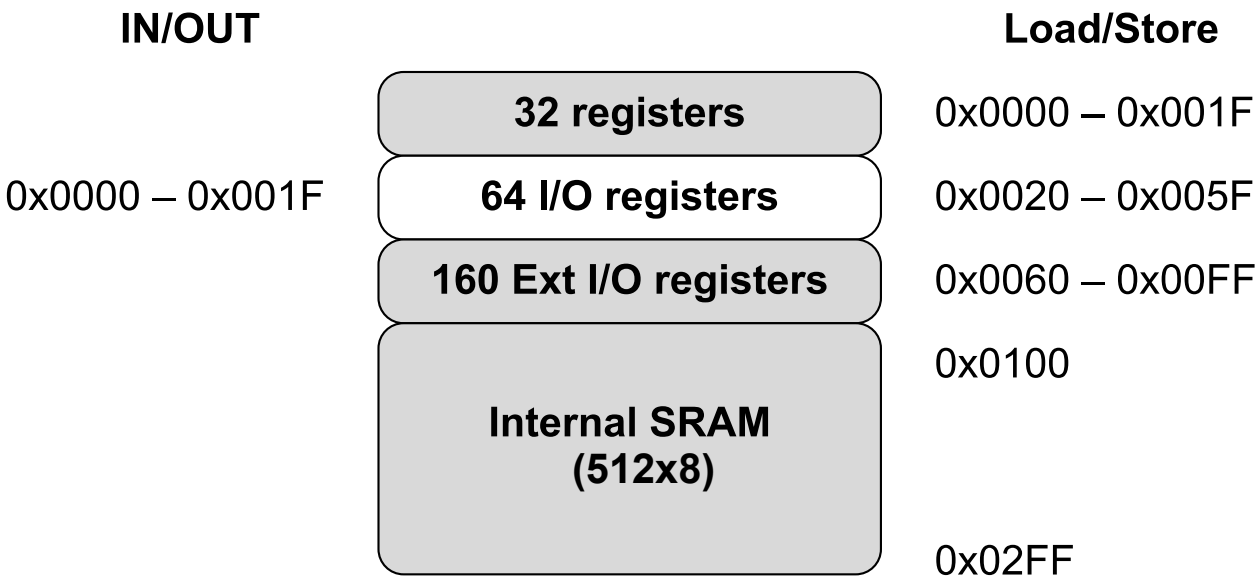
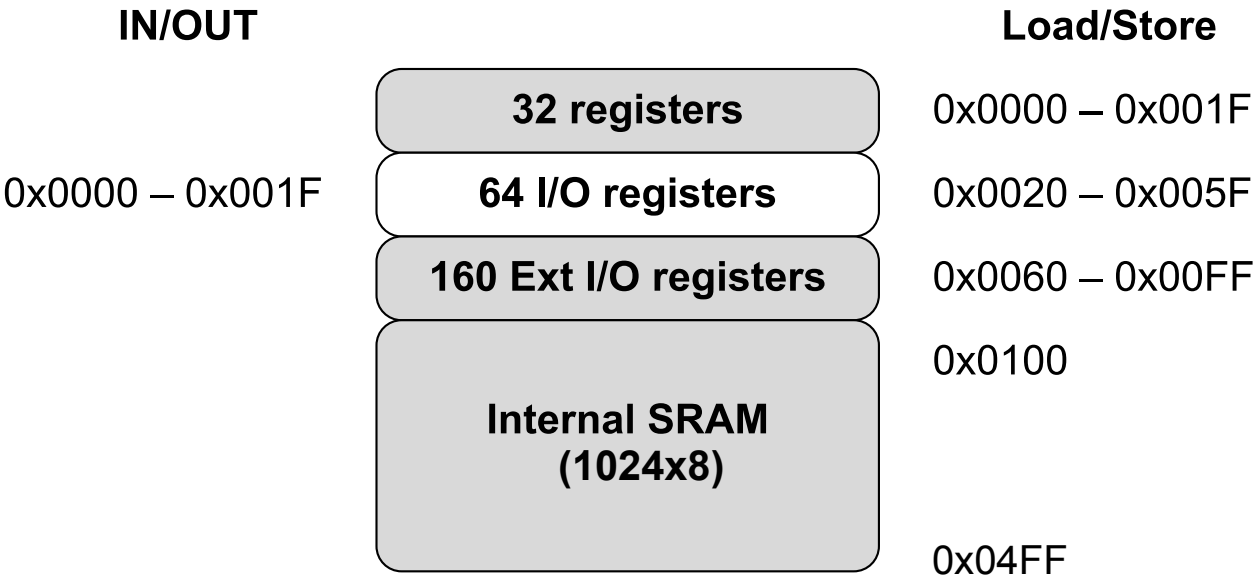


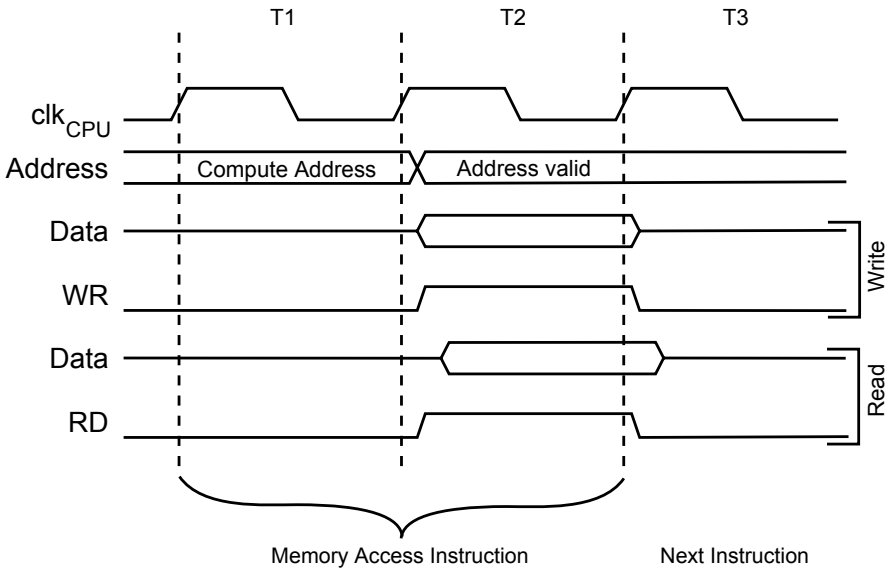
**Figure 12-5. Data Memory Map with 1024 byte internal data SRAM**




### 12.3.1. Data Memory Access Times

The internal data SRAM access is performed in two $clk_{CPU}$ cycles as described in the following Figure.

**Figure 12-6. On-chip Data SRAM Access Cycles**




## 12.4. EEPROM Data Memory

The ATmega48A/88A/168A contains 256/512/512 bytes of data EEPROM memory. It is organized as a separate data space, in which single bytes can be read and written. The EEPROM has an endurance of at least 100,000 write/erase cycles. The access between the EEPROM and the CPU is described in the following, specifying the EEPROM Address Registers, the EEPROM Data Register, and the EEPROM Control Register.

See the related links for a detailed description on EEPROM Programming in SPI or Parallel Programming mode.

**Related Links**

MEMPROG- Memory Programming on page 364

### 12.4.1. EEPROM Read/Write Access

The EEPROM Access Registers are accessible in the I/O space.

The write access time for the EEPROM is given in Table 12-2. A self-timing function, however, lets the user software detect when the next byte can be written. If the user code contains instructions that write the EEPROM, some precautions must be taken. In heavily filtered power supplies, $V_{CC}$ is likely to rise or fall slowly on power-up/down. This causes the device for some period of time to run at a voltage lower than specified as minimum for the clock frequency used. Please refer to Preventing EEPROM Corruption for details on how to avoid problems in these situations.

In order to prevent unintentional EEPROM writes, a specific write procedure must be followed. Refer to the description of the EEPROM Control Register for details on this.

When the EEPROM is read, the CPU is halted for four clock cycles before the next instruction is executed. When the EEPROM is written, the CPU is halted for two clock cycles before the next instruction is executed.

### 12.4.2. 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.

An EEPROM data corruption can be caused by two situations when the voltage is too low. First, a regular write sequence to the EEPROM requires a minimum voltage to operate correctly. Secondly, the CPU itself can execute instructions incorrectly, if the supply voltage is too low.

EEPROM data corruption can easily be avoided by following this design recommendation:

Keep the AVR $\overline{\text{RESET}}$ active (low) during periods of insufficient power supply voltage. This can be done by enabling the internal Brown-out Detector (BOD). If the detection level of the internal BOD does not match the needed detection level, 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.

## 12.5. I/O Memory

The I/O space definition of the device is shown in the *Register Summary*.

All device I/Os and peripherals are placed in the I/O space. All I/O locations may be accessed by the LD/LDS/LDD and ST/STS/STD instructions, transferring data between the 32 general purpose working registers and the I/O space. 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.

When using the I/O specific commands IN and OUT, the I/O addresses 0x00-0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The device is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60..0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

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

Some of the Status Flags are cleared by writing a '1' to them; this is described in the flag descriptions. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such Status Flags. The CBI and SBI instructions work with registers 0x00-0x1F only.

The I/O and Peripherals Control Registers are explained in later sections.

**Related Links**

MEMPROG- Memory Programming on page 364
Register Summary on page 465
Instruction Set Summary on page 469

### 12.5.1. General Purpose I/O Registers

The device contains three General Purpose I/O Registers, General Purpose I/O Register 0/1/2 (GPIOR 0/1/2). These registers can be used for storing any information, and they are particularly useful for storing global variables and Status Flags. General Purpose I/O Registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI, CBI, SBIS, and SBIC instructions.

## 12.6. Register Description

### 12.6.1. EEPROM Address Register High

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** EEARH
**Offset:** 0x42
**Reset:** 0x0X
**Property:** When addressing as I/O Register: address offset is 0x22

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | EEAR9 | EEAR8 |
| Access | | | | | | | R/W | R/W |
| Reset | | | | | | | x | x |

**Bit 1 – EEAR9: EEPROM Address 9**
Refer to EEARL.

It is unused bits and must always be written to zero for ATmega48A.

**Bit 0 – EEAR8: EEPROM Address 8**
Refer to EEARL.

It is unused bits and must always be written to zero for ATmega48A.

### 12.6.2. EEPROM Address Register Low

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** EEARL
**Offset:** 0x41
**Reset:** 0xXX
**Property:** When addressing as I/O Register: address offset is 0x21

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | EEAR7 | EEAR6 | EEAR5 | EEAR4 | EEAR3 | EEAR2 | EEAR1 | EEAR0 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | x | x | x | x | x | x | x | x |

**Bits 7:0 – EEARn: EEPROM Address**

The EEPROM Address Registers – EEARH and EEARL specify the EEPROM address in the 256/512/512 Bytes EEPROM space. The EEPROM data bytes are addressed linearly between 0 and 255/511/511. The initial value of EEAR is undefined. A proper value must be written before the EEPROM may be accessed.

### 12.6.3. EEPROM Data Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** EEDR
**Offset:** 0x40
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x20

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | EEDR[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – EEDR[7:0]: EEPROM Data**

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

### 12.6.4. EEPROM Control Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** EECR
**Offset:** 0x3F
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x1F

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | EEPM1 | EEPM0 | EERIE | EEMPE | EEPE | EERE |
| Access | | | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | | | x | x | 0 | 0 | x | 0 |

**Bits 5:4 – EEPMn: EEPROM Programming Mode Bits [n = 1:0]**

The EEPROM Programming mode bit setting defines which programming action that will be triggered when writing EEPE. It is possible to program data in one atomic operation (erase the old value and program the new value) or to split the Erase and Write operations in two different operations. The Programming times for the different modes are shown in the table below. While 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.

**Table 12-1. EEPROM Mode Bits**

| EEPM[1:0] | Programming Time | Operation |
|---|---|---|
| 00 | 3.4ms | Erase and Write in one operation (Atomic Operation) |
| 01 | 1.8ms | Erase Only |
| 10 | 1.8ms | Write Only |
| 11 | - | Reserved for future use |

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

Writing EERIE to one enables the EEPROM Ready Interrupt if the I bit in SREG is set. Writing EERIE to zero disables the interrupt. The EEPROM Ready interrupt generates a constant interrupt when EEPE is cleared. The interrupt will not be generated during EEPROM write or SPM.

**Bit 2 – EEMPE: EEPROM Master Write Enable**

The EEMPE bit determines whether writing EEPE to '1' causes the EEPROM to be written.
When EEMPE is '1', setting EEPE within four clock cycles will write data to the EEPROM at the selected address.

If EEMPE is zero, setting EEPE will have no effect. When EEMPE has been written to '1' by software, hardware clears the bit to zero after four clock cycles. See the description of the EEPE bit for an EEPROM write procedure.

**Bit 1 – EEPE: EEPROM Write Enable**

The EEPROM Write Enable Signal EEPE is the write strobe to the EEPROM. When address and data are correctly set up, the EEPE bit must be written to '1' to write the value into the EEPROM. The EEMPE bit

must be written to '1' before EEPE is written to '1', otherwise no EEPROM write takes place. The following procedure should be followed when writing the EEPROM (the order of steps 3 and 4 is not essential):

1. Wait until EEPE becomes zero.
2. Wait until SPMEN in SPMCSR becomes zero.
3. Write new EEPROM address to EEAR (optional).
4. Write new EEPROM data to EEDR (optional).
5. Write a '1' to the EEMPE bit while writing a zero to EEPE in EECR.
6. Within four clock cycles after setting EEMPE, write a '1' to EEPE.

The EEPROM can not be programmed during a CPU write to the Flash memory. The software must check that the Flash programming is completed before initiating a new EEPROM write. Step 2 is only relevant if the software contains a Boot Loader allowing the CPU to program the Flash. If the Flash is never being updated by the CPU, step 2 can be omitted.

**Caution:**
An interrupt between step 5 and step 6 will make the write cycle fail, since the EEPROM Master Write Enable will time-out. If an interrupt routine accessing the EEPROM is interrupting another EEPROM access, the EEAR or EEDR Register will be modified, causing the interrupted EEPROM access to fail. It is recommended to have the Global Interrupt Flag cleared during all the steps to avoid these problems.

When the write access time has elapsed, the EEPE bit is cleared by hardware. The user software can poll this bit and wait for a zero before writing the next byte. When EEPE has been set, the CPU is halted for two cycles before the next instruction is executed.

### Bit 0 – EERE: EEPROM Read Enable

The EEPROM Read Enable Signal EERE is the read strobe to the EEPROM. When the correct address is set up in the EEAR Register, the EERE bit must be written to a '1' to trigger the EEPROM read. The 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 is neither possible to read the EEPROM, nor to change the EEAR Register.

The calibrated Oscillator is used to time the EEPROM accesses. See the following table for typical programming times for EEPROM access from the CPU.

**Table 12-2. EEPROM Programming Time**

| Symbol | Number of Calibrated RC Oscillator Cycles | Typ. Programming Time |
|---|---|---|
| EEPROM write (from CPU) | 26,368 | 3.3ms |

The following code examples show one assembly and one C function for writing to the EEPROM. The examples assume that interrupts are controlled (e.g. by disabling interrupts globally) so that no interrupts will occur during execution of these functions. The examples also assume that no Flash Boot Loader is present in the software. If such code is present, the EEPROM write function must also wait for any ongoing SPM command to finish.

**Assembly Code Example(1)**

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

```
      rjmp     EEPROM_write
      ; Set up address (r18:r17) in address register
      out      EEARH, r18
      out      EEARL, r17
      ; Write data (r16) to Data Register
      out      EEDR,r16
      ; Write logical one to EEMPE
      sbi      EECR,EEMPE
      ; Start eeprom write by setting EEPE
      sbi      EECR,EEPE
      ret
```

### C Code Example(1)

```
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
    /* Wait for completion of previous write */
    while(EECR & (1<<EEPE))
    ;
    /* Set up address and Data Registers */
    EEAR = uiAddress;
    EEDR = ucData;
    /* Write logical one to EEMPE */
    EECR |= (1<<EEMPE);
    /* Start eeprom write by setting EEPE */
    EECR |= (1<<EEPE);
}
```

**Note:** (1) Please refer to *About Code Examples*

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(1)

```
EEPROM_read:
   ; Wait for completion of previous write
   sbic     EECR,EEPE
   rjmp     EEPROM_read
   ; Set up address (r18:r17) in address register
   out      EEARH, r18
   out      EEARL, r17
   ; Start eeprom read by writing EERE
   sbi      EECR,EERE
   ; Read data from Data Register
   in       r16,EEDR
   ret
```

### C Code Example(1)

```
unsigned char EEPROM_read(unsigned int uiAddress)
{
    /* Wait for completion of previous write */
    while(EECR & (1<<EEPE))
    ;
    /* Set up address register */
    EEAR = uiAddress;
    /* Start eeprom read by writing EERE */
    EECR |= (1<<EERE);
    /* Return data from Data Register */
    return EEDR;
}
```

**Note:** (1) Please refer to *About Code Examples*

### 12.6.5. GPIOR2 – General Purpose I/O Register 2

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** GPIOR2
**Offset:** 0x4B
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x2B

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | GPIOR2[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – GPIOR2[7:0]: General Purpose I/O**

### 12.6.6. GPIOR1 – General Purpose I/O Register 1

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** GPIOR1
**Offset:** 0x4A
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x2A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | GPIOR1[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – GPIOR1[7:0]: General Purpose I/O**

### 12.6.7. GPIOR0 – General Purpose I/O Register 0

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** GPIOR0
**Offset:** 0x3E
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x1E

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | GPIOR0[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – GPIOR0[7:0]: General Purpose I/O**

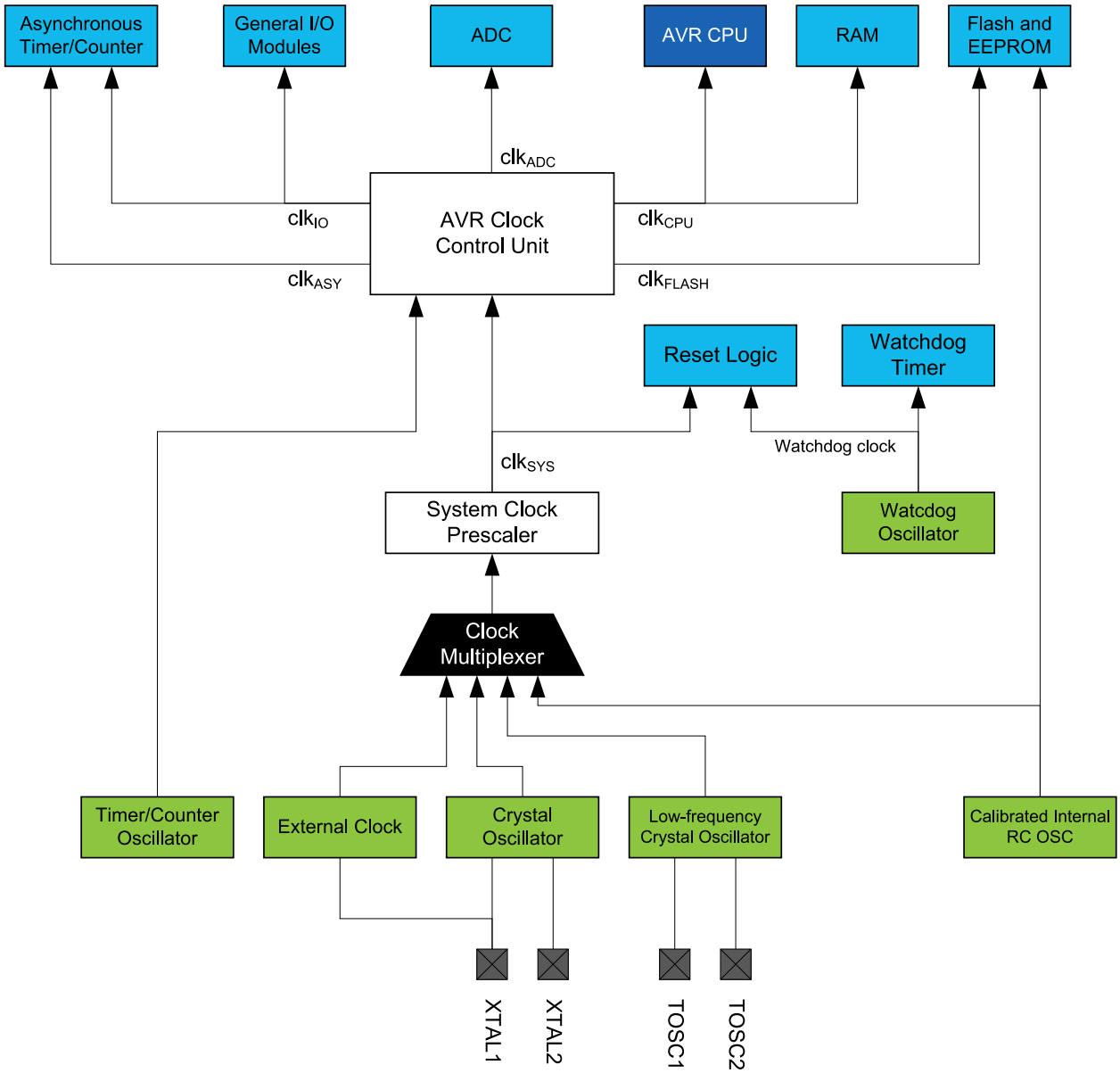
# 13. System Clock and Clock Options

## 13.1. Clock Systems and Their Distribution

The following figure illustrates the principal clock systems in the device and their distribution. All 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. The clock systems are described in the following sections.

The system clock frequency refers to the frequency generated from the System Clock Prescaler. All clock outputs from the AVR Clock Control Unit runs in the same frequency.

**Figure 13-1. Clock Distribution**



### 13.1.1. 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 Status Register and the data memory holding the Stack Pointer. Halting the CPU clock inhibits the core from performing general operations and calculations.

### 13.1.2. I/O Clock – $clk_{I/O}$

The I/O clock is used by the majority of the I/O modules, like Timer/Counters, SPI, and USART. The I/O clock is also used by the External Interrupt module, but the start condition detection in the USI module is carried out asynchronously when $clk_{I/O}$ is halted, TWI address recognition in all sleep modes.

**Note:** 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 by the SUT and CKSEL Fuses.

### 13.1.3. Flash Clock – $clk_{FLASH}$

The Flash clock controls operation of the Flash interface. The Flash clock is usually active simultaneously with the CPU clock.

### 13.1.4. Asynchronous Timer Clock – $clk_{ASY}$

The Asynchronous Timer clock allows Asynchronous Timer/Counters to be clocked directly from an external clock or an external 32kHz clock crystal. The dedicated clock domain allows using this Timer/Counter as a real-time counter even when the device is in sleep mode.

### 13.1.5. 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.

## 13.2. Clock Sources

The device has the following clock source options, selectable by Flash Fuse bits as shown below. The clock from the selected source is input to the AVR clock generator, and routed to the appropriate modules.

**Table 13-1. Device Clocking Options Select**

| Device Clocking Option | CKSEL[3:0] |
|---|---|
| Low Power Crystal Oscillator | 1111 - 1000 |
| Full Swing Crystal Oscillator | 0111 - 0110 |
| Low Frequency Crystal Oscillator | 0101 - 0100 |
| Internal 128kHz RC Oscillator | 0011 |
| Calibrated Internal RC Oscillator | 0010 |
| External Clock | 0000 |
| Reserved | 0001 |

**Note:** For all fuses, '1' means unprogrammed while '0' means programmed.

### 13.2.1. Default Clock Source

The device is shipped with internal RC oscillator at 8.0MHz and with the fuse CKDIV8 programmed, resulting in 1.0MHz system clock. The startup time is set to maximum, and the time-out period is enabled: CKSEL=0010, SUT=10, CKDIV8=0. This default setting ensures that all users can make their desired clock source setting using any available programming interface.

### 13.2.2. Clock Startup Sequence

Any clock source needs a sufficient $V_{CC}$ to start oscillating and a minimum number of oscillating cycles before it can be considered stable.

To ensure sufficient $V_{CC}$, the device issues an internal reset with a time-out delay ($t_{TOUT}$) after the device reset is released by all other reset sources. See the Related Links for a description of the start conditions for the internal reset. The delay ($t_{TOUT}$) is timed from the Watchdog Oscillator and the number of cycles in the delay is set by the SUTx and CKSELx fuse bits. The selectable delays are shown in the Table below. The frequency of the Watchdog Oscillator is voltage dependent.

**Table 13-2. Number of Watchdog Oscillator Cycles**

| Typ. Time-out ($V_{CC}$ = 5.0V) | Typ. Time-out ($V_{CC}$ = 3.0V) | Number of Cycles |
|---|---|---|
| 0ms | 0ms | 0 |
| 4.1ms | 4.3ms | 512 |
| 65ms | 69ms | 8K (8,192) |

Main purpose of the delay is to keep the device in reset until it is supplied with minimum $V_{CC}$. The delay will not monitor the actual voltage, so it is required to select a delay longer than the $V_{CC}$ rise time. If this is not possible, an internal or external Brown-Out Detection circuit should be used. A BOD circuit will ensure sufficient $V_{CC}$ before it releases the reset, and the time-out delay can be disabled. Disabling the time-out delay without utilizing a Brown-Out Detection circuit is not recommended.

The oscillator is required to oscillate for a minimum number of cycles before the clock is considered stable. An internal ripple counter monitors the oscillator output clock, and keeps the internal reset active for a given number of clock cycles. The reset is then released and the device will start to execute. The recommended oscillator start-up time is dependent on the clock type, and varies from 6 cycles for an externally applied clock to 32K cycles for a low frequency crystal.

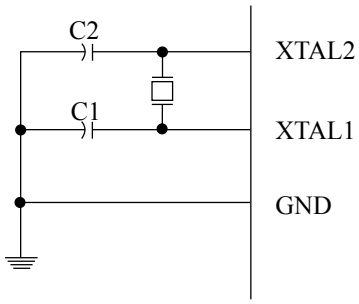
The start-up sequence for the clock includes both the time-out delay and the start-up time when the device starts up from reset. When starting up from Power-save or Power-down mode, $V_{CC}$ is assumed to be at a sufficient level and only the start-up time is included.

### 13.2.3. Low Power Crystal Oscillator

Pins XTAL1 and XTAL2 are input and output, respectively, of an inverting amplifier which can be configured for use as an On-chip Oscillator, as shown in the Figure below. Either a quartz crystal or a ceramic resonator may be used.

C1 and C2 should always be equal for both crystals and resonators. The optimal value of the capacitors depends on the crystal or resonator in use, the amount of stray capacitance, and the electromagnetic noise of the environment. Some initial guidelines for choosing capacitors for use with crystals are given in the next Table. For ceramic resonators, the capacitor values given by the manufacturer should be used.

**Figure 13-2. Crystal Oscillator Connections**




**Related Links**

Low Power Crystal Oscillator on page 54
Full Swing Crystal Oscillator on page 55

## 13.3. Low Power Crystal Oscillator

This Crystal Oscillator is a low power oscillator, with reduced voltage swing on the XTAL2 output. It gives the lowest power consumption, but is not capable of driving other clock inputs, and may be more susceptible to noise in noisy environments. In these cases, refer to Full Swing Crystal Oscillator.

The crystal should be connected as described in *Clock Source Connections*.

The Low Power Oscillator can operate in three different modes, each optimized for a specific frequency range. The operating mode is selected by the fuses CKSEL[3:1], as shown in the following table:

**Table 13-3. Low Power Crystal Oscillator Operating Modes(1)**

| Frequency Range [MHz] | CKSEL[3:1](2) | Range for Capacitors C1 and C2 [pF] |
|---|---|---|
| 0.4 - 0.9 | 100(3) | – |
| 0.9 - 3.0 | 101 | 12 - 22 |
| 3.0 - 8.0 | 110 | 12 - 22 |
| 8.0 - 16.0 | 111 | 12 - 22 |

**Note:**

1. If the crystal frequency exceeds the specification of the device (depends on $V_{CC}$), the CKDIV8 Fuse can be programmed in order to divide the internal frequency by 8. It must be ensured that the resulting divided clock meets the frequency specification of the device.
2. This is the recommended CKSEL settings for the difference frequency ranges.
3. This option should not be used with crystals, only with ceramic resonators.

The CKSEL0 Fuse together with the SUT[1:0] Fuses select the start-up times, as shown in the following table:

**Table 13-4. Start-up Times for the Low Power Crystal Oscillator Clock Selection**

| Oscillator Source / Power Conditions | Start-up Time from Power-down and Power-save | Additional Delay from Reset ($V_{CC}$ = 5.0V) | CKSEL0 | SUT[1:0] |
|---|---|---|---|---|
| Ceramic resonator, fast rising power | 258 CK | 14CK + 4.1ms(1) | 0 | 00 |
| Ceramic resonator, slowly rising power | 258 CK | 14CK + 65ms(1) | 0 | 01 |
| Ceramic resonator, BOD enabled | 1K CK | 14CK(2) | 0 | 10 |
| Ceramic resonator, fast rising power | 1K CK | 14CK + 4.1ms(2) | 0 | 11 |
| Ceramic resonator, slowly rising power | 1K CK | 14CK + 65ms(2) | 1 | 00 |
| Crystal Oscillator, BOD enabled | 16K CK | 14CK | 1 | 01 |
| Crystal Oscillator, fast rising power | 16K CK | 14CK + 4.1ms | 1 | 10 |
| Crystal Oscillator, slowly rising power | 16K CK | 14CK + 65ms | 1 | 11 |

**Note:**

1. These options should only be used when not operating close to the maximum frequency of the device, and only if frequency stability at start-up is not important for the application. These options are not suitable for crystals.
2. These options are intended for use with ceramic resonators and will ensure frequency stability at start-up. They can also be used with crystals when not operating close to the maximum frequency of the device, and if frequency stability at start-up is not important for the application.

**Related Links**

Low Power Crystal Oscillator on page 53

## 13.4. Full Swing Crystal Oscillator

This Crystal Oscillator is a full swing oscillator, with rail-to-rail swing on the XTAL2 output. This is useful for driving other clock inputs and in noisy environments. The current consumption is higher than for the Low Power Crystal Oscillator. Note that the Full Swing Crystal Oscillator will only operate for $V_{CC}$=2.7-5.5V.

Some initial guidelines for choosing capacitors for use with crystals are given in Table 13-6. The crystal should be connected as described in *Clock Source Connections*".

The operating mode is selected based on the fuses CKSEL[3:1] as shown in the table:

**Table 13-5. Full Swing Crystal Oscillator operating modes**

| Frequency Range(1) [MHz] | CKSEL[3:1] | Recommended Range for Capacitors C1 and C2 [pF] |
|---|---|---|
| 0.4 - 20 | 011 | 12 - 22 |

**Note:**

1. If the crystal frequency exceeds the specification of the device (depends on $V_{CC}$), the CKDIV8 Fuse can be programmed in order to divide the internal frequency by 8. It must be ensured that the resulting divided clock meets the frequency specification of the device.

For the Crystall Oscillator connections refer to Low Power Crystal Oscillator.

**Table 13-6. Start-Up Times for the Full Swing Crystal Oscillator Clock Selection**

| Oscillator Source / Power Conditions | Start-Up Time from Power-down and Power-save | Additional Delay from Reset ($V_{CC}$ = 5.0V) | CKSEL0 | SUT[1:0] |
|---|---|---|---|---|
| Ceramic resonator, fast rising power | 258 CK | 14CK + 4.1ms(1) | 0 | 00 |
| Ceramic resonator, slowly rising power | 258 CK | 14CK + 65ms(1) | 0 | 01 |
| Ceramic resonator, BOD enabled | 1K CK | 14CK(2) | 0 | 10 |
| Ceramic resonator, fast rising power | 1K CK | 14CK + 4.1ms(2) | 0 | 11 |
| Ceramic resonator, slowly rising power | 1K CK | 14CK + 65ms(2) | 1 | 00 |
| Crystal Oscillator, BOD enabled | 16K CK | 14CK | 1 | 01 |
| Crystal Oscillator, fast rising power | 16K CK | 14CK + 4.1ms | 1 | 10 |
| Crystal Oscillator, slowly rising power | 16K CK | 14CK + 65ms | 1 | 11 |

**Note:**

1. These options should only be used when not operating close to the maximum frequency of the device, and only if frequency stability at start-up is not important for the application. These options are not suitable for crystals.
2. These options are intended for use with ceramic resonators and will ensure frequency stability at start-up. They can also be used with crystals when not operating close to the maximum frequency of the device, and if frequency stability at start-up is not important for the application.

**Related Links**

Low Power Crystal Oscillator on page 53

## 13.5. Low Frequency Crystal Oscillator

The Low-frequency Crystal Oscillator is optimized for use with a 32.768kHz watch crystal. When selecting crystals, load capacitance and crystal's Equivalent Series Resistance (ESR) must be taken into consideration. Both values are specified by the crystal vendor. The oscillator is optimized for very low power consumption, and thus when selecting crystals, consider the Maximum ESR Recommendations:

**Table 13-7. Maximum ESR Recommendation for 32.768kHz Crystal**

| Crystal CL [pF] | Max. ESR [kΩ](1) |
|---|---|
| 6.5 | 75 |
| 9.0 | 65 |
| 12.5 | 30 |

**Note:**

1. Maximum ESR is typical value based on characterization.

The Low-frequency Crystal Oscillator provides an internal load capacitance at each TOSC pin:

**Table 13-8. Capacitance for Low-Frequency Oscillator**

| 32kHz Osc. Type | Cap. (XTAL1/TOSC1) | Cap. (XTAL2/TSOC2) |
|---|---|---|
| System Osc. | 18pF | 8pF |
| Timer Osc. | 6pF | 6pF |

The capacitance (Ce+Ci) needed at each TOSC pin can be calculated by using:

$C = 2CL - C_s$

where:

- Ce - is optional external capacitors as described in Figure 13-2.
- Ci - is the pin capacitance in the above table.
- CL - is the load capacitance for a 32.768kHz crystal specified by the crystal vendor.
- $C_S$ - is the total stray capacitance for one TOSC pin.

Crystals specifying a load capacitance (CL) higher than 6pF require external capacitors applied as described in Low Power Crystal Oscillator.

The Low-frequency Crystal Oscillator must be selected by setting the CKSEL Fuses to 0110 or 0111.

**Table 13-9. Start-up Times for the Low-frequency Crystal Oscillator Clock Selection**

| CKSEL[3:0] | Start-up Time from Power-down and Power-save | Recommended Usage |
|---|---|---|
| 0100(1) | 1K CK | |
| 0101 | 32K CK | Stable frequency at start-up |

**Note:**

1. This option should only be used if frequency stability at start-up is not important for the application

Start-up times are determined by the SUT Fuses as shown in the following table.

**Table 13-10. Start-up Times for the Low Frequency Crystal Oscillator Clock Selection**

| SUT[1:0] | Additional Delay from Reset ($V_{CC}$ = 5.0V) | Power Conditions |
|---|---|---|
| 00 | 14CK | BOD enabled |
| 01 | 14CK + 4.1 ms | Fast rising power |
| 10 | 14CK + 65 ms | Slowly rising power |
| 11 | Reserved | |

**Related Links**

Timer/Counter Oscillator on page 60

## 13.6. Calibrated Internal RC Oscillator

By default, the Internal RC Oscillator provides an 8.0MHz clock. Though voltage and temperature dependent, this clock can be very accurately calibrated by the user. The device is shipped with the CKDIV8 Fuse programmed.

This clock may be selected as the system clock by programming the CKSEL Fuses as shown in the following Table. If selected, it will operate with no external components. During reset, hardware loads the

pre-programmed calibration value into the OSCCAL Register and thereby automatically calibrates the RC Oscillator.

By changing the OSCCAL register from SW, it is possible to get a higher calibration accuracy than by using the factory calibration.

When this Oscillator is used as the chip clock, the Watchdog Oscillator will still be used for the Watchdog Timer and for the Reset Time-Out. For more information on the pre-programmed calibration value.

**Table 13-11. Internal Calibrated RC Oscillator Operating Modes**

| Frequency Range(1) [MHz] | CKSEL[3:0] |
|---|---|
| 7.3 - 8.1 | 0010(2) |

**Note:**

1. If 8MHz frequency exceeds the specification of the device (depends on $V_{CC}$), the CKDIV8 Fuse can be programmed in order to divide the internal frequency by 8.
2. The device is shipped with this option selected.

When this Oscillator is selected, start-up times are determined by the SUT Fuses:

**Table 13-12. Start-Up Times for the Internal Calibrated RC Oscillator Clock Selection - SUT**

| Power Conditions | Start-Up Time from Power-down and Power-Save | Additional Delay from Reset ($V_{CC}$ = 5.0V) | SUT[1:0] |
|---|---|---|---|
| BOD enabled | 6 CK | 14CK | 00 |
| Fast rising power | 6 CK | 14CK + 4.1ms | 01 |
| Slowly rising power | 6 CK | 14CK + 65ms | 10(1) |
| Reserved | | | 11 |

**Note:**

1. The device is shipped with this option selected.

**Related Links**

Clock Characteristics on page 387
System Clock Prescaler on page 60
Calibration Byte on page 368
OSCCAL on page 62

## 13.7. 128kHz Internal Oscillator

The 128kHz internal Oscillator is a low power Oscillator providing a clock of 128kHz. The frequency is nominal at 3V and 25°C. This clock may be select as the system clock by programming the CKSEL Fuses to '0011':

**Table 13-13. 128kHz Internal Oscillator Operating Modes**

| Nominal Frequency(1) | CKSEL[3:0] |
|---|---|
| 128kHz | 0011 |

**Note:**

1. The 128kHz oscillator is a very low power clock source, and is not designed for high accuracy.

When this clock source is selected, start-up times are determined by the SUT Fuses:

**Table 13-14. Start-Up Times for the 128kHz Internal Oscillator**

| Power Conditions | Start-Up Time from Power-down and Power-save | Additional Delay from Reset | SUT[1:0] |
|---|---|---|---|
| BOD enabled | 6 CK | 14CK | 00 |
| Fast rising power | 6 CK | 14CK + 4ms | 01 |
| Slowly rising power | 6 CK | 14CK + 64ms | 10 |
| Reserved | | | 11 |

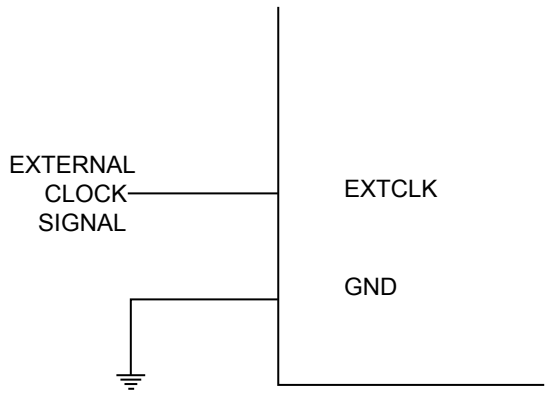
## 13.8. External Clock

To drive the device from an external clock source, EXTCLK should be driven as shown in the Figure below. To run the device on an external clock, the CKSEL Fuses must be programmed to '0000':

**Table 13-15. External Clock Frequency**

| Frequency | CKSEL[3:0] |
|---|---|
| 0 - 20MHz | 0000 |

**Figure 13-3. External Clock Drive Configuration**




When this clock source is selected, start-up times are determined by the SUT Fuses:

**Table 13-16. Start-Up Times for the External Clock Selection - SUT**

| Power Conditions | Start-Up Time from Power-down and Power-save | Additional Delay from Reset ($V_{CC}$ = 5.0V) | SUT[1:0] |
|---|---|---|---|
| BOD enabled | 6 CK | 14CK | 00 |
| Fast rising power | 6 CK | 14CK + 4.1ms | 01 |
| Slowly rising power | 6 CK | 14CK + 65ms | 10 |
| Reserved | | | 11 |

When applying an external clock, it is required to avoid sudden changes in the applied clock frequency to ensure stable operation of the MCU. A variation in frequency of more than 2% from one clock cycle to the

next can lead to unpredictable behavior. If changes of more than 2% is required, ensure that the MCU is kept in Reset during the changes.

The System Clock Prescaler can be used to implement run-time changes of the internal clock frequency while still ensuring stable operation.

**Related Links**

System Clock Prescaler on page 60

## 13.9. Timer/Counter Oscillator

The device uses the same crystal oscillator for Low-frequency Oscillator and Timer/Counter Oscillator. See Low Frequency Crystal Oscillator for details on the oscillator and crystal requirements.

On this device, the Timer/Counter Oscillator Pins (TOSC1 and TOSC2) are shared with XTAL1 and XTAL2. When using the Timer/Counter Oscillator, the system clock needs to be four times the oscillator frequency. Due to this and the pin sharing, the Timer/Counter Oscillator can only be used when the Calibrated Internal RC Oscillator is selected as system clock source.

Applying an external clock source to TOSC1 can be done if the Enable External Clock Input bit in the Asynchronous Status Register (ASSR.EXCLK) is written to '1'. See the description of the Asynchronous Operation of Timer/Counter2 for further description on selecting external clock as input instead of a 32.768kHz watch crystal.

**Related Links**

OCR2B on page 217

ASSR on page 220

Low Frequency Crystal Oscillator on page 56

## 13.10. 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. The clock also will be output during reset, and the normal operation of I/O pin will be overridden when the fuse is programmed. Any clock source, including the internal RC Oscillator, 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.

## 13.11. System Clock Prescaler

The device has a system clock prescaler, and the system clock can be divided by configuring the Clock Prescale Register (CLKPR). This feature can be used to decrease the system clock frequency and the 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 the CLKPR description.

When switching between prescaler settings, the System Clock Prescaler ensures that no glitches occurs in the clock system. It also ensures 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, the exact time it takes to switch from one clock division to the other cannot be exactly predicted. From the time the Clock Prescaler Selection bits (CLKPS[3:0]) values

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

To avoid unintentional changes of clock frequency, a special write procedure must be followed to change the CLKPS bits:

1. Write the Clock Prescaler Change Enable (CLKPCE) bit to '1' and all other bits in CLKPR to zero: CLKPR=0x80.
2. Within four cycles, write the desired value to CLKPS[3:0] while writing a zero to CLKPCE: CLKPR=0x0N

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

**Related Links**

Calibrated Internal RC Oscillator on page 57

External Clock on page 59

CLKPR on page 63

## 13.12. Register Description

### 13.12.1. Oscillator Calibration Register

**Name:** OSCCAL
**Offset:** 0x66
**Reset:** Device Specific Calibration Value
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CAL7 | CAL6 | CAL5 | CAL4 | CAL3 | CAL2 | CAL1 | CAL0 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | x | x | x | x | x | x | x | x |

**Bits 7:0 – CALn: Oscillator Calibration Value [n = 7:0]**
The Oscillator Calibration Register is used to trim the Calibrated Internal RC Oscillator 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 the *Clock Characteristics* section of *Electrical Characteristics* chapter.. The application software can write this register to change the oscillator frequency. The oscillator can be calibrated to frequencies as specified in the *Clock Characteristics* section of *Electrical Characteristics* chapter.. Calibration outside that range is not guaranteed.

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

The CAL7 bit determines the range of operation for the oscillator. Setting this bit to 0 gives the lowest frequency range, setting this bit to 1 gives the highest frequency range. The two frequency ranges are overlapping, in other words a setting of OSCCAL=0x7F gives a higher frequency than OSCCAL=0x80.

The CAL[6:0] bits are used to tune the frequency within the selected range. A setting of 0x00 gives the lowest frequency in that range, and a setting of 0x7F gives the highest frequency in the range.

### 13.12.2. Clock Prescaler Register

**Name:** CLKPR
**Offset:** 0x61
**Reset:** Refer to the bit description
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CLKPCE | | | | CLKPS3 | CLKPS2 | CLKPS1 | CLKPS0 |
| Access | R/W | | | | R/W | R/W | R/W | R/W |
| Reset | 0 | | | | x | x | x | x |

**Bit 7 – CLKPCE: Clock Prescaler Change Enable**
The CLKPCE bit must be written to logic one to enable change of the CLKPS bits. The CLKPCE bit is only updated when the other bits in CLKPR are simultaneously written to zero. CLKPCE is cleared by hardware four cycles after it is written or when CLKPS bits are written. Rewriting the CLKPCE bit within this time-out period does neither extend the time-out period, nor clear the CLKPCE bit.

**Bits 3:0 – CLKPSn: Clock Prescaler Select n [n = 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 the table below.

The CKDIV8 Fuse determines the initial value of the CLKPS bits. If CKDIV8 is unprogrammed, the CLKPS bits will be reset to “0000”. If CKDIV8 is programmed, CLKPS bits are reset to “0011”, giving a division factor of 8 at start up. This feature should be used if the selected clock source has a higher frequency than the maximum frequency of the device at the present operating conditions. Note that any value can be written to the CLKPS bits regardless of the CKDIV8 Fuse setting. The Application software must ensure that a sufficient division factor is chosen if the selected clock source has a higher frequency than the maximum frequency of the device at the present operating conditions. The device is shipped with the CKDIV8 Fuse programmed.

**Table 13-17. Clock Prescaler Select**

| CLKPS[3:0] | Clock Division Factor |
|---|---|
| 0000 | 1 |
| 0001 | 2 |
| 0010 | 4 |
| 0011 | 8 |
| 0100 | 16 |
| 0101 | 32 |
| 0110 | 64 |
| 0111 | 128 |
| 1000 | 256 |
| 1001 | Reserved |

| CLKPS[3:0] | Clock Division Factor |
|---|---|
| 1010 | Reserved |
| 1011 | Reserved |
| 1100 | Reserved |
| 1101 | Reserved |
| 1110 | Reserved |
| 1111 | Reserved |

# 14. PM - Power Management and Sleep Modes

## 14.1. Overview

Sleep modes enable the application to shut down unused modules in the MCU, thereby saving power. The device provides various sleep modes allowing the user to tailor the power consumption to the application requirements.

## 14.2. Sleep Modes

The following Table shows the different sleep modes and their wake-up sources.

**Table 14-1. Active Clock Domains and Wake-up Sources in the Different Sleep Modes.**

| Sleep Mode | Active Clock Domains | | | | | Oscillators | | Wake-up Sources | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $clk_{CPU}$ | $clk_{FLASH}$ | $clk_{IO}$ | $clk_{ADC}$ | $clk_{ASY}$ | Main Clock Source Enabled | Timer Oscillator Enabled | INT and PCINT | TWI Address Match | Timer2 | SPM/EEPROM Ready | ADC | WDT | Other I/O |
| Idle | | | Yes | Yes | Yes | Yes | Yes(2) | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| ADC Noise Reduction | | | | Yes | Yes | Yes | Yes(2) | Yes(3) | Yes | Yes(2) | Yes | Yes | Yes | |
| Power-down | | | | | | | | Yes(3) | Yes | | | | Yes | |
| Power-save | | | | | Yes | Yes | Yes(2) | Yes(3) | Yes | Yes | | | Yes | |
| Standby(1) | | | | | | Yes | | Yes(3) | Yes | | | | Yes | |
| Extended Standby | | | | | Yes(2) | Yes | Yes(2) | Yes(3) | Yes | Yes | | | Yes | |

**Note:**

1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

To enter any of the six sleep modes, the Sleep Enable bit in the Sleep Mode Control Register (SMCR.SE) must be written to '1' and a SLEEP instruction must be executed. Sleep Mode Select bits (SMCR.SM[2:0]) select which sleep mode (Idle, ADC Noise Reduction, Power-down, Power-save, Standby, or Extended Standby) will be activated by the SLEEP instruction.

**Note:** The block diagram in the section *System Clock and Clock Options* provides an overview over the different clock systems in the device, and their distribution. This figure is helpful in selecting an appropriate sleep mode.

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.

**Related Links**

System Clock and Clock Options on page 51

## 14.3. Idle Mode

When the SM[2:0] bits are written to '000', the SLEEP instruction makes the MCU enter Idle mode, stopping the CPU but allowing the SPI, USART, Analog Comparator, 2-wire Serial Interface, Timer/

Counters, Watchdog, and the interrupt system to continue operating. This sleep mode basically halts $clk_{CPU}$ and $clk_{FLASH}$, while allowing the other clocks to run.

Idle mode enables the MCU to wake up from external triggered interrupts as well as internal ones like the Timer Overflow and USART Transmit Complete interrupts. If wake-up from the Analog Comparator interrupt is not required, the Analog Comparator can be powered down by setting the ACD bit in the Analog Comparator Control and Status Register – ACSR. This will reduce power consumption in Idle mode.

## 14.4. ADC Noise Reduction Mode

When the SM[2:0] bits are written to '001', the SLEEP instruction makes the MCU enter ADC Noise Reduction mode, stopping the CPU but allowing the ADC, the external interrupts, the 2-wire Serial Interface address watch, Timer/Counter2[1], and the Watchdog to continue operating (if enabled). This sleep mode basically halts $clk_{I/O}$, $clk_{CPU}$, and $clk_{FLASH}$, while allowing the other clocks to run.

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. Apart from the ADC Conversion Complete interrupt, only these events can wake up the MCU from ADC Noise Reduction mode:

- External Reset
- Watchdog System Reset
- Watchdog Interrupt
- Brown-out Reset
- 2-wire Serial Interface address match
- Timer/Counter2 interrupt
- SPM/EEPROM ready interrupt
- External level interrupt on INT
- Pin change interrupt

**Note:** 1. Timer/Counter2 will only keep running in asynchronous mode.

**Related Links**

8-bit Timer/Counter2 with PWM and Asynchronous Operation on page 196

## 14.5. Power-Down Mode

When the SM[2:0] bits are written to '010', the SLEEP instruction makes the MCU enter Power-Down mode. In this mode, the external Oscillator is stopped, while the external interrupts, the 2-wire Serial Interface address watch, and the Watchdog continue operating (if enabled).

Only one of these events can wake up the MCU:

- External Reset
- Watchdog System Reset
- Watchdog Interrupt
- Brown-out Reset
- 2-wire Serial Interface address match
- External level interrupt on INT
- Pin change interrupt

This sleep mode basically halts all generated clocks, allowing operation of asynchronous modules only.

**Note:** 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 by the SUT and CKSEL Fuses.

When waking up from Power-Down mode, there is a delay from the wake-up condition occurs until the wake-up becomes effective. This allows the clock to restart and become stable after having been stopped. The wake-up period is defined by the same CKSEL Fuses that define the Reset Time-out period.

**Related Links**

System Clock and Clock Options on page 51

## 14.6. Power-save Mode

When the SM[2:0] bits are written to 011, the SLEEP instruction makes the MCU enter Power-save mode. This mode is identical to Power-down, with one exception:

If Timer/Counter2 is enabled, it will keep running during sleep. The device can wake up from either Timer Overflow or Output Compare event from Timer/Counter2 if the corresponding Timer/Counter2 interrupt enable bits are set in TIMSK2, and the Global Interrupt Enable bit in SREG is set.

If Timer/Counter2 is not running, Power-down mode is recommended instead of Power-save mode.

The Timer/Counter2 can be clocked both synchronously and asynchronously in Power-save mode. If Timer/Counter2 is not using the asynchronous clock, the Timer/Counter Oscillator is stopped during sleep. If Timer/Counter2 is not using the synchronous clock, the clock source is stopped during sleep. Even if the synchronous clock is running in Power-save, this clock is only available for Timer/Counter2.

## 14.7. Standby Mode

When the SM[2:0] bits are written to '110' and an external crystal/resonator clock option is selected, the SLEEP instruction makes the MCU enter Standby mode. This mode is identical to Power-Down with the exception that the Oscillator is kept running. From Standby mode, the device wakes up in six clock cycles.

## 14.8. Extended Standby Mode

When the SM[2:0] bits are written to '111' and an external crystal/resonator clock option is selected, the SLEEP instruction makes the MCU enter Extended Standby mode. This mode is identical to Power-Save mode with the exception that the Oscillator is kept running. From Extended Standby mode, the device wakes up in six clock cycles.

## 14.9. Power Reduction Register

The Power Reduction Register (PRR) provides a method to stop the clock to individual peripherals to reduce power consumption. The current state of the peripheral is frozen and the I/O registers can not be read or written. Resources used by the peripheral when stopping the clock will remain occupied, hence the peripheral should in most cases be disabled before stopping the clock. Waking up a module, which is done by clearing the corresponding bit in the PRR, puts the module in the same state as before shutdown.

Module shutdown can be used in Idle mode and Active mode to significantly reduce the overall power consumption. In all other sleep modes, the clock is already stopped.

## 14.10. Minimizing Power Consumption

There are several possibilities 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.

### 14.10.1. 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.

**Related Links**

Analog-to-Digital Converter on page 312

### 14.10.2. 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 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.

**Related Links**

Analog Comparator on page 306

### 14.10.3. Brown-Out Detector

If the Brown-Out Detector (BOD) is not needed by the application, this module should be turned off. If the BOD is enabled by the BODLEVEL 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.

**Related Links**

System Control and Reset on page 74

### 14.10.4. Internal Voltage Reference

The Internal Voltage Reference will be enabled when needed by the Brown-Out Detection, the Analog Comparator or the Analog-to-Digital Converter. 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.

**Related Links**

System Control and Reset on page 74

### 14.10.5. Watchdog Timer

If the Watchdog Timer is not needed in the application, the 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 significantly to the total current consumption.

**Related Links**

System Control and Reset on page 74

### 14.10.6. Port Pins

When entering a sleep mode, all port pins should be configured to use minimum power. The most important 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. Refer to the section *Digital Input Enable and Sleep Modes* for details on which pins are enabled. If the input buffer is enabled and the input signal is left floating or have 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 Registers (DIDR0 for ADC, DIDR1 for AC).

**Related Links**

Digital Input Enable and Sleep Modes on page 108

### 14.10.7. On-chip Debug System

If the On-chip debug system is enabled by the Fuse and the chip enters sleep mode, the main clock source is enabled and hence always consumes power. In the deeper sleep modes, this will contribute significantly to the total current consumption.

## 14.11. Register Description

### 14.11.1. Sleep Mode Control Register

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

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** SMCR
**Offset:** 0x53
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x33

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | SM2 | SM1 | SM0 | SE |
| Access | | | | | R/W | R/W | R/W | R/W |
| Reset | | | | | 0 | 0 | 0 | 0 |

**Bit 3 – SM2: Sleep Mode Select 2**
The SM[2:0] bits select between the five available sleep modes.

**Table 14-2. Sleep Mode Select**

| SM2,SM1,SM0 | Sleep Mode |
|---|---|
| 000 | Idle |
| 001 | ADC Noise Reduction |
| 010 | Power-down |
| 011 | Power-save |
| 100 | Reserved |
| 101 | Reserved |
| 110 | Standby(1) |
| 111 | Extended Standby(1) |

**Note:**
1. Standby mode is only recommended for use with external crystals or resonators.

**Bit 2 – SM1: Sleep Mode Select 1**
Refer to SM2.

**Bit 1 – SM0: Sleep Mode Select 0**
Refer to SM2.

**Bit 0 – SE: Sleep Enable**
The SE 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 the sleep mode unless it is the programmer's purpose, 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.

### 14.11.2. MCU Control Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** MCUCR
**Offset:** 0x55
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x35

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | PUD | | | IVSEL | IVCE |
| Access | | | | R/W | | | R/W | R/W |
| Reset | | | | 0 | | | 0 | 0 |

**Bit 4 – PUD: Pull-up Disable**
When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01).

**Bit 1 – IVSEL: Interrupt Vector Select**
When the IVSEL bit is cleared (zero), the Interrupt Vectors are placed at the start of the Flash memory. When this bit is set (one), the Interrupt Vectors are moved to the beginning of the Boot Loader section of the Flash. The actual address of the start of the Boot Flash Section is determined by the BOOTSZ Fuses. To avoid unintentional changes of Interrupt Vector tables, a special write procedure must be followed to change the IVSEL bit:

1. Write the Interrupt Vector Change Enable (IVCE) bit to one.
2. Within four cycles, write the desired value to IVSEL while writing a zero to IVCE.

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

**Note:** If Interrupt Vectors are placed in the Boot Loader section and Boot Lock bit BLB02 is programmed, interrupts are disabled while executing from the Application section. If Interrupt Vectors are placed in the Application section and Boot Lock bit BLB12 is programed, interrupts are disabled while executing from the Boot Loader section.

**Bit 0 – IVCE: Interrupt Vector Change Enable**
The IVCE bit must be written to logic one to enable change of the IVSEL bit. IVCE is cleared by hardware four cycles after it is written or when IVSEL is written. Setting the IVCE bit will disable interrupts, as explained in the IVSEL description above. See Code Example below.

### Assembly Code Example

```
Move_interrupts:
; Get MCUCR
in    r16, MCUCR
mov   r17, r16
; Enable change of Interrupt Vectors
ori   r16, (1<<IVCE)
out   MCUCR, r16
; Move interrupts to Boot Flash section
ori   r17, (1<<IVSEL)
out   MCUCR, r17
ret
```

### C Code Example

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

### 14.11.3. Power Reduction Register

**Name:** PRR
**Offset:** 0x64
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | PRTWI0 | PRTIM2 | PRTIM0 | | PRTIM1 | PRSPI0 | PRUSART0 | PRADC |
| Access | R/W | R/W | R/W | | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | | 0 | 0 | 0 | 0 |

**Bit 7 – PRTWI0: Power Reduction TWI0**
Writing a logic one to this bit shuts down the TWI 0 by stopping the clock to the module. When waking up the TWI again, the TWI should be re initialized to ensure proper operation.

**Bit 6 – PRTIM2: Power Reduction Timer/Counter2**
Writing a logic one to this bit shuts down the Timer/Counter2 module in synchronous mode (AS2 is 0). When the Timer/Counter2 is enabled, operation will continue like before the shutdown.

**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 – PRSPI0: Power Reduction Serial Peripheral Interface 0**
If using debugWIRE On-chip Debug System, this bit should not be written to one. Writing a logic one to this bit shuts down the Serial Peripheral Interface 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 USART by stopping the clock to the module. When waking up the USART again, the USART should be re initialized to ensure proper operation.

**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 use the ADC input MUX when the ADC is shut down.

# 15. SCRST - System Control and Reset

## 15.1. 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 an Absolute Jump instruction (JMP) to the reset handling routine for ATmega168A. The instruction placed at the Reset Vector must be an Relative Jump instruction (RJMP) to the reset handling routine for ATmega48A and ATmega88A. 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 (only for ATmega88Aand ATmega168A). The circuit diagram in the next section shows the 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. The time-out period of the delay counter is defined by the user through the SUT and CKSEL Fuses. The different selections for the delay period are presented in the *System Clock and Clock Options* chapter.

**Related Links**

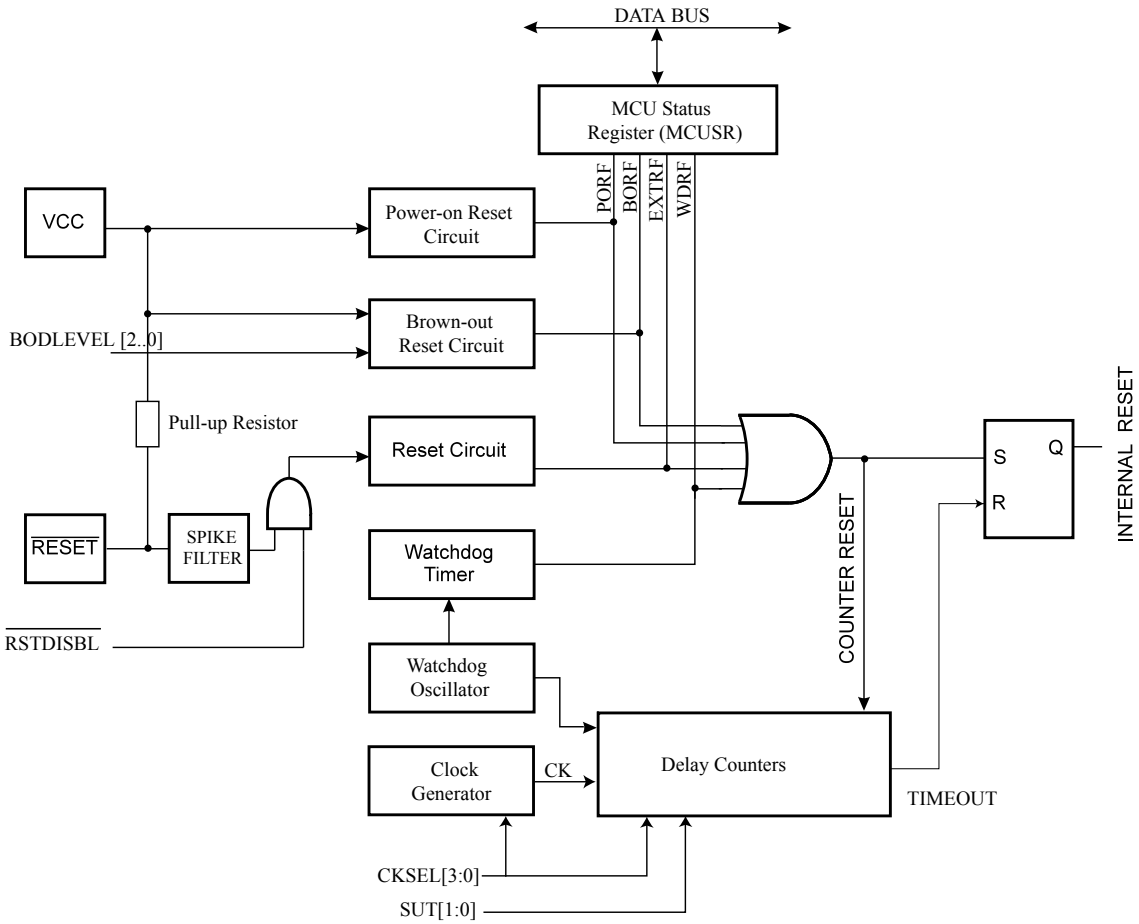
System Clock and Clock Options on page 51

## 15.2. Reset Sources

The device has the following sources of reset:

- Power-on Reset. The MCU is reset when the supply voltage is less than 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 System Reset. The MCU is reset when the Watchdog Timer period expires and the Watchdog System Reset mode is enabled.
- Brown-out Reset. The MCU is reset when the supply voltage $V_{CC}$ is less than the Brown-out Reset threshold ($V_{BOT}$) and the Brown-out Detector is enabled.

**Figure 15-1. Reset Logic**




## 15.3. Power-on Reset

A Power-on Reset (POR) pulse is generated by an On-chip detection circuit. 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 15-2. MCU Start-up, $\overline{\text{RESET}}$ Tied to $V_{CC}$**

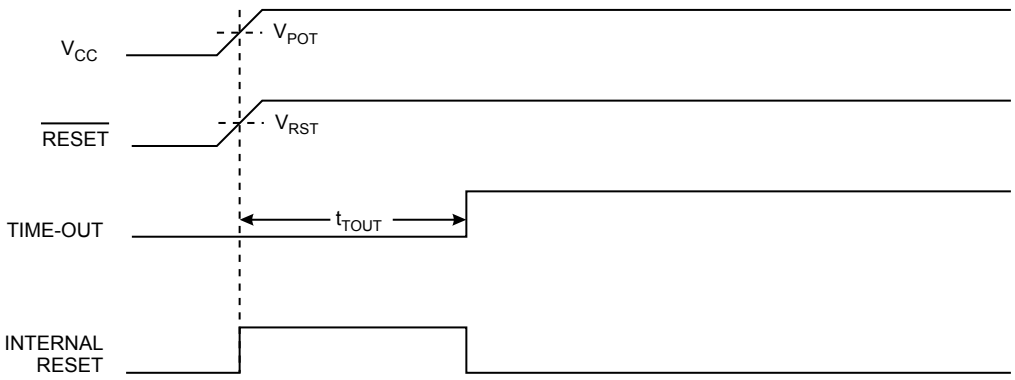
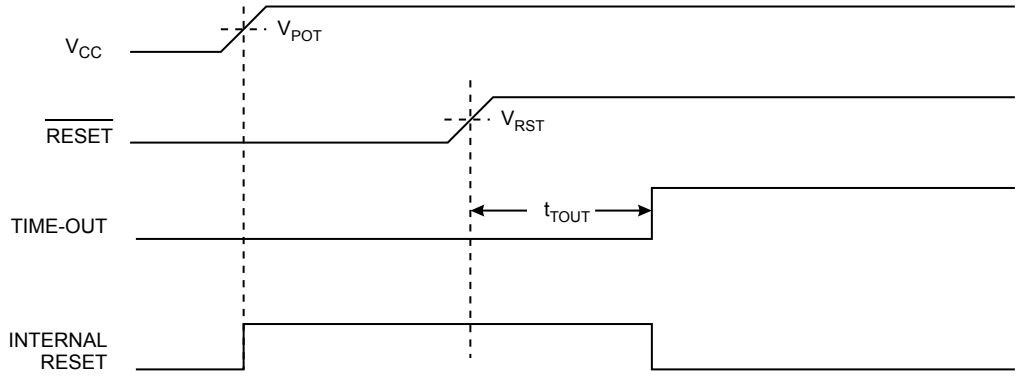


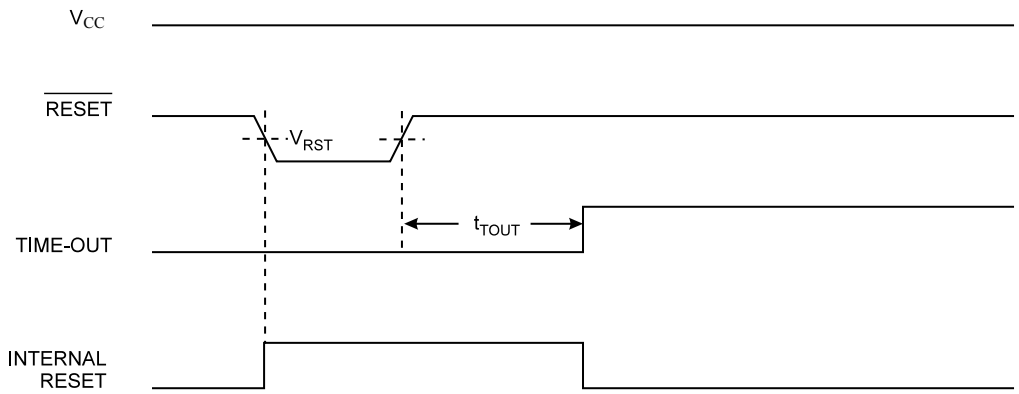
**Figure 15-3. MCU Start-up, $\overline{\text{RESET}}$ Extended Externally**




## 15.4. External Reset

An External Reset is generated by a low level on the $\overline{\text{RESET}}$ pin. Reset pulses longer than the minimum pulse width 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. The External Reset can be disabled by the RSTDISBL fuse.

**Figure 15-4. External Reset During Operation**

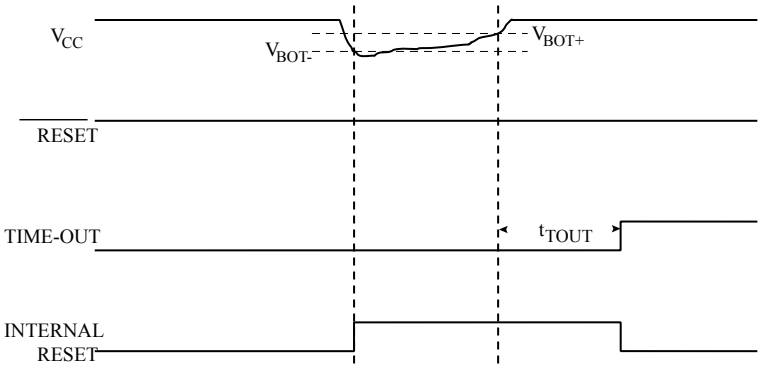



## 15.5. Brown-out Detection

The device has an On-chip Brown-out Detection (BOD) circuit for monitoring the $V_{CC}$ level during operation by comparing it to a fixed trigger level. The trigger level for the BOD can be selected by the BODLEVEL Fuses. The trigger level has a hysteresis to ensure spike free Brown-out Detection. The hysteresis on the detection level should be interpreted as $V_{BOT+} = V_{BOT} + V_{HYST}/2$ and $V_{BOT-} = V_{BOT} - V_{HYST}/2$. When the BOD is enabled, and $V_{CC}$ decreases to a value below the trigger level ($V_{BOT-}$ in the following figure), the Brown-out Reset is immediately activated. When $V_{CC}$ increases above the trigger level ($V_{BOT+}$ in the following figure), the delay counter starts the MCU after the Time-out period $t_{TOUT}$ has expired.

The BOD circuit will only detect a drop in $V_{CC}$ if the voltage stays below the trigger level for longer than $t_{BOD}$.

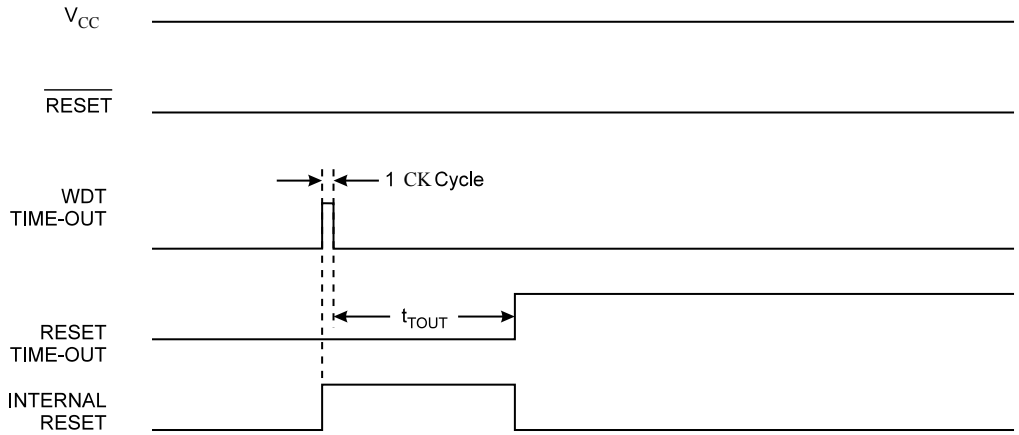
**Figure 15-5. Brown-out Reset During Operation**




## 15.6. Watchdog System Reset

When the Watchdog times out, it will generate a short reset pulse of one CK cycle duration. On the falling edge of this pulse, the delay timer starts counting the Time-out period $t_{TOUT}$.

**Figure 15-6. Watchdog System Reset During Operation**




## 15.7. Internal Voltage Reference

The device 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.

### 15.7.1. 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. To save power, the reference is not always turned on. The reference is on during the following situations:

1. When the BOD is enabled (by programming the BODLEVEL [2:0] Fuses).
2. When the bandgap reference is connected to the Analog Comparator (by setting the ACBG bit in ACSR (ACSR.ACBG)).
3. When the ADC is enabled.

Thus, when the BOD is not enabled, after setting ACSR.ACBG 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.

## 15.8. Watchdog Timer

If the watchdog timer is not needed in the application, the 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 significantly to the total current consumption.

Refer to Watchdog System Reset for details on how to configure the watchdog timer.

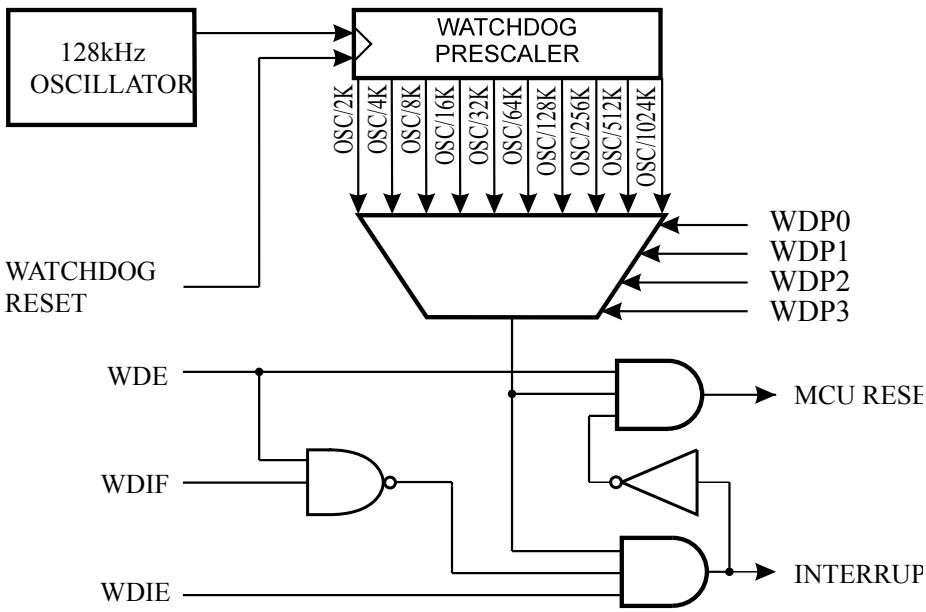
### 15.8.1. Features

- Clocked from separate On-chip Oscillator
- Three operating modes:
  - Interrupt
  - System Reset
  - Interrupt and System Reset
- Selectable Time-out period from 16ms to 8s
- Possible Hardware fuse Watchdog always on (WDTON) for fail-safe mode

### 15.8.2. Overview

The device has an Enhanced Watchdog Timer (WDT). The WDT is a timer counting cycles of a separate on-chip 128kHz oscillator. The WDT gives an interrupt or a system reset when the counter reaches a given time-out value. In normal operation mode, it is required that the system uses the Watchdog Timer Reset (WDR) instruction to restart the counter before the time-out value is reached. If the system doesn't restart the counter, an interrupt or system reset will be issued.

**Figure 15-7. Watchdog Timer**




In Interrupt mode, the WDT gives an interrupt when the timer expires. This interrupt can be used to wake the device from sleep-modes, and also as a general system timer. One example is to limit the maximum time allowed for certain operations, giving an interrupt when the operation has run longer than expected. In System Reset mode, the WDT gives a reset when the timer expires. This is typically used to prevent system hang-up in case of runaway code. The third mode, Interrupt and System Reset mode, combines the other two modes by first giving an interrupt and then switch to System Reset mode. This mode will for instance allow a safe shutdown by saving critical parameters before a system reset.

The Watchdog always on (WDTON) fuse, if programmed, will force the Watchdog Timer to System Reset mode. With the fuse programmed the System Reset mode bit (WDE) and Interrupt mode bit (WDIE) are locked to 1 and 0 respectively. To further ensure program security, alterations to the Watchdog set-up must follow timed sequences. The sequence for clearing WDE and changing time-out configuration is as follows:

1. In the same operation, write a logic one to the Watchdog change enable bit (WDCE) and Watchdog System Reset Enable (WDE) in Watchdog Timer Control Register (WDTCSR.WDCE and WDTCSR.WDE). A logic one must be written to WDTCSR.WDE regardless of the previous value of the WDTCSR.WDE.
2. Within the next four clock cycles, write the WDTCSR.WDE and Watchdog prescaler bits group (WDTCSR.WDP) as desired, but with the WDTCSR.WDCE cleared. This must be done in one operation.

The following examples show a function for turning off the Watchdog Timer. The examples assume that interrupts are controlled (e.g. by disabling interrupts globally) so that no interrupts will occur during the execution of these functions.

**Assembly Code Example**

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

**C Code Example**

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

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

The following code examples shows how to change the time-out value of the Watchdog Timer.

**Assembly Code Example**

```
WDT_Prescaler_Change:
   ; Turn off global interrupt
   cli
   ; Reset Watchdog Timer
   wdr
   ; Start timed sequence
   lds r16, WDTCSR
   ori r16, (1<<WDCE) | (1<<WDE)
   sts WDTCSR, r16
   ; -- Got four cycles to set the new values from here -
   ; Set new prescaler(time-out) value = 64K cycles (~0.5 s)
   ldi r16, (1<<WDE) | (1<<WDP2) | (1<<WDP0)
   sts WDTCSR, r16
   ; -- Finished setting new values, used 2 cycles -
   ; Turn on global interrupt
   sei
   ret
```

**C Code Example**

```
void WDT_Prescaler_Change(void)
{
    __disable_interrupt();
    __watchdog_reset();
    /* Start timed sequence */
    WDTCSR |= (1<<WDCE) | (1<<WDE);
    /* Set new prescaler(time-out) value = 64K cycles (~0.5 s) */
    WDTCSR = (1<<WDE) | (1<<WDP2) | (1<<WDP0);
    __enable_interrupt();
}
```

**Note:** The Watchdog Timer should be reset before any change of the WDTCSR.WDP bits, since a change in the WDTCSR.WDP bits can result in a time-out when switching to a shorter time-out period.

## 15.9. Register Description

### 15.9.1. MCU Status Register

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.

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** MCUSR
**Offset:** 0x54
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x34

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | WDRF | BORF | EXTRF | PORF |
| Access | | | | | R/W | R/W | R/W | R/W |
| Reset | | | | | 0 | 0 | 0 | 0 |

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

This bit is set if a Watchdog System Reset occurs. The bit is reset by a Power-on Reset, or by writing a '0' to it.

**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 '0' to it.

**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 '0' to it.

**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 '0' to it.

### 15.9.2. WDTCSR – Watchdog Timer Control Register

**Name:** WDTCSR
**Offset:** 0x60
**Reset:** 0x00
**Property:**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | WDIF | WDIE | WDP[3] | WDCE | WDE | WDP[2:0] | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bit 7 – WDIF: Watchdog Interrupt Flag**
This bit is set when a timeout 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 '1' to it. When the I-bit in SREG and WDIE are set, the Watchdog Timeout Interrupt is executed.

**Bit 6 – WDIE: Watchdog Interrupt Enable**
When this bit is written to '1' and the I-bit in the Status Register is set, the Watchdog Interrupt is enabled. If WDE is cleared in combination with this setting, the Watchdog Timer is in Interrupt Mode, and the corresponding interrupt is executed if timeout in the Watchdog Timer occurs. If WDE is set, the Watchdog Timer is in Interrupt and System Reset Mode. The first timeout in the Watchdog Timer will set WDIF. Executing the corresponding interrupt vector will clear WDIE and WDIF automatically by hardware (the Watchdog goes to System Reset Mode).

This is useful for keeping the Watchdog Timer security while using the interrupt. To stay in Interrupt and System Reset Mode, WDIE must be set after each interrupt. This should however not be done within the interrupt service routine itself, as this might compromise the safety function of the Watchdog System Reset mode. If the interrupt is not executed before the next timeout, a System Reset will be applied.

**Table 15-1. Watchdog Timer Configuration**

| WDTON(1) | WDE | WDIE | Mode | Action on Time-out |
|---|---|---|---|---|
| 1 | 0 | 0 | Stopped | None |
| 1 | 0 | 1 | Interrupt Mode | Interrupt |
| 1 | 1 | 0 | System Reset Mode | Reset |
| 1 | 1 | 1 | Interrupt and System Reset Mode | Interrupt, then go to System Reset Mode |
| 0 | x | x | System Reset Mode | Reset |

**Note:** 1. WDTON Fuse set to '0' means programmed and '1' means unprogrammed.

**Bit 5 – WDP[3]: Watchdog Timer Prescaler 3**

**Bit 4 – WDCE: Watchdog Change Enable**
This bit is used in timed sequences for changing WDE and prescaler bits. To clear the WDE bit, and/or change the prescaler bits, WDCE must be set. Once written to '1', hardware will clear WDCE after four clock cycles.

**Bit 3 – WDE: Watchdog System Reset Enable**

WDE is overridden by WDRF in MCUSR. This means that WDE is always set when WDRF is set. To clear WDE, WDRF must be cleared first. This feature ensures multiple resets during conditions causing failure, and a safe startup after the failure.

**Bits 2:0 – WDP[2:0]: Watchdog Timer Prescaler 2, 1, and 0**

The WDP[3:0] bits determine the Watchdog Timer prescaling when the Watchdog Timer is running. The different prescaling values and their corresponding timeout periods are shown in the following table.

**Table 15-2. Watchdog Timer Prescale Select**

| WDP[3] | WDP[2] | WDP[1] | WDP[0] | Number of WDT Oscillator (Cycles) | Oscillator |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 2K (2048) | 16ms |
| 0 | 0 | 0 | 1 | 4K (4096) | 32ms |
| 0 | 0 | 1 | 0 | 8K (8192) | 64ms |
| 0 | 0 | 1 | 1 | 16K (16384) | 0.125s |
| 0 | 1 | 0 | 0 | 32K (32768) | 0.25s |
| 0 | 1 | 0 | 1 | 64K (65536) | 0.5s |
| 0 | 1 | 1 | 0 | 128K (131072) | 1.0s |
| 0 | 1 | 1 | 1 | 256K (262144) | 2.0s |
| 1 | 0 | 0 | 0 | 512K (524288) | 4.0s |
| 1 | 0 | 0 | 1 | 1024K (1048576) | 8.0s |
| 1 | 0 | 1 | 0 | Reserved | |
| 1 | 0 | 1 | 1 | | |
| 1 | 1 | 0 | 0 | | |
| 1 | 1 | 0 | 1 | | |
| 1 | 1 | 1 | 0 | | |
| 1 | 1 | 1 | 1 | | |

# 16. Interrupts

This section describes the specifics of the interrupt handling of the device. For a general explanation of the AVR interrupt handling, refer to the description of *Reset and Interrupt Handling*.

The interrupt vectors in ATmega48A/88A/168A are generally the same, with the following differences:

- Each Interrupt Vector occupies two instruction words ATmega168A ; and one instruction word in ATmega48A and ATmega88A.
- ATmega48A does not have a separate Boot Loader Section. In ATmega88A and ATmega168A the Reset Vector is affected by the BOOTRST fuse, and the Interrupt Vector start address is affected by the IVSEL bit in MCUCR

## 16.1. Interrupt Vectors in ATmega48A

**Table 16-1. Reset and Interrupt Vectors in ATmega48A**

| Vector No | Program Address | Source | Interrupts definition |
|---|---|---|---|
| 1 | 0x0000 | RESET | External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset |
| 2 | 0x0001 | INT0 | External Interrupt Request 0 |
| 3 | 0x0002 | INT1 | External Interrupt Request 0 |
| 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 | WDT | Watchdog Time-out Interrupt |
| 8 | 0x0007 | TIMER2_COMPA | Timer/Counter2 Compare Match A |
| 9 | 0x0008 | TIMER2_COMPB | Timer/Coutner2 Compare Match B |
| 10 | 0x0009 | TIMER2_OVF | Timer/Counter2 Overflow |
| 11 | 0x000A | TIMER1_CAPT | Timer/Counter1 Capture Event |
| 12 | 0x000B | TIMER1_COMPA | Timer/Counter1 Compare Match A |
| 13 | 0x000C | TIMER1_COMPB | Timer/Coutner1 Compare Match B |
| 14 | 0x000D | TIMER1_OVF | Timer/Counter1 Overflow |
| 15 | 0x000E | TIMER0_COMPA | Timer/Counter0 Compare Match A |
| 16 | 0x000F | TIMER0_COMPB | Timer/Coutner0 Compare Match B |
| 17 | 0x0010 | TIMER0_OVF | Timer/Counter0 Overflow |
| 18 | 0x0011 | SPI STC | SPI Serial Transfer Complete |
| 19 | 0x0012 | USART_RX | USART Rx Complete |
| 20 | 0x0013 | USART_UDRE | USART Data Register Empty |
| 21 | 0x0014 | USART_TX | USART Tx Complete |
| 22 | 0x0015 | ADC | ADC Conversion Complete |
| 23 | 0x0016 | EE READY | EEPROM Ready |

| Vector No | Program Address | Source | Interrupts definition |
|---|---|---|---|
| 24 | 0x0017 | ANALOG COMP | Analog Comparator |
| 25 | 0x0018 | TWI | 2-wire Serial Interface ($I^2C$) |
| 26 | 0x0019 | SPM READY | Store Program Memory Ready |

The most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address   Labels    Code                        Comments
0x0000              jmp     RESET               ; Reset
0x0001              jmp     INT0                ; IRQ0
0x0002              jmp     INT1                ; IRQ1
0x0003              jmp     PCINT0              ; PCINT0
0x0004              jmp     PCINT1              ; PCINT1
0x0005              jmp     PCINT2              ; PCINT2
0x0006              jmp     WDT                 ; Watchdog Timeout
0x0007              jmp     TIM2_COMPA          ; Timer2 CompareA
0x0008              jmp     TIM2_COMPB          ; Timer2 CompareB
0x0009              jmp     TIM2_OVF            ; Timer2 Overflow
0x000A              jmp     TIM1_CAPT           ; Timer1 Capture
0x000B              jmp     TIM1_COMPA          ; Timer1 CompareA
0x000C              jmp     TIM1_COMPB          ; Timer1 CompareB
0x000C              jmp     TIM1_OVF            ; Timer1 Overflow
0x000E              jmp     TIM0_COMPA          ; Timer0 CompareA
0x000F              jmp     TIM0_COMPB          ; Timer0 CompareB
0x0010              jmp     TIM0_OVF            ; Timer0 Overflow
0x0011              jmp     SPI_STC             ; SPI Transfer Complete
0x0012              jmp     USART_RXC           ; USART RX Complete
0x0013              jmp     USART_UDRE          ; USART UDR Empty
0x0014              jmp     USART_TXC           ; USART TX Complete
0x0015              jmp     ADC                 ; ADC Conversion Complete
0x0016              jmp     EE_RDY              ; EEPROM Ready
0x0017              jmp     ANA_COMP            ; Analog Comparator
0x0018              jmp     TWI                 ; 2-wire Serial
0x0019              jmp     SPM_RDY             ; SPM Ready
;
0x001A    RESET:    ldi     r16,high(RAMEND)    ; Main program start
0x001B              out     SPH,r16             ; Set Stack Pointer to top of RAM
0x001C              ldi     r16,low(RAMEND)
0x001D              out     SPL,r16
0x001E              sei                         ; Enable interrupts
0x001F              <instr> xxx
...       ...       ...                            ...
```

## 16.2. Interrupt Vectors in ATmega88A

**Table 16-2. Reset and Interrupt Vectors in ATmega88A**

| Vector No | Program Address(2) | Source | Interrupts definition |
|---|---|---|---|
| 1 | 0x0000(1) | RESET | External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset |
| 2 | 0x0001 | INT0 | External Interrupt Request 0 |
| 3 | 0x0002 | INT1 | External Interrupt Request 0 |
| 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 | WDT | Watchdog Time-out Interrupt |
| 8 | 0x0007 | TIMER2_COMPA | Timer/Counter2 Compare Match A |
| 9 | 0x0008 | TIMER2_COMPB | Timer/Coutner2 Compare Match B |

| Vector No | Program Address(2) | Source | Interrupts definition |
|---|---|---|---|
| 10 | 0x0009 | TIMER2_OVF | Timer/Counter2 Overflow |
| 11 | 0x000A | TIMER1_CAPT | Timer/Counter1 Capture Event |
| 12 | 0x000B | TIMER1_COMPA | Timer/Counter1 Compare Match A |
| 13 | 0x000C | TIMER1_COMPB | Timer/Coutner1 Compare Match B |
| 14 | 0x000D | TIMER1_OVF | Timer/Counter1 Overflow |
| 15 | 0x000E | TIMER0_COMPA | Timer/Counter0 Compare Match A |
| 16 | 0x000F | TIMER0_COMPB | Timer/Coutner0 Compare Match B |
| 17 | 0x0010 | TIMER0_OVF | Timer/Counter0 Overflow |
| 18 | 0x0011 | SPI STC | SPI Serial Transfer Complete |
| 19 | 0x0012 | USART_RX | USART Rx Complete |
| 20 | 0x0013 | USART_UDRE | USART Data Register Empty |
| 21 | 0x0014 | USART_TX | USART Tx Complete |
| 22 | 0x0015 | ADC | ADC Conversion Complete |
| 23 | 0x0016 | EE READY | EEPROM Ready |
| 24 | 0x0017 | ANALOG COMP | Analog Comparator |
| 25 | 0x0018 | TWI | 2-wire Serial Interface ($I^2C$) |
| 26 | 0x0019 | SPM READY | Store Program Memory Ready |

**Note:**

1. When the BOOTRST Fuse is programmed, the device will jump to the Boot Loader address at reset, see ”Boot Loader Support – Read-While-Write Self- Programming”
2. When the IVSEL bit in MCUCR is set, Interrupt Vectors will be moved to the start of the Boot Flash Section. The address of each Interrupt Vector will then be the address in this table added to the start address of the Boot Flash Section.

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

**Table 16-3. Reset and Interrupt Vectors Placement**

| BOOTRST(1) | IVSEL | Reset Address | Interrupt Vectors Start Address |
|---|---|---|---|
| 1 | 0 | 0x000 | 0x001 |
| 1 | 1 | 0x000 | Boot Reset Address + 0x0001 |
| 0 | 0 | Boot Reset Address | 0x001 |
| 0 | 1 | Boot Reset Address | Boot Reset Address + 0x0001 |

**Note:** 1. For the BOOTRST Fuse “1” means unprogrammed while “0” means programmed.

The most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address   Labels    Code                        Comments
0x0000               jmp      RESET               ; Reset
```

```
0x0001              jmp     INT0                ; IRQ0
0x0002              jmp     INT1                ; IRQ1
0x0003              jmp     PCINT0              ; PCINT0
0x0004              jmp     PCINT1              ; PCINT1
0x0005              jmp     PCINT2              ; PCINT2
0x0006              jmp     WDT                 ; Watchdog Timeout
0x0007              jmp     TIM2_COMPA          ; Timer2 CompareA
0x0008              jmp     TIM2_COMPB          ; Timer2 CompareB
0x0009              jmp     TIM2_OVF            ; Timer2 Overflow
0x000A              jmp     TIM1_CAPT           ; Timer1 Capture
0x000B              jmp     TIM1_COMPA          ; Timer1 CompareA
0x000C              jmp     TIM1_COMPB          ; Timer1 CompareB
0x000C              jmp     TIM1_OVF            ; Timer1 Overflow
0x000E              jmp     TIM0_COMPA          ; Timer0 CompareA
0x000F              jmp     TIM0_COMPB          ; Timer0 CompareB
0x0010              jmp     TIM0_OVF            ; Timer0 Overflow
0x0011              jmp     SPI_STC             ; SPI Transfer Complete
0x0012              jmp     USART_RXC           ; USART RX Complete
0x0013              jmp     USART_UDRE          ; USART UDR Empty
0x0014              jmp     USART_TXC           ; USART TX Complete
0x0015              jmp     ADC                 ; ADC Conversion Complete
0x0016              jmp     EE_RDY              ; EEPROM Ready
0x0017              jmp     ANA_COMP            ; Analog Comparator
0x0018              jmp     TWI                 ; 2-wire Serial
0x0019              jmp     SPM_RDY             ; SPM Ready
;
0x001A    RESET:    ldi     r16,high(RAMEND)    ; Main program start
0x001B              out     SPH,r16             ; Set Stack Pointer to top of RAM
0x001C              ldi     r16,low(RAMEND)
0x001D              out     SPL,r16
0x001E              sei                         ; Enable interrupts
0x001F              <instr>  xxx
...       ...       ...                            ...
```

When the BOOTRST Fuse is unprogrammed, the Boot section size set to 2Kbytes and the MCUCR.IVSEL is set before any interrupts are enabled, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address   Labels     Code                        Comments
0x000   RESET:     ldi     r16,high(RAMEND)   ; Main program start
0x001              out     SPH,r16            ; Set Stack Pointer to top of RAM
0x002              ldi     r16,low(RAMEND)
0x0003              out      SPL,r16
0x0004              sei                         ; Enable interrupts
0x0005              <instr> xxx
;
.org 0xC01
0x0C01              jmp      EXT_INT0           ; IRQ0 Handler
0x0C02              jmp      EXT_INT1           ; IRQ1 Handler
...                  ...      ...                ;
0x0C19              jmp      SPM_RDY            ; SPM Ready Handler
```

When the BOOTRST Fuse is programmed and the Boot section size set to 2Kbytes, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address   Labels     Code                          Comments
.org 0x0002
0x0002              jmp      EXT_INT0           ; IRQ0 Handler
0x0004              jmp      EXT_INT1           ; IRQ1 Handler
...                  ...      ...                ;
0x0032              jmp      SPM_RDY            ; SPM Ready Handler
;
.org  0xC00
0x0C00    RESET:    ldi      r16,high(RAMEND)    ; Main program start
0x0C01              out      SPH,r16             ; Set Stack Pointer to top of RAM
0x0C02              ldi      r16,low(RAMEND)
0x0C03              out      SPL,r16
0x0C04              sei                          ; Enable interrupts
0x0C05              <instr>  xxx
```

When the BOOTRST Fuse is programmed, the Boot section size set to 2K bytes and the MCUCR.IVSEL is set before any interrupts are enabled, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address    Labels    Code                          Comments
;
.org 0x0C00
0x0C00               jmp      RESET               ; Reset handler
0x0C01               jmp      EXT_INT0            ; IRQ0 Handler
0x0C02               jmp      EXT_INT1            ; IRQ1 Handler
...                   ...      ...                 ;
0x0C19               jmp      SPM_RDY             ; SPM Ready Handler
;
0x0C1A    RESET:     ldi      r16,high(RAMEND)    ; Main program start
0x0C1B               out      SPH,r16             ; Set Stack Pointer to top of RAM
0x0C1C               ldi      r16,low(RAMEND)
0x0C1D               out      SPL,r16
0x0C1E               sei                          ; Enable interrupts
0x0C1F               <instr>  xxx
```

## 16.3. Interrupt Vectors in ATmega168A

**Table 16-4. Reset and Interrupt Vectors in ATmega168A**

| Vector No | Program Address(2) | Source | Interrupts definition |
|---|---|---|---|
| 1 | 0x0000(1) | RESET | External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset |
| 2 | 0x0002 | INT0 | External Interrupt Request 0 |
| 3 | 0x0004 | INT1 | External Interrupt Request 0 |
| 4 | 0x0006 | PCINT0 | Pin Change Interrupt Request 0 |
| 5 | 0x0008 | PCINT1 | Pin Change Interrupt Request 1 |
| 6 | 0x000A | PCINT2 | Pin Change Interrupt Request 2 |
| 7 | 0x000C | WDT | Watchdog Time-out Interrupt |
| 8 | 0x000E | TIMER2_COMPA | Timer/Counter2 Compare Match A |
| 9 | 0x0010 | TIMER2_COMPB | Timer/Coutner2 Compare Match B |
| 10 | 0x0012 | TIMER2_OVF | Timer/Counter2 Overflow |
| 11 | 0x0014 | TIMER1_CAPT | Timer/Counter1 Capture Event |
| 12 | 0x0016 | TIMER1_COMPA | Timer/Counter1 Compare Match A |
| 13 | 0x0018 | TIMER1_COMPB | Timer/Coutner1 Compare Match B |
| 14 | 0x001A | TIMER1_OVF | Timer/Counter1 Overflow |
| 15 | 0x001C | TIMER0_COMPA | Timer/Counter0 Compare Match A |
| 16 | 0x001E | TIMER0_COMPB | Timer/Coutner0 Compare Match B |
| 17 | 0x0020 | TIMER0_OVF | Timer/Counter0 Overflow |
| 18 | 0x0022 | SPI STC | SPI Serial Transfer Complete |
| 19 | 0x0024 | USART_RX | USART Rx Complete |
| 20 | 0x0026 | USART_UDRE | USART Data Register Empty |
| 21 | 0x0028 | USART_TX | USART Tx Complete |

| Vector No | Program Address(2) | Source | Interrupts definition |
|---|---|---|---|
| 22 | 0x002A | ADC | ADC Conversion Complete |
| 23 | 0x002C | EE READY | EEPROM Ready |
| 24 | 0x002E | ANALOG COMP | Analog Comparator |
| 25 | 0x0030 | TWI | 2-wire Serial Interface ($I^2C$) |
| 26 | 0x0032 | SPM READY | Store Program Memory Ready |

**Note:**

1. When the BOOTRST Fuse is programmed, the device will jump to the Boot Loader address at reset, see ”Boot Loader Support – Read-While-Write Self- Programming”
2. When the IVSEL bit in MCUCR is set, Interrupt Vectors will be moved to the start of the Boot Flash Section. The address of each Interrupt Vector will then be the address in this table added to the start address of the Boot Flash Section.

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

**Table 16-5. Reset and Interrupt Vectors Placement**

| BOOTRST(1) | IVSEL | Reset Address | Interrupt Vectors Start Address |
|---|---|---|---|
| 1 | 0 | 0x000 | 0x002 |
| 1 | 1 | 0x000 | Boot Reset Address + 0x0002 |
| 0 | 0 | Boot Reset Address | 0x002 |
| 0 | 1 | Boot Reset Address | Boot Reset Address + 0x0002 |

**Note:** 1. For the BOOTRST Fuse “1” means unprogrammed while “0” means programmed.

The most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address   Labels    Code                        Comments
0x0000              jmp     RESET               ; Reset
0x0002              jmp     INT0                ; IRQ0
0x0004              jmp     INT1                ; IRQ1
0x0006              jmp     PCINT0              ; PCINT0
0x0008              jmp     PCINT1              ; PCINT1
0x000A              jmp     PCINT2              ; PCINT2
0x000C              jmp     WDT                 ; Watchdog Timeout
0x000E              jmp     TIM2_COMPA          ; Timer2 CompareA
0x0010              jmp     TIM2_COMPB          ; Timer2 CompareB
0x0012              jmp     TIM2_OVF            ; Timer2 Overflow
0x0014              jmp     TIM1_CAPT           ; Timer1 Capture
0x0016              jmp     TIM1_COMPA          ; Timer1 CompareA
0x0018              jmp     TIM1_COMPB          ; Timer1 CompareB
0x001A              jmp     TIM1_OVF            ; Timer1 Overflow
0x001C              jmp     TIM0_COMPA          ; Timer0 CompareA
0x001E              jmp     TIM0_COMPB          ; Timer0 CompareB
0x0020              jmp     TIM0_OVF            ; Timer0 Overflow
0x0022              jmp     SPI_STC             ; SPI Transfer Complete
0x0024              jmp     USART_RXC           ; USART RX Complete
0x0026              jmp     USART_UDRE          ; USART UDR Empty
0x0028              jmp     USART_TXC           ; USART TX Complete
0x002A              jmp     ADC                 ; ADC Conversion Complete
0x002C              jmp     EE_RDY              ; EEPROM Ready
0x002E              jmp     ANA_COMP            ; Analog Comparator
0x0030              jmp     TWI                 ; 2-wire Serial
0x0032              jmp     SPM_RDY             ; SPM Ready
;
```

```
0x0034    RESET:    ldi     r16,high(RAMEND)    ; Main program start
0x0035              out     SPH,r16             ; Set Stack Pointer to top of RAM
0x0036              ldi     r16,low(RAMEND)
0x0037              out     SPL,r16
0x0038              sei                         ; Enable interrupts
0x0039              <instr>  xxx
...       ...       ...                           ...
```

When the BOOTRST Fuse is unprogrammed, the Boot section size set to 2Kbytes and the MCUCR.IVSEL is set before any interrupts are enabled, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address   Labels     Code                        Comments
0x0000   RESET:     ldi     r16,high(RAMEND)   ; Main program start
0x0001              out     SPH,r16            ; Set Stack Pointer to top of RAM
0x0002              ldi     r16,low(RAMEND)
0x0003              out     SPL,r16
0x0004              sei                        ; Enable interrupts
0x0005              <instr> xxx
;
.org 0x1C02
0x1C02              jmp     EXT_INT0           ; IRQ0 Handler
0x1C04              jmp     EXT_INT1           ; IRQ1 Handler
...                  ...     ...                ;
0x1C32              jmp     SPM_RDY            ; SPM Ready Handler
```

When the BOOTRST Fuse is programmed and the Boot section size set to 2Kbytes, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address   Labels     Code                        Comments
.org 0x0002
0x0002              jmp     EXT_INT0           ; IRQ0 Handler
0x0004              jmp     EXT_INT1           ; IRQ1 Handler
...                  ...     ...                ;
0x0032              jmp     SPM_RDY            ; SPM Ready Handler
;
.org  0x1C00
0x1C00    RESET:    ldi     r16,high(RAMEND)    ; Main program start
0x1C01              out     SPH,r16             ; Set Stack Pointer to top of RAM
0x1C02              ldi     r16,low(RAMEND)
0x1C03              out     SPL,r16
0x1C04              sei                         ; Enable interrupts
0x1C05              <instr>  xxx
```

When the BOOTRST Fuse is programmed, the Boot section size set to 2K bytes and the MCUCR.IVSEL is set before any interrupts are enabled, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address    Labels    Code                         Comments
;
.org 0x1C00
0x1C00              jmp     RESET               ; Reset handler
0x1C02              jmp     EXT_INT0            ; IRQ0 Handler
0x1C04              jmp     EXT_INT1            ; IRQ1 Handler
...                  ...     ...                 ;
0x1C32              jmp     SPM_RDY             ; SPM Ready Handler
;
0x1C34    RESET:    ldi     r16,high(RAMEND)    ; Main program start
0x1C35              out     SPH,r16             ; Set Stack Pointer to top of RAM
0x1C36              ldi     r16,low(RAMEND)
0x1C37              out     SPL,r16
0x1C38              sei                         ; Enable interrupts
0x1C39              <instr>  xxx
```

## 16.4. Register Description

### 16.4.1. Moving Interrupts Between Application and Boot Space

The MCU Control Register controls the placement of the Interrupt Vector table. (For ATmega88A and ATmega168A )

### 16.4.2. MCU Control Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** MCUCR
**Offset:** 0x55
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x35

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | PUD | | | IVSEL | IVCE |
| Access | | | | R/W | | | R/W | R/W |
| Reset | | | | 0 | | | 0 | 0 |

**Bit 4 – PUD: Pull-up Disable**

When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01).

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

When the IVSEL bit is cleared (zero), the Interrupt Vectors are placed at the start of the Flash memory. When this bit is set (one), the Interrupt Vectors are moved to the beginning of the Boot Loader section of the Flash. The actual address of the start of the Boot Flash Section is determined by the BOOTSZ Fuses. To avoid unintentional changes of Interrupt Vector tables, a special write procedure must be followed to change the IVSEL bit:

1. Write the Interrupt Vector Change Enable (IVCE) bit to one.
2. Within four cycles, write the desired value to IVSEL while writing a zero to IVCE.

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

**Note:** If Interrupt Vectors are placed in the Boot Loader section and Boot Lock bit BLB02 is programmed, interrupts are disabled while executing from the Application section. If Interrupt Vectors are placed in the Application section and Boot Lock bit BLB12 is programed, interrupts are disabled while executing from the Boot Loader section.

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

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

Assembly Code Example

```
Move_interrupts:
; Get MCUCR
in    r16, MCUCR
mov   r17, r16
; Enable change of Interrupt Vectors
ori   r16, (1<<IVCE)
out   MCUCR, r16
; Move interrupts to Boot Flash section
ori   r17, (1<<IVSEL)
out   MCUCR, r17
ret
```

C Code Example

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

# 17. EXINT - External Interrupts

The External Interrupts are triggered by the INT pins or any of the PCINT pins. Observe that, if enabled, the interrupts will trigger even if the INT or PCINT pins are configured as outputs. This feature provides a way of generating a software interrupt.

The Pin Change Interrupt Request 2 (PCI2) will trigger if any enabled PCINT[23:16] pin toggles. The Pin Change Interrupt Request 1 (PCI1) will trigger if any enabled PCINT[14:8] pin toggles. The Pin Change Interrupt Request 0 (PCI0) will trigger if any enabled PCINT[7:0] pin toggles. The PCMSK2, PCMSK1 and PCMSK0 Registers control which pins contribute to the pin change interrupts. Pin change interrupts on PCINT are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than Idle mode.

The External Interrupts can be triggered by a falling or rising edge or a low level. This is set up as indicated in the specification for the External Interrupt Control Register A (EICRA). When the External Interrupts are enabled and are configured as level triggered, the interrupts will trigger as long as the pin is held low. Note that recognition of falling or rising edge interrupts on INT requires the presence of an I/O clock. Low level interrupt on INT is detected asynchronously. This implies that this interrupt can be used for waking the part also from sleep modes other than Idle mode. The I/O clock is halted in all sleep modes except Idle mode.

**Note:** 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 by the SUT and CKSEL Fuses.

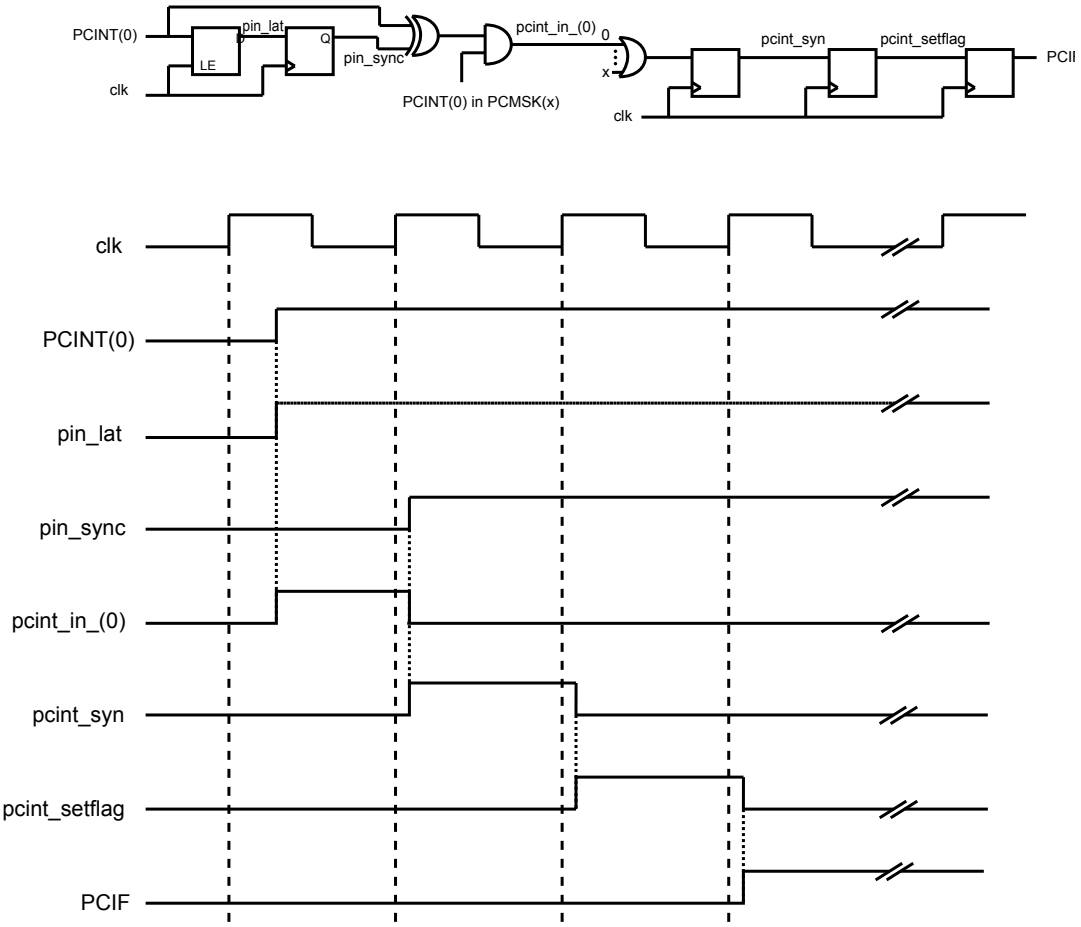
**Related Links**

System Control and Reset on page 74
Clock Systems and Their Distribution on page 51
System Clock and Clock Options on page 51

## 17.1. Pin Change Interrupt Timing

An example of timing of a pin change interrupt is shown in the following figure.

**Figure 17-1. Timing of pin change interrupts**




**Related Links**

System Control and Reset on page 74

Clock Systems and Their Distribution on page 51

System Clock and Clock Options on page 51

## 17.2. Register Description

### 17.2.1. External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

**Name:** EICRA
**Offset:** 0x69
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | ISC11 | ISC10 | ISC01 | ISC00 |
| Access | | | | | R/W | R/W | R/W | R/W |
| Reset | | | | | 0 | 0 | 0 | 0 |

**Bits 3:2 – ISC1n: Interrupt Sense Control 1 [n = 1:0]**

The External Interrupt 1 is activated by the external pin INT1 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT1 pin that activate the interrupt are defined in the table below. The value on the 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. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

| Value | Description |
|---|---|
| 00 | The low level of INT1 generates an interrupt request. |
| 01 | Any logical change on INT1 generates an interrupt request. |
| 10 | The falling edge of INT1 generates an interrupt request. |
| 11 | The rising edge of INT1 generates an interrupt request. |

**Bits 1:0 – ISC0n: Interrupt Sense Control 0 [n = 1:0]**

The External Interrupt 0 is activated by the external pin INT0 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT0 pin that activate the interrupt are defined in table below. The value on the INT0 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. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

| Value | Description |
|---|---|
| 00 | The low level of INT0 generates an interrupt request. |
| 01 | Any logical change on INT0 generates an interrupt request. |
| 10 | The falling edge of INT0 generates an interrupt request. |
| 11 | The rising edge of INT0 generates an interrupt request. |

### 17.2.2. External Interrupt Mask Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** EIMSK
**Offset:** 0x3D
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x1D

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | INT1 | INT0 |
| Access | | | | | | | R/W | R/W |
| Reset | | | | | | | 0 | 0 |

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

When the INT1 bit is set and the I-bit in the Status Register (SREG) is set, the external pin interrupt is enabled. The Interrupt Sense Control1 bits 1/0 (ISC11 and ISC10) in the External Interrupt Control Register A (EICRA) define whether the external interrupt is activated on rising and/or falling edge of the INT1 pin or level sensed. Activity on the pin will cause an interrupt request even if INT1 is configured as an output. The corresponding interrupt of External Interrupt Request 1 is executed from the INT1 Interrupt Vector.

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

When the INT0 bit is set and the I-bit in the Status Register (SREG) is set, the external pin interrupt is enabled. The Interrupt Sense Control0 bits 1/0 (ISC01 and ISC00) in the External Interrupt Control Register A (EICRA) define whether the external interrupt is activated on rising and/or falling edge of the INT0 pin or level sensed. Activity on the pin will cause an interrupt request even if INT0 is configured as an output. The corresponding interrupt of External Interrupt Request 0 is executed from the INT0 Interrupt Vector.

### 17.2.3. External Interrupt Flag Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** EIFR
**Offset:** 0x3C
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x1C

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | INTF1 | INTF0 |
| Access | | | | | | | R/W | R/W |
| Reset | | | | | | | 0 | 0 |

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

When an edge or logic change on the INT1 pin triggers an interrupt request, INTF1 will be set. If 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 routine is executed. Alternatively, the flag can be cleared by writing '1' to it. This flag is always cleared when INT1 is configured as a level interrupt.

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

When an edge or logic change on the INT0 pin triggers an interrupt request, INTF0 will be set. If 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 routine is executed. Alternatively, the flag can be cleared by writing '1' to it. This flag is always cleared when INT0 is configured as a level interrupt.

### 17.2.4. Pin Change Interrupt Control Register

**Name:** PCICR
**Offset:** 0x68
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | PCIE2 | PCIE1 | PCIE0 |
| Access | | | | | | R/W | R/W | R/W |
| Reset | | | | | | 0 | 0 | 0 |

**Bit 2 – PCIE2: Pin Change Interrupt Enable 2**
When the PCIE2 bit is set and the I-bit in the Status Register (SREG) is set, pin change interrupt 2 is enabled. Any change on any enabled PCINT[23:16] pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PCI2 Interrupt Vector. PCINT[23:16] pins are enabled individually by the PCMSK2 Register.

**Bit 1 – PCIE1: Pin Change Interrupt Enable 1**
When the PCIE1 bit is set and the I-bit in the Status Register (SREG) is set, pin change interrupt 1 is enabled. Any change on any enabled PCINT[14:8] pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PCI1 Interrupt Vector. PCINT[14:8] pins are enabled individually by the PCMSK1 Register.

**Bit 0 – PCIE0: Pin Change Interrupt Enable 0**
When the PCIE0 bit is set and the I-bit in the Status Register (SREG) is set, pin change interrupt 0 is enabled. Any change on any enabled PCINT[7:0] pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PCI0 Interrupt Vector. PCINT[7:0] pins are enabled individually by the PCMSK0 Register.

### 17.2.5. Pin Change Interrupt Flag Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** PCIFR
**Offset:** 0x3B
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x1B

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | PCIF2 | PCIF1 | PCIF0 |
| Access | | | | | | R/W | R/W | R/W |
| Reset | | | | | | 0 | 0 | 0 |

**Bit 2 – PCIF2: Pin Change Interrupt Flag 2**
When a logic change on any PCINT[23:16] pin triggers an interrupt request, PCIF2 will be set. If 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 '1' to it.

**Bit 1 – PCIF1: Pin Change Interrupt Flag 1**
When a logic change on any PCINT[14:8] pin triggers an interrupt request, PCIF1 will be set. If 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 '1' to it.

**Bit 0 – PCIF0: Pin Change Interrupt Flag 0**
When a logic change on any PCINT[7:0] pin triggers an interrupt request, PCIF0 will be set. If 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 '1' to it.

### 17.2.6. Pin Change Mask Register 2

**Name:** PCMSK2
**Offset:** 0x6D
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | PCINT23 | PCINT22 | PCINT21 | PCINT20 | PCINT19 | PCINT18 | PCINT17 | PCINT16 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 0, 1, 2, 3, 4, 5, 6, 7 – PCINT16, PCINT17, PCINT18, PCINT19, PCINT20, PCINT21, PCINT22, PCINT23: Pin Change Enable Mask**
Each PCINT[23:16]-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT[23:16] is set and the PCIE2 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT[23:16] is cleared, pin change interrupt on the corresponding I/O pin is disabled.

### 17.2.7. Pin Change Mask Register 1

**Name:** PCMSK1
**Offset:** 0x6C
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | PCINT14 | PCINT13 | PCINT12 | PCINT11 | PCINT10 | PCINT9 | PCINT8 |
| Access | | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 0, 1, 2, 3, 4, 5, 6 – PCINT8, PCINT9, PCINT10, PCINT11, PCINT12, PCINT13, PCINT14: Pin Change Enable Mask**
Each PCINT[15:8]-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT[15:8] is set and the PCIE1 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT[15:8] is cleared, pin change interrupt on the corresponding I/O pin is disabled.

### 17.2.8. Pin Change Mask Register 0

**Name:** PCMSK0
**Offset:** 0x6B
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | PCINT7 | PCINT6 | PCINT5 | PCINT4 | PCINT3 | PCINT2 | PCINT1 | PCINT0 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

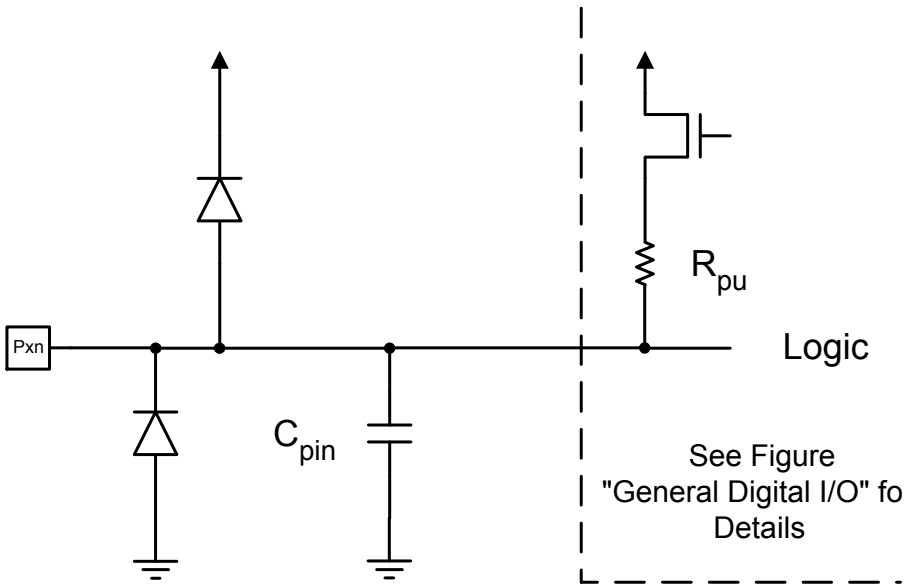
**Bits 7:0 – PCINTn: Pin Change Enable Mask [n = 7:0]**
Each PCINT[7:0] bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT[7:0] is set and the PCIE0 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT[7:0] is cleared, pin change interrupt on the corresponding I/O pin is disabled.

# 18. I/O-Ports

## 18.1. 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 (if configured as input). Each output buffer has symmetrical drive characteristics with both high sink and source capability. 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 the following figure.

**Figure 18-1. I/O Pin Equivalent Schematic**




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

I/O memory address locations are allocated for each port, one each for the Data Register – PORTx, Data Direction Register – DDRx, and the Port Input Pins – PINx. The Port Input Pins I/O location is read only, while the Data Register and the Data Direction Register are read/write. However, writing '1' to a bit in the PINx Register will result in a toggle in the corresponding bit in the Data Register. In addition, the Pull-up Disable – PUD bit in MCUCR disables the pull-up function for all pins in all ports when set.

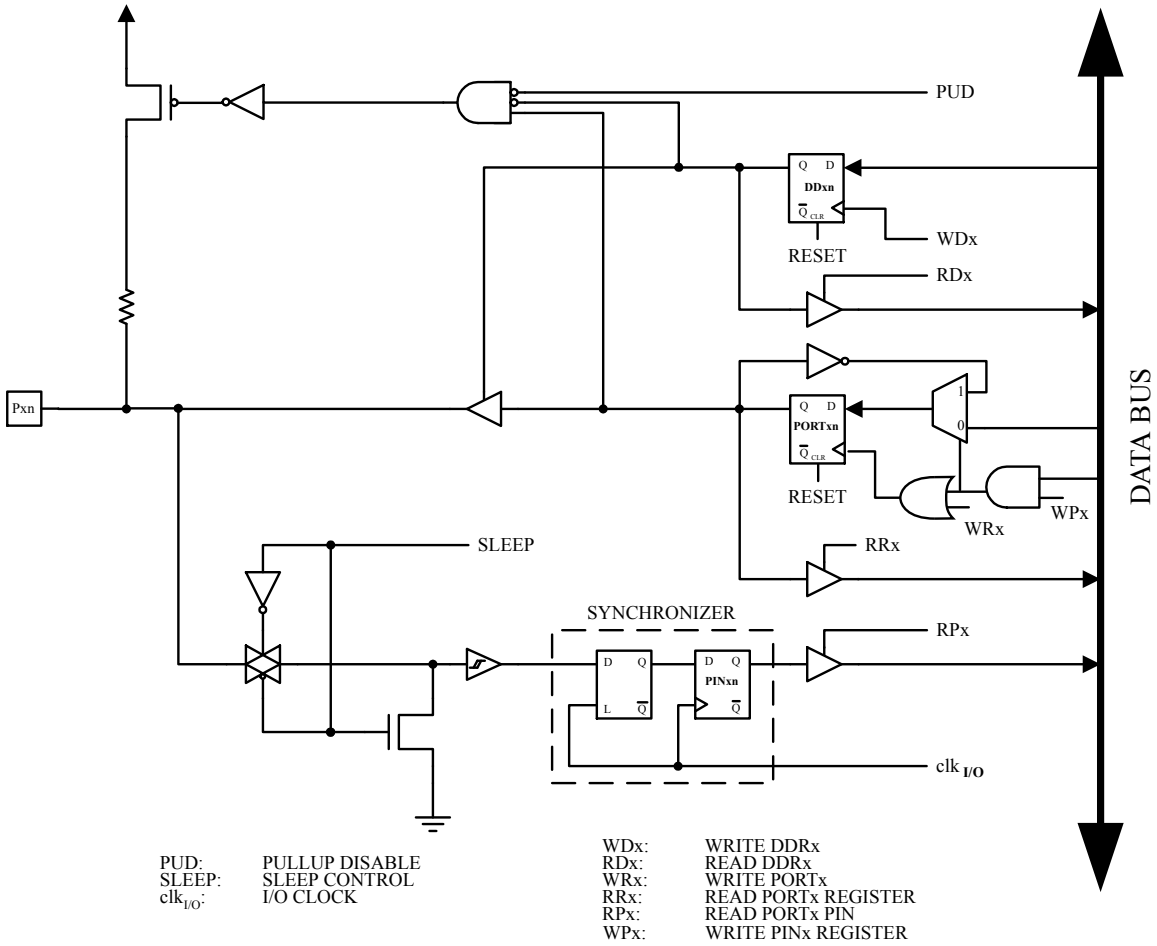
Using the I/O port as General Digital I/O is described in next section. Most port pins are multiplexed with alternate functions for the peripheral features on the device. How each alternate function interferes with the port pin is described in *Alternate Port Functions* section in this chapter. Refer to the individual module sections for a full description of the alternate functions.

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

## 18.2. Ports as General Digital I/O

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

**Figure 18-2. General Digital I/O(1)**




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

### 18.2.1. Configuring the Pin

Each port pin consists of three register bits: DDxn, PORTxn, and PINxn. As shown in the Register Description, the DDxn bits are accessed at the DDRx I/O address, the PORTxn bits at the PORTx 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 to '1', Pxn is configured as an output pin. If DDxn is written to '0', Pxn is configured as an input pin.

If PORTxn is written to '1' when the pin is configured as an input pin, the pull-up resistor is activated. To switch the pull-up resistor off, PORTxn has to be written to '0' or the pin has to be configured as an output pin. The port pins are tri-stated when reset condition becomes active, even if no clocks are running.

If PORTxn is written to '1' when the pin is configured as an output pin, the port pin is driven high. If PORTxn is written logic zero when the pin is configured as an output pin, the port pin is driven low.

### 18.2.2. Toggling the Pin

Writing a '1' to PINxn toggles the value of PORTxn, independent on the value of DDRxn. The SBI instruction can be used to toggle one single bit in a port.

### 18.2.3. Switching Between Input and Output

When switching between tri-state ({DDxn, PORTxn} = 0b00) and output high ({DDxn, PORTxn} = 0b11), an intermediate state with either pull-up enabled {DDxn, PORTxn} = 0b01) or output low ({DDxn, PORTxn} = 0b10) must occur. Normally, the pull-up enabled state is fully acceptable, as a high-impedance environment will not notice the difference between a strong high driver and a pull-up. If this is not the case, the PUD bit in the MCUCR Register can be set to disable all pull-ups in all ports.

Switching between input with pull-up and output low generates the same problem. The user must use either the tri-state ({DDxn, PORTxn} = 0b00) or the output high state ({DDxn, PORTxn} = 0b11) as an intermediate step.

The following table summarizes the control signals for the pin value.

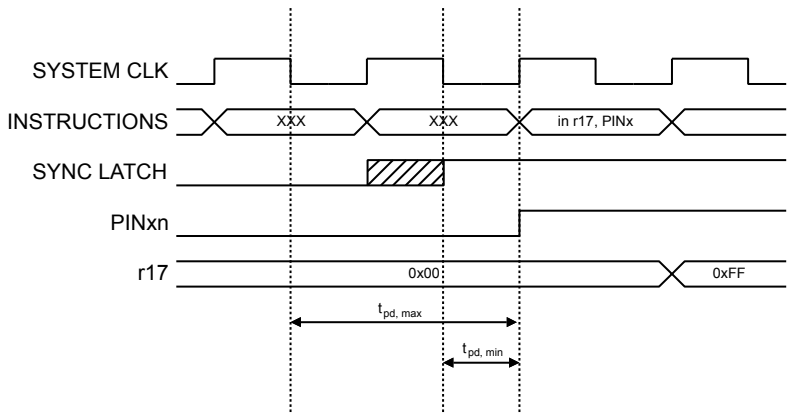
**Table 18-1. Port Pin Configurations**

| DDxn | PORTxn | PUD (in MCUCR) | I/O | Pull-up | Comment |
|---|---|---|---|---|---|
| 0 | 0 | X | Input | No | Tri-state (Hi-Z) |
| 0 | 1 | 0 | Input | Yes | Pxn will source current if ext. pulled low |
| 0 | 1 | 1 | Input | No | Tri-state (Hi-Z) |
| 1 | 0 | X | Output | No | Output Low (Sink) |
| 1 | 1 | X | Output | No | Output High (Source) |

### 18.2.4. 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 Ports as General Digital I/O, 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. The following figure 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 18-3. Synchronization when Reading an Externally Applied Pin value**

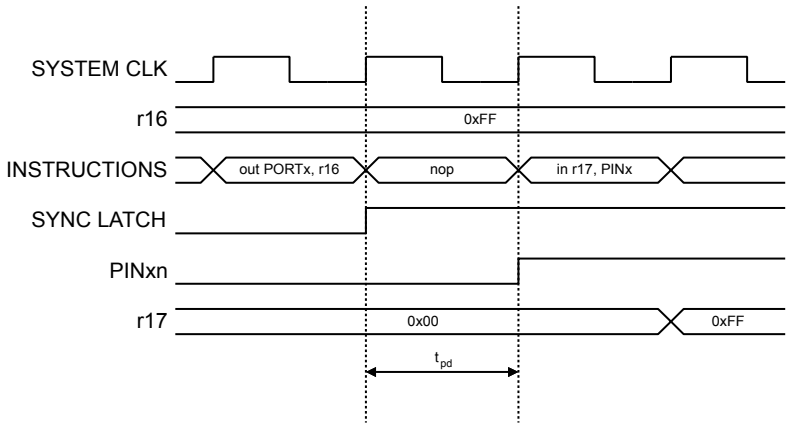



Consider the clock period starting shortly after the first falling edge of the system clock. The latch is closed when the clock is low, and goes transparent when the clock is high, as indicated by the shaded

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 the following figure. 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 1 system clock period.

**Figure 18-4. Synchronization when Reading a Software Assigned Pin Value**




The following code example shows how to set port B pins 0 and 1 high, 2 and 3 low, and define the port pins from 4 to 7 as input with pull-ups assigned to port pins 6 and 7. 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(1)**

```
...
 ; Define pull-ups and set outputs high
 ; Define directions for port pins
 ldi r16,(1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0)
 ldi r17,(1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0)
 out PORTB,r16
 out DDRB,r17
 ; Insert nop for synchronization
 nop
 ; Read port pins
 in r16,PINB
 ...
```

**Note:** 1. For the assembly program, two temporary registers are used to minimize the time from pull-ups are set on pins 0, 1, 6, and 7, until the direction bits are correctly set, defining bit 2 and 3 as low and redefining bits 0 and 1 as strong high drivers.

**C Code Example**

```
unsigned char i;
 ...
 /* Define pull-ups and set outputs high */
 /* Define directions for port pins */
 PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0);
 DDRB = (1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0);
 /* Insert nop for synchronization*/
 __no_operation();
 /* Read port pins */
```

```
i = PINB;
...
```

### 18.2.5. Digital Input Enable and Sleep Modes

As shown in the figure of General Digital I/O, 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 mode and Standby mode 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 alternate functions as described in *Alternate Port Functions* section in this chapter.

If a logic high level 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.

### 18.2.6. Unconnected Pins

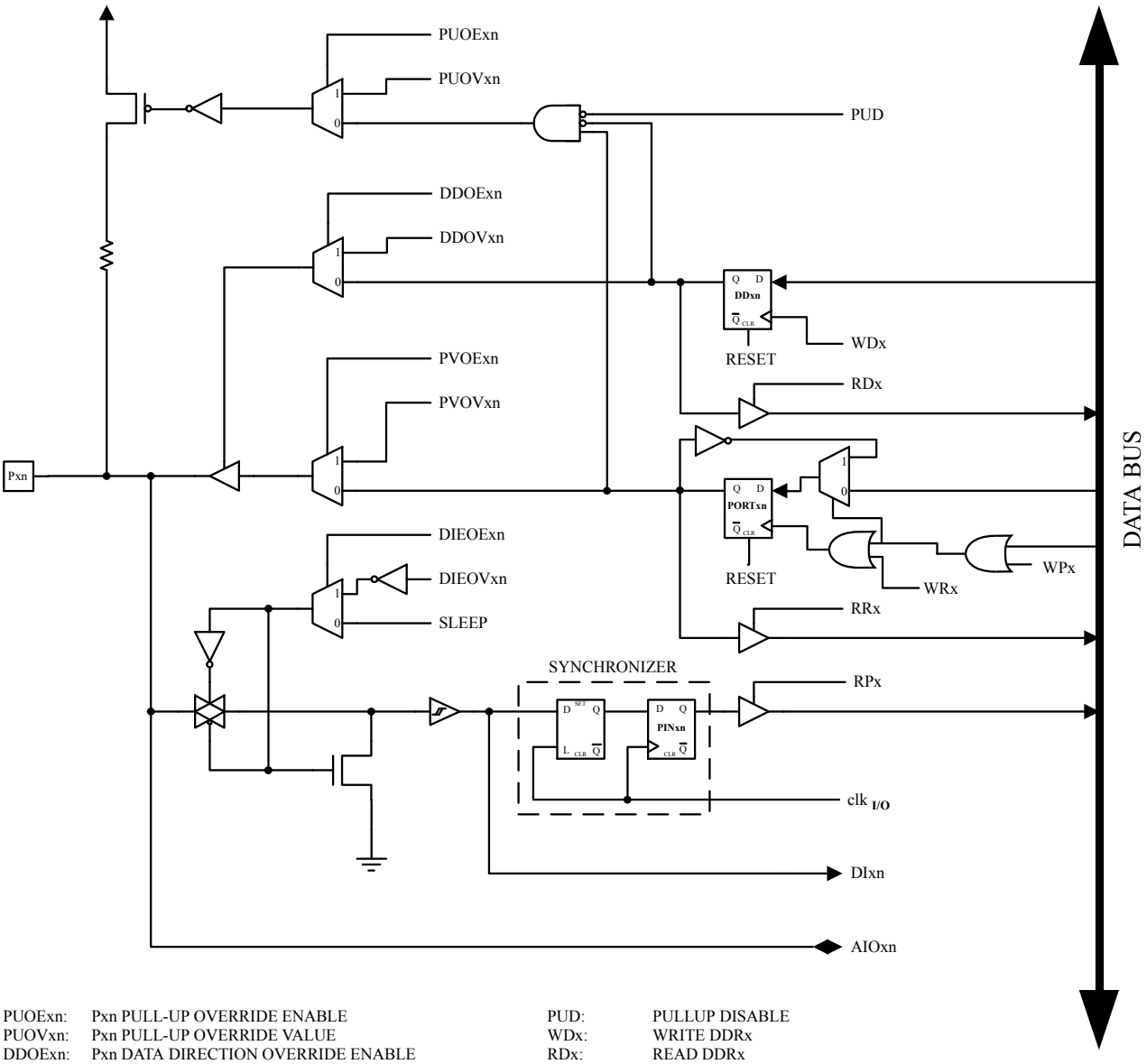
If some pins are unused, it is recommended to ensure that these pins have a defined level. Even though most of the digital inputs are disabled in the deep sleep modes as described above, floating inputs should be avoided to reduce current consumption in all other modes where the digital inputs are enabled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

## 18.3. Alternate Port Functions

Most port pins have alternate functions in addition to being general digital I/Os. The following figure shows how the port pin control signals from the simplified Figure 18-2 can be overridden by alternate functions. The overriding signals may not be present in all port pins, but the figure serves as a generic description applicable to all port pins in the AVR microcontroller family.

**Figure 18-5. Alternate Port Functions**[1]




| | |
|---|---|
| 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 |

| | |
|---|---|
| PUD: | PULLUP DISABLE |
| 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:** 1. WRx, WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports. All other signals are unique for each pin.

The following table summarizes the function of the overriding signals. The pin and port indexes from previous figure are not shown in the succeeding tables. The overriding signals are generated internally in the modules having the alternate function.

**Table 18-2. Generic Description of Overriding Signals for Alternate Functions**

| Signal Name | 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 {DDxn, PORTxn, PUD} = 0b010. |
| 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 DDxn, PORTxn, and PUD Register bits. |
| 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. |
| 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 alternate 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 alternate function will use its own synchronizer. |
| AIO | Analog Input/ Output | This is the Analog Input/output to/from alternate functions. The signal is connected directly to the pad, and can be used bi-directionally. |

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

### 18.3.1. Alternate Functions of Port B

The Port B pins with alternate functions are shown in the table below:

**Table 18-3. Port B Pins Alternate Functions**

| Port Pin | Alternate Functions |
|---|---|
| PB7 | XTAL2 (Chip Clock Oscillator pin 2)<br>TOSC2 (Timer Oscillator pin 2)<br>PCINT7 (Pin Change Interrupt 7) |
| PB6 | XTAL1 (Chip Clock Oscillator pin 1 or External clock input)<br>TOSC1 (Timer Oscillator pin 1)<br>PCINT6 (Pin Change Interrupt 6) |
| PB5 | SCK (SPI Bus Master clock Input)<br>PCINT5 (Pin Change Interrupt 5) |
| PB4 | MISO (SPI Bus Master Input/Slave Output)<br>PCINT4 (Pin Change Interrupt 4) |
| PB3 | MOSI (SPI Bus Master Output/Slave Input)<br>OC2A (Timer/Counter2 Output Compare Match A Output)<br>PCINT3 (Pin Change Interrupt 3) |
| PB2 | $\overline{SS}$ (SPI Bus Master Slave select)<br>OC1B (Timer/Counter1 Output Compare Match B Output)<br>PCINT2 (Pin Change Interrupt 2) |
| PB1 | OC1A (Timer/Counter1 Output Compare Match A Output)<br>PCINT1 (Pin Change Interrupt 1) |
| PB0 | ICP1 (Timer/Counter1 Input Capture Input)<br>CLKO (Divided System Clock Output)<br>PCINT0 (Pin Change Interrupt 0) |

The alternate pin configuration is as follows:

- XTAL2/TOSC2/PCINT7 – Port B, Bit 7

- XTAL2: Chip clock Oscillator pin 2. Used as clock pin for crystal Oscillator or Low-frequency crystal Oscillator. When used as a clock pin, the pin can not be used as an I/O pin.
- TOSC2: Timer Oscillator pin 2. Used only if internal calibrated RC Oscillator is selected as chip clock source, and the asynchronous timer is enabled by the correct setting in ASSR. When the AS2 bit in ASSR is set (one) and the EXCLK bit is cleared (zero) to enable asynchronous clocking of Timer/Counter2 using the Crystal Oscillator, pin PB7 is disconnected from the port, and becomes the inverting output of the Oscillator amplifier. In this mode, a crystal Oscillator is connected to this pin, and the pin cannot be used as an I/O pin.
- PCINT7: Pin Change Interrupt source 7. The PB7 pin can serve as an external interrupt source.

If PB7 is used as a clock pin, DDB7, PORTB7 and PINB7 will all read 0.

- XTAL1/TOSC1/PCINT6 – Port B, Bit 6
  - XTAL1: Chip clock Oscillator pin 1. Used for all chip clock sources except internal calibrated RC Oscillator. When used as a clock pin, the pin can not be used as an I/O pin.
  - TOSC1: Timer Oscillator pin 1. Used only if internal calibrated RC Oscillator is selected as chip clock source, and the asynchronous timer is enabled by the correct setting in ASSR. When the AS2 bit in ASSR is set (one) to enable asynchronous clocking of Timer/Counter2, pin PB6 is disconnected from the port, and becomes the input of the inverting Oscillator amplifier. In this mode, a crystal Oscillator is connected to this pin, and the pin can not be used as an I/O pin.
  - PCINT6: Pin Change Interrupt source 6. The PB6 pin can serve as an external interrupt source.

If PB6 is used as a clock pin, DDB6, PORTB6 and PINB6 will all read 0.

- SCK/PCINT5 – Port B, Bit 5
  - SCK: Master Clock output, Slave Clock input pin for SPI channel. When the SPI is enabled as a Slave, this pin is configured as an input regardless of the setting of DDB5. When the SPI is enabled as a Master, the data direction of this pin is controlled by DDB5. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB5 bit.
  - PCINT5: Pin Change Interrupt source 5. The PB5 pin can serve as an external interrupt source.
- MISO/PCINT4 – Port B, Bit 4
  - MISO: Master Data input, Slave Data output pin for SPI channel. When the SPI is enabled as a Master, this pin is configured as an input regardless of the setting of DDB4. When the SPI is enabled as a Slave, the data direction of this pin is controlled by DDB4. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB4 bit.
  - PCINT4: Pin Change Interrupt source 4. The PB4 pin can serve as an external interrupt source.
- MOSI/OC2A/PCINT3 – Port B, Bit 3
  - MOSI: SPI Master Data output, Slave Data input for SPI channel. When the SPI is enabled as a Slave, this pin is configured as an input regardless of the setting of DDB3. When the SPI is enabled as a Master, the data direction of this pin is controlled by DDB3. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB3 bit.
  - OC2A: Output Compare Match output. The PB3 pin can serve as an external output for the Timer/Counter2 Compare Match A. The PB3 pin has to be configured as an output (DDB3 set

'1') to serve this function. The OC2A pin is also the output pin for the PWM mode timer function.
  - PCINT3: Pin Change Interrupt source 3. The PB3 pin can serve as an external interrupt source.
- $\overline{SS}$/OC1B/PCINT2 – Port B, Bit 2
  - $\overline{SS}$: Slave Select input. When the SPI is enabled as a Slave, this pin is configured as an input regardless of the setting of DDB2. As a Slave, the SPI is activated when this pin is driven low. When the SPI is enabled as a Master, the data direction of this pin is controlled by DDB2. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB2 bit.
  - OC1B: Output Compare Match output. The PB2 pin can serve as an external output for the Timer/Counter1 Compare Match B. The PB2 pin has to be configured as an output (DDB2 set (one)) to serve this function. The OC1B pin is also the output pin for the PWM mode timer function.
  - PCINT2: Pin Change Interrupt source 2. The PB2 pin can serve as an external interrupt source.
- OC1A/PCINT1 – Port B, Bit 1
  - OC1A: Output Compare Match output. The PB1 pin can serve as an external output for the Timer/Counter1 Compare Match A. The PB1 pin has to be configured as an output (DDB1 set (one)) to serve this function. The OC1A pin is also the output pin for the PWM mode timer function.
  - PCINT1: Pin Change Interrupt source 1. The PB1 pin can serve as an external interrupt source.
- ICP1/CLKO/PCINT0 – Port B, Bit 0
  - ICP1: Input Capture Pin. The PB0 pin can act as an Input Capture Pin for Timer/Counter1.
  - CLKO: Divided System Clock. The divided system clock can be output on the PB0 pin. The divided system clock will be output if the CKOUT Fuse is programmed, regardless of the PORTB0 and DDB0 settings. It will also be output during reset.
  - PCINT0: Pin Change Interrupt source 0. The PB0 pin can serve as an external interrupt source.

The tables below relate the alternate functions of Port B to the overriding signals shown in Figure 18-5. SPI MSTR INPUT and SPI SLAVE OUTPUT constitute the MISO signal, while MOSI is divided into SPI MSTR OUTPUT and SPI SLAVE INPUT.

**Table 18-4. Overriding Signals for Alternate Functions in PB7...PB4**

| Signal Name | PB7/XTAL2/TOSC2/PCINT7(1) | PB6/XTAL1/TOSC1/ PCINT6(1) | PB5/SCK/PCINT5 | PB4/MISO/PCINT4 |
|---|---|---|---|---|
| PUOE | $\overline{INTRC}$ • $\overline{EXTCK}$+ AS2 | INTRC + AS2 | SPE • $\overline{MSTR}$ | SPE • MSTR |
| PUOV | 0 | 0 | PORTB5 • $\overline{PUD}$ | PORTB4 • $\overline{PUD}$ |
| DDOE | $\overline{INTRC}$ • $\overline{EXTCK}$+ AS2 | INTRC + AS2 | SPE • $\overline{MSTR}$ | SPE • MSTR |
| DDOV | 0 | 0 | 0 | 0 |
| PVOE | 0 | 0 | SPE • MSTR | SPE • $\overline{MSTR}$ |
| PVOV | 0 | 0 | SCK OUTPUT | SPI SLAVE OUTPUT |

| Signal Name | PB7/XTAL2/TOSC2/PCINT7(1) | PB6/XTAL1/TOSC1/ PCINT6(1) | PB5/SCK/PCINT5 | PB4/MISO/PCINT4 |
|---|---|---|---|---|
| DIEOE | $\overline{\text{INTRC}}$ • $\overline{\text{EXTCK}}$ + AS2 + PCINT7 • PCIE0 | $\overline{\text{INTRC}}$ + AS2 + PCINT6 • PCIE0 | PCINT5 • PCIE0 | PCINT4 • PCIE0 |
| DIEOV | (INTRC + EXTCK) • $\overline{\text{AS2}}$ | INTRC • $\overline{\text{AS2}}$ | 1 | 1 |
| DI | PCINT7 INPUT | PCINT6 INPUT | PCINT5 INPUT<br>SCK INPUT | PCINT4 INPUT<br>SPI MSTR INPUT |
| AIO | Oscillator Output | Oscillator/Clock Input | – | – |

Notes: 1. INTRC means that one of the internal RC Oscillators are selected (by the CKSEL fuses), EXTCK means that external clock is selected (by the CKSEL fuses).

**Table 18-5. Overriding Signals for Alternate Functions in PB3...PB0**

| Signal Name | PB3/MOSI/TXD1/OC2A/PCINT3 | PB2/SS/OC1B/PCINT2 | PB1/OC1A/PCINT1 | PB0/ICP1/CLKO/ PCINT0 |
|---|---|---|---|---|
| PUOE | SPE • $\overline{\text{MSTR}}$ + TXEN1 | SPE • $\overline{\text{MSTR}}$ | 0 | 0 |
| PUOV | PORTB3 • $\overline{\text{PUD}}$ | PORTB2 • $\overline{\text{PUD}}$ | 0 | 0 |
| DDOE | SPE • $\overline{\text{MSTR}}$ + TXEN1 | SPE • $\overline{\text{MSTR}}$ | 0 | 0 |
| DDOV | 0 | 0 | 0 | 0 |
| PVOE | SPE • MSTR + OC2A ENABLE | OC1B ENABLE | OC1A ENABLE | 0 |
| PVOV | SPI MSTR OUTPUT + OC2A + TXD1 | OC1B | OC1A | 0 |
| DIEOE | PCINT3 • PCIE0 | PCINT2 • PCIE0 | PCINT1 • PCIE0 | PCINT0 • PCIE0 |
| DIEOV | 1 | 1 | 1 | 1 |
| DI | PCINT3 INPUT<br>SPI SLAVE INPUT | PCINT2 INPUT<br>SPI SS | PCINT1 INPUT | PCINT0 INPUT<br>ICP1 INPUT |
| AIO | – | – | – | – |

### 18.3.2. Alternate Functions of Port C

The Port C pins with alternate functions are shown in the table below:

**Table 18-6. Port C Pins Alternate Functions**

| Port Pin | Alternate Function |
|---|---|
| PC6 | RESET (Reset pin)<br>PCINT14 (Pin Change Interrupt 14) |
| PC5 | ADC5 (ADC Input Channel 5)<br>SCL (2-wire Serial Bus Clock Line)<br>PCINT13 (Pin Change Interrupt 13) |

| Port Pin | Alternate Function |
|---|---|
| PC4 | ADC4 (ADC Input Channel 4)<br>SDA (2-wire Serial Bus Data Input/Output Line)<br>PCINT12 (Pin Change Interrupt 12) |
| PC3 | ADC3 (ADC Input Channel 3)<br>PCINT11 (Pin Change Interrupt 11) |
| PC2 | ADC2 (ADC Input Channel 2)<br>PCINT10 (Pin Change Interrupt 10) |
| PC1 | ADC1 (ADC Input Channel 1)<br>PCINT9 (Pin Change Interrupt 9) |
| PC0 | ADC0 (ADC Input Channel 0)<br>PCINT8 (Pin Change Interrupt 8) |

The alternate pin configuration is as follows:

- $\overline{\text{RESET}}$/PCINT14 – Port C, Bit 6
  - $\overline{\text{RESET}}$: Reset pin. When the RSTDISBL Fuse is programmed, this pin functions as a normal I/O pin, and the part will have to rely on Power-on Reset and Brown-out Reset as its reset sources. When the RSTDISBL Fuse is unprogrammed, the reset circuitry is connected to the pin, and the pin can not be used as an I/O pin.
  - PCINT14: Pin Change Interrupt source 14. The PC6 pin can serve as an external interrupt source.

If PC6 is used as a reset pin, DDC6, PORTC6 and PINC6 will all read 0.

- SCL/ADC5/PCINT13 – Port C, Bit 5
  - SCL: 2-wire Serial Interface Clock. When the TWEN bit in TWCR is set (one) to enable the 2-wire Serial Interface, pin PC5 is disconnected from the port and becomes the Serial Clock I/O pin for the 2-wire Serial Interface. In this mode, there is a spike filter on the pin to suppress spikes shorter than 50 ns on the input signal, and the pin is driven by an open drain driver with slew-rate limitation.
  - PCINT13: Pin Change Interrupt source 13. The PC5 pin can serve as an external interrupt source.
  - PC5 can also be used as ADC input Channel 5. The ADC input channel 5 uses digital power.
- SDA/ADC4/PCINT12 – Port C, Bit 4
  - SDA: 2-wire Serial Interface Data. When the TWEN bit in TWCR is set (one) to enable the 2-wire Serial Interface, pin PC4 is disconnected from the port and becomes the Serial Data I/O pin for the 2-wire Serial Interface. In this mode, there is a spike filter on the pin to suppress spikes shorter than 50 ns on the input signal, and the pin is driven by an open drain driver with slew-rate limitation.
  - PCINT12: Pin Change Interrupt source 12. The PC4 pin can serve as an external interrupt source.
  - PC4 can also be used as ADC input Channel 4. The ADC input channel 4 uses digital power.
- ADC3/PCINT11 – Port C, Bit 3
  - PC3 can also be used as ADC input Channel 3. The ADC input channel 3 uses analog power.
  - PCINT11: Pin Change Interrupt source 11. The PC3 pin can serve as an external interrupt source.

- ADC2/PCINT10 – Port C, Bit 2
  - PC2 can also be used as ADC input Channel 2. The ADC input channel 2 uses analog power.
  - PCINT10: Pin Change Interrupt source 10. The PC2 pin can serve as an external interrupt source.
- ADC1/PCINT9 – Port C, Bit 1
  - PC1 can also be used as ADC input Channel 1. The ADC input channel 1 uses analog power.
  - PCINT9: Pin Change Interrupt source 9. The PC1 pin can serve as an external interrupt source.
- ADC0//CINT8 – Port C, Bit 0
  - PC0 can also be used as ADC input Channel 0. The ADC input channel 0 uses analog power.
  - PCINT8: Pin Change Interrupt source 8. The PC0 pin can serve as an external interrupt source.

The tables below relate the alternate functions of Port C to the overriding signals shown in Figure 18-5.

**Table 18-7. Overriding Signals for Alternate Functions in PC6...PC4[1]**

| Signal Name | PC6/$\overline{\text{RESET}}$/PCINT14 | PC5/SCL/ADC5/PCINT13 | PC4/SDA/ADC4/PCINT12 |
|---|---|---|---|
| PUOE | RSTDISBL | TWEN | TWEN |
| PUOV | 1 | PORTC5 • $\overline{\text{PUD}}$ | PORTC4 • $\overline{\text{PUD}}$ |
| DDOE | RSTDISBL | TWEN | TWEN |
| DDOV | 0 | SCL_OUT | SDA_OUT |
| PVOE | 0 | TWEN | TWEN |
| PVOV | 0 | 0 | 0 |
| DIEOE | RSTDISBL + PCINT14 • PCIE1 | PCINT13 • PCIE1 + ADC5D | PCINT12 • PCIE1 + ADC4D |
| DIEOV | RSTDISBL | PCINT13 • PCIE1 | PCINT12 • PCIE1 |
| DI | PCINT14 INPUT | PCINT13 INPUT | PCINT12 INPUT |
| AIO | RESET INPUT | ADC5 INPUT / SCL INPUT | ADC4 INPUT / SDA INPUT |

**Note:** 1. When enabled, the 2-wire Serial Interface enables slew-rate controls on the output pins PC4 and PC5. This is not shown in the figure. In addition, spike filters are connected between the AIO outputs shown in the port figure and the digital logic of the TWI module.

**Table 18-8. Overriding Signals for Alternate Functions in PC3...PC0**

| Signal Name | PC3/ADC3/ PCINT11 | PC2/ADC2/ PCINT10 | PC1/ADC1/ PCINT9 | PC0/ADC0/ PCINT8 |
|---|---|---|---|---|
| PUOE | 0 | 0 | 0 | 0 |
| PUOV | 0 | 0 | 0 | 0 |
| DDOE | 0 | 0 | 0 | 0 |
| DDOV | 0 | 0 | 0 | 0 |

| Signal Name | PC3/ADC3/ PCINT11 | PC2/ADC2/ PCINT10 | PC1/ADC1/ PCINT9 | PC0/ADC0/ PCINT8 |
|---|---|---|---|---|
| PVOE | 0 | 0 | 0 | 0 |
| PVOV | 0 | 0 | 0 | 0 |
| DIEOE | PCINT11 • PCIE1 + ADC3D | PCINT10 • PCIE1 + ADC2D | PCINT9 • PCIE1 + ADC1D | PCINT8 • PCIE1 + ADC0D |
| DIEOV | PCINT11 • PCIE1 | PCINT10 • PCIE1 | PCINT9 • PCIE1 | PCINT8 • PCIE1 |
| DI | PCINT11 INPUT | PCINT10 INPUT | PCINT9 INPUT | PCINT8 INPUT |
| AIO | ADC3 INPUT | ADC2 INPUT | ADC1 INPUT | ADC0 INPUT |

### 18.3.3. Alternate Functions of Port D

The Port D pins with alternate functions are shown in the table below:

**Table 18-9. Port D Pins Alternate Functions**

| Port Pin | Alternate Function |
|---|---|
| PD7 | AIN1 (Analog Comparator Negative Input)<br>PCINT23 (Pin Change Interrupt 23) |
| PD6 | AIN0 (Analog Comparator Positive Input)<br>OC0A (Timer/Counter0 Output Compare Match A Output)<br>PCINT22 (Pin Change Interrupt 22) |
| PD5 | T1 (Timer/Counter 1 External Counter Input)<br>OC0B (Timer/Counter0 Output Compare Match B Output)<br>PCINT21 (Pin Change Interrupt 21) |
| PD4 | XCK (USART External Clock Input/Output)<br>T0 (Timer/Counter 0 External Counter Input)<br>PCINT20 (Pin Change Interrupt 20) |
| PD3 | INT1 (External Interrupt 1 Input)<br>OC2B (Timer/Counter2 Output Compare Match B Output)<br>PCINT19 (Pin Change Interrupt 19) |

| Port Pin | Alternate Function |
|---|---|
| PD2 | INT0 (External Interrupt 0 Input)<br><br>PCINT18 (Pin Change Interrupt 18) |
| PD1 | TXD (USART Output Pin)<br><br>PCINT17 (Pin Change Interrupt 17) |
| PD0 | RXD (USART Input Pin)<br><br>PCINT16 (Pin Change Interrupt 16) |

The alternate pin configuration is as follows:

- AIN1/OC2B/PCINT23 – Port D, Bit 7
  - AIN1: Analog Comparator1 Negative Input. Configure the 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.
  - PCINT23: Pin Change Interrupt source 23. The PD7 pin can serve as an external interrupt source.
- AIN0/OC0A/PCINT22 – Port D, Bit 6
  - AIN0: Analog Comparator0 Positive Input. Configure the 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.
  - OC0A: Output Compare Match output. The PD6 pin can serve as an external output for the Timer/Counter0 Compare Match A. The PD6 pin has to be configured as an output (DDD6 set (one)) to serve this function. The OC0A pin is also the output pin for the PWM mode timer function.
  - PCINT22: Pin Change Interrupt source 22. The PD6 pin can serve as an external interrupt source.
- T1/OC0B/PCINT21 – Port D, Bit 5
  - T1: Timer/Counter1 counter source.
  - OC0B: Output Compare Match output. The PD5 pin can serve as an external output for the Timer/Counter0 Compare Match B. The PD5 pin has to be configured as an output (DDD5 set (one)) to serve this function. The OC0B pin is also the output pin for the PWM mode timer function.
  - PCINT21: Pin Change Interrupt source 21. The PD5 pin can serve as an external interrupt source.
- XCK/T0/PCINT20 – Port D, Bit 4
  - XCK: USART external clock.
  - T0: Timer/Counter0 counter source.
  - PCINT20: Pin Change Interrupt source 20. The PD4 pin can serve as an external interrupt source.
- INT1/OC2B/PCINT19 – Port D, Bit 3

- INT1: External Interrupt source 1. The PD3 pin can serve as an external interrupt source.
- OC2B: Output Compare Match output: The PD3 pin can serve as an external output for the Timer/Counter2 Compare Match B. The PD3 pin has to be configured as an output (DDD3 set (one)) to serve this function. The OC2B pin is also the output pin for the PWM mode timer function.
- PCINT19: Pin Change Interrupt source 19. The PD3 pin can serve as an external interrupt source.

- INT0/PCINT18 – Port D, Bit 2
  - INT0: External Interrupt source 0. The PD2 pin can serve as an external interrupt source.
  - PCINT18: Pin Change Interrupt source 18. The PD2 pin can serve as an external interrupt source.
- TXD/PCINT17 – Port D, Bit 1
  - TXD: Transmit Data (Data output pin for the USART). When the USART Transmitter is enabled, this pin is configured as an output regardless of the value of DDD1.
  - PCINT17: Pin Change Interrupt source 17. The PD1 pin can serve as an external interrupt source.
- RXD/PCINT16 – Port D, Bit 0
  - RXD: Receive Data (Data input pin for the USART). When the USART Receiver is enabled this pin is configured as an input regardless of the value of DDD0. When the USART forces this pin to be an input, the pull-up can still be controlled by the PORTD0 bit.
  - PCINT16: Pin Change Interrupt source 16. The PD0 pin can serve as an external interrupt source.

The tables below relate the alternate functions of Port D to the overriding signals shown in Figure 18-5.

**Table 18-10. Overriding Signals for Alternate Functions PD7...PD4**

| Signal Name | PD7/AIN1 /PCINT23 | PD6/AIN0/ OC0A/PCINT22 | PD5/T1/OC0B/ PCINT21 | PD4/XCK/ T0/PCINT20 |
|---|---|---|---|---|
| PUOE | 0 | 0 | 0 | 0 |
| PUO | 0 | 0 | 0 | 0 |
| DDOE | 0 | 0 | 0 | 0 |
| DDOV | 0 | 0 | 0 | 0 |
| PVOE | 0 | OC0A ENABLE | OC0B ENABLE | UMSEL |
| PVOV | 0 | OC0A | OC0B | XCK OUTPUT |
| DIEOE | PCINT23 • PCIE2 | PCINT22 • PCIE2 | PCINT21 • PCIE2 | PCINT20 • PCIE2 |
| DIEOV | 1 | 1 | 1 | 1 |
| DI | PCINT23 INPUT | PCINT22 INPUT | PCINT21 INPUT / T1 INPUT | PCINT20 INPUT / XCK INPUT / T0 INPUT |
| AIO | AIN1 INPUT | AIN0 INPUT | – | – |

**Table 18-11. Overriding Signals for Alternate Functions in PD3...PD0**

| Signal Name | PD3/OC2B/INT1/ PCINT19 | PD2/INT0/ PCINT18 | PD1/TXD/ PCINT17 | PD0/RXD/ PCINT16 |
|---|---|---|---|---|
| PUOE | 0 | 0 | TXEN | RXEN |
| PUO | 0 | 0 | 0 | PORTD0 • $\overline{PUD}$ |
| DDOE | 0 | 0 | TXEN | RXEN |
| DDOV | 0 | 0 | 1 | 0 |
| PVOE | OC2B ENABLE | 0 | TXEN | 0 |
| PVOV | OC2B | 0 | TXD | 0 |
| DIEOE | INT1 ENABLE + PCINT19 • PCIE2 | INT0 ENABLE + PCINT18 • PCIE1 | PCINT17 • PCIE2 | PCINT16 • PCIE2 |
| DIEOV | 1 | 1 | 1 | 1 |
| DI | PCINT19 INPUT / INT1 INPUT | PCINT18 INPUT / INT0 INPUT | PCINT17 INPUT | PCINT16 INPUT / RXD |
| AIO | – | – | – | – |

## 18.4. Register Description

### 18.4.1. MCU Control Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** MCUCR
**Offset:** 0x55
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x35

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | PUD | | | IVSEL | IVCE |
| Access | | | | R/W | | | R/W | R/W |
| Reset | | | | 0 | | | 0 | 0 |

**Bit 4 – PUD: Pull-up Disable**
When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01).

**Bit 1 – IVSEL: Interrupt Vector Select**
When the IVSEL bit is cleared (zero), the Interrupt Vectors are placed at the start of the Flash memory. When this bit is set (one), the Interrupt Vectors are moved to the beginning of the Boot Loader section of the Flash. The actual address of the start of the Boot Flash Section is determined by the BOOTSZ Fuses. To avoid unintentional changes of Interrupt Vector tables, a special write procedure must be followed to change the IVSEL bit:

1. Write the Interrupt Vector Change Enable (IVCE) bit to one.
2. Within four cycles, write the desired value to IVSEL while writing a zero to IVCE.

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

**Note:** If Interrupt Vectors are placed in the Boot Loader section and Boot Lock bit BLB02 is programmed, interrupts are disabled while executing from the Application section. If Interrupt Vectors are placed in the Application section and Boot Lock bit BLB12 is programed, interrupts are disabled while executing from the Boot Loader section.

**Bit 0 – IVCE: Interrupt Vector Change Enable**
The IVCE bit must be written to logic one to enable change of the IVSEL bit. IVCE is cleared by hardware four cycles after it is written or when IVSEL is written. Setting the IVCE bit will disable interrupts, as explained in the IVSEL description above. See Code Example below.

### Assembly Code Example

```
Move_interrupts:
; Get MCUCR
in    r16, MCUCR
mov   r17, r16
; Enable change of Interrupt Vectors
ori   r16, (1<<IVCE)
out   MCUCR, r16
; Move interrupts to Boot Flash section
ori   r17, (1<<IVSEL)
out   MCUCR, r17
ret
```

### C Code Example

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

### 18.4.2. Port B Data Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** PORTB
**Offset:** 0x25
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x05

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | PORTB7 | PORTB6 | PORTB5 | PORTB4 | PORTB3 | PORTB2 | PORTB1 | PORTB0 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – PORTBn: Port B Data [n = 0:7]**

### 18.4.3. Port B Data Direction Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** DDRB
**Offset:** 0x24
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x04

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DDRB7 | DDRB6 | DDRB5 | DDRB4 | DDRB3 | DDRB2 | DDRB1 | DDRB0 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – DDRBn: Port B Data Direction [n = 7:0]**

### 18.4.4. Port B Input Pins Address

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** PINB
**Offset:** 0x23
**Reset:** N/A
**Property:** When addressing as I/O Register: address offset is 0x03

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | PINB7 | PINB6 | PINB5 | PINB4 | PINB3 | PINB2 | PINB1 | PINB0 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | x | x | x | x | x | x | x | x |

**Bits 7:0 – PINBn: Port B Input Pins Address [n = 7:0]**
Writing to the pin register provides toggle functionality for IO. Refer to Toggling the Pin.

### 18.4.5. Port C Data Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** PORTC
**Offset:** 0x28
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x08

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | PORTC6 | PORTC5 | PORTC4 | PORTC3 | PORTC2 | PORTC1 | PORTC0 |
| Access | | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 6:0 – PORTCn: Port C Data [n = 6:0]**

### 18.4.6. Port C Data Direction Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** DDRC
**Offset:** 0x27
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x07

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | DDRC6 | DDRC5 | DDRC4 | DDRC3 | DDRC2 | DDRC1 | DDRC0 |
| Access | | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 6:0 – DDRCn: Port C Data Direction [n = 6:0]**

### 18.4.7. Port C Input Pins Address

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** PINC
**Offset:** 0x26
**Reset:** N/A
**Property:** When addressing as I/O Register: address offset is 0x06

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | PINC6 | PINC5 | PINC4 | PINC3 | PINC2 | PINC1 | PINC0 |
| Access | | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | | x | x | x | x | x | x | x |

**Bits 6:0 – PINCn: Port C Input Pins Address [n = 6:0]**
Writing to the pin register provides toggle functionality for IO. Refer to Toggling the Pin.

### 18.4.8. Port D Data Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** PORTD
**Offset:** 0x2B
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x0B

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | PORTD7 | PORTD6 | PORTD5 | PORTD4 | PORTD3 | PORTD2 | PORTD1 | PORTD0 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – PORTDn: Port D Data [n = 7:0]**

### 18.4.9. Port D Data Direction Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** DDRD
**Offset:** 0x2A
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x0A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DDRD7 | DDRD6 | DDRD5 | DDRD4 | DDRD3 | DDRD2 | DDRD1 | DDRD0 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – DDRDn: Port D Data Direction [n = 7:0]**

### 18.4.10. Port D Input Pins Address

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** PIND
**Offset:** 0x29
**Reset:** N/A
**Property:** When addressing as I/O Register: address offset is 0x09

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | PIND7 | PIND6 | PIND5 | PIND4 | PIND3 | PIND2 | PIND1 | PIND0 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | x | x | x | x | x | x | x | x |

**Bits 7:0 – PINDn: Port D Input Pins Address [n = 7:0]**
Writing to the pin register provides toggle functionality for IO. Refer to Toggling the Pin.

# 19. TC0 - 8-bit Timer/Counter0 with PWM

**Related Links**

Timer/Counter0 and Timer/Counter1 Prescalers on page 193

## 19.1. 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)

## 19.2. Overview

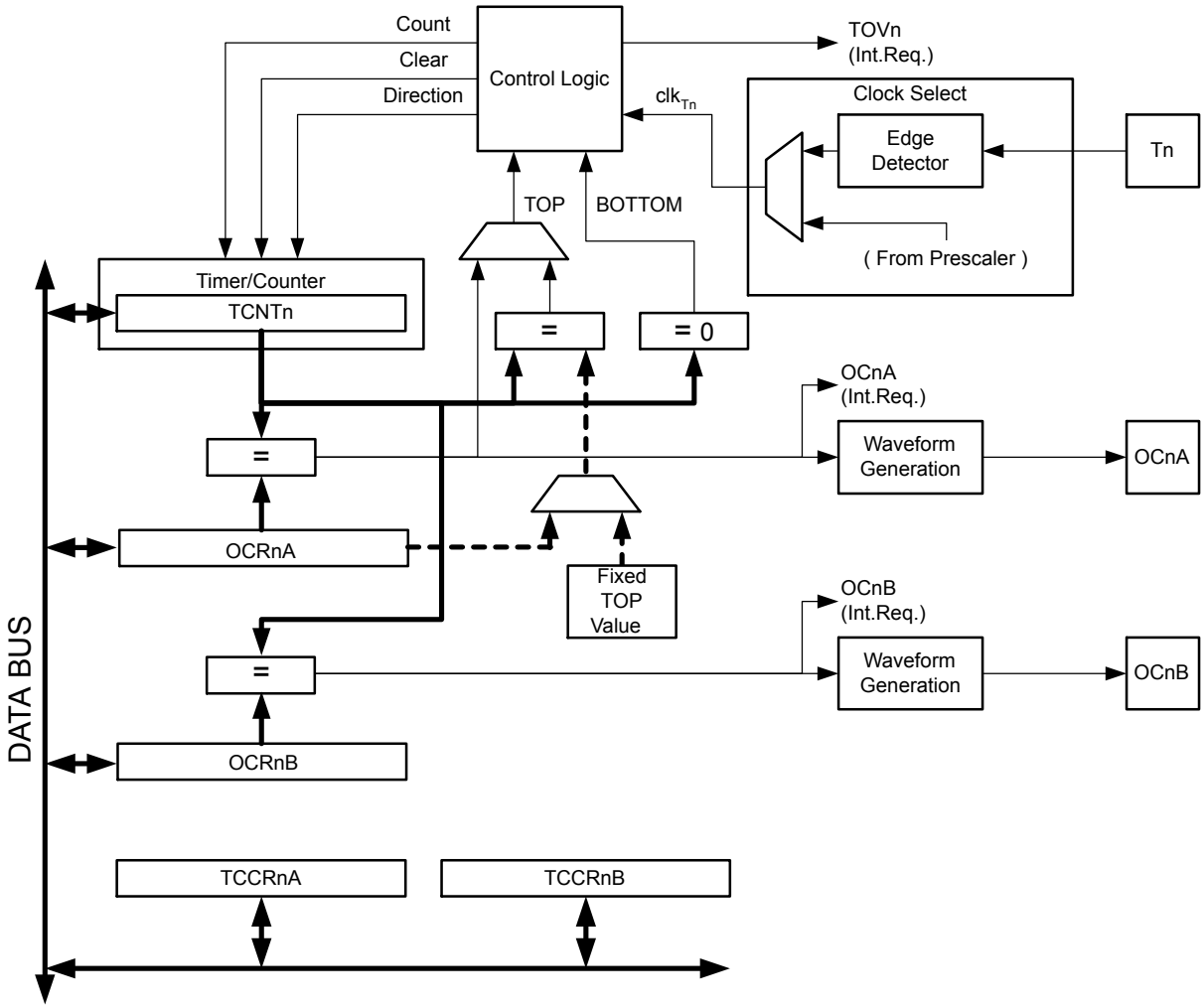
Timer/Counter0 (TC0) is a general purpose 8-bit Timer/Counter module, with two independent Output Compare Units, and PWM support. It allows accurate program execution timing (event management) and wave generation.

A simplified block diagram of the 8-bit Timer/Counter is shown below. 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. For the actual placement of I/O pins, refer to the pinout diagram.

The TC0 is enabled by writing the PRTIM0 bit in ”Minimizing Power Consumption” to '0'.

The TC0 is enabled when the PRTIM0 bit in the Power Reduction Register (PRR.PRTIM0) is written to '1'.

**Figure 19-1. 8-bit Timer/Counter Block Diagram**




### 19.2.1. Definitions

Many register and bit references in this section are written in general form:

- n=0 represents the Timer/Counter number
- x=A,B represents the Output Compare Unit A or B

However, when using the register or bit definitions in a program, the precise form must be used, i.e., TCNT0 for accessing Timer/Counter0 counter value.

The following definitions are used throughout the section:

**Table 19-1. Definitions**

| Constant | Description |
|---|---|
| BOTTOM | The counter reaches the BOTTOM when it becomes zero (0x00 for 8-bit counters, or 0x0000 for 16-bit counters). |

| | |
|---|---|
| MAX | The counter reaches its Maximum when it becomes 0xFF (decimal 255, for 8-bit counters) or 0xFFFF (decimal 65535, for 16-bit counters). |
| 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 MAX or the value stored in the OCR0A Register. The assignment is dependent on the mode of operation. |

### 19.2.2. Registers

The Timer/Counter 0 register (TCNT0) and Output Compare TC0x registers (OCR0x) are 8-bit registers. Interrupt request (abbreviated to Int.Req. in the block diagram) signals are all visible in the Timer Interrupt Flag Register 0 (TIFR0). All interrupts are individually masked with the Timer Interrupt Mask Register 0 (TIMSK0). TIFR0 and TIMSK0 are not shown in the figure.

The TC 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 is used by the Timer/Counter to increment (or decrement) its value. The TC 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) are 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 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.

**Related Links**

Timer/Counter 0, 1 Prescalers on page 193

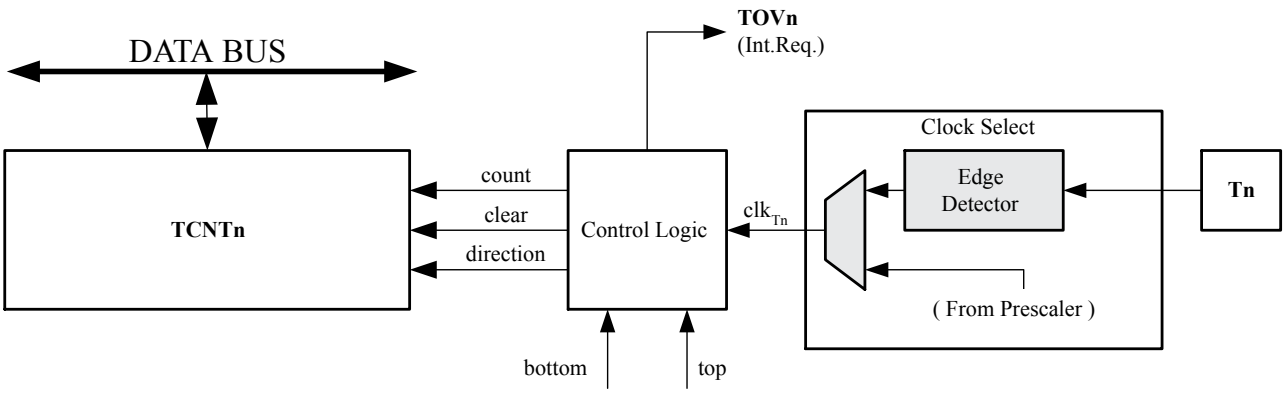
## 19.3. Timer/Counter Clock Sources

The TC can be clocked by an internal or an external clock source. The clock source is selected by writing to the Clock Select (CS0[2:0]) bits in the Timer/Counter Control Register (TCCR0B).

## 19.4. Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Below is the block diagram of the counter and its surroundings.

**Figure 19-2. Counter Unit Block Diagram**




**Note:** The “n” in the register and bit names indicates the device number (n = 0 for Timer/Counter 0), and the “x” indicates Output Compare unit (A/B).

**Table 19-2. Signal description (internal signals)**

| Signal Name | Description |
|---|---|
| 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=0x0) 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.

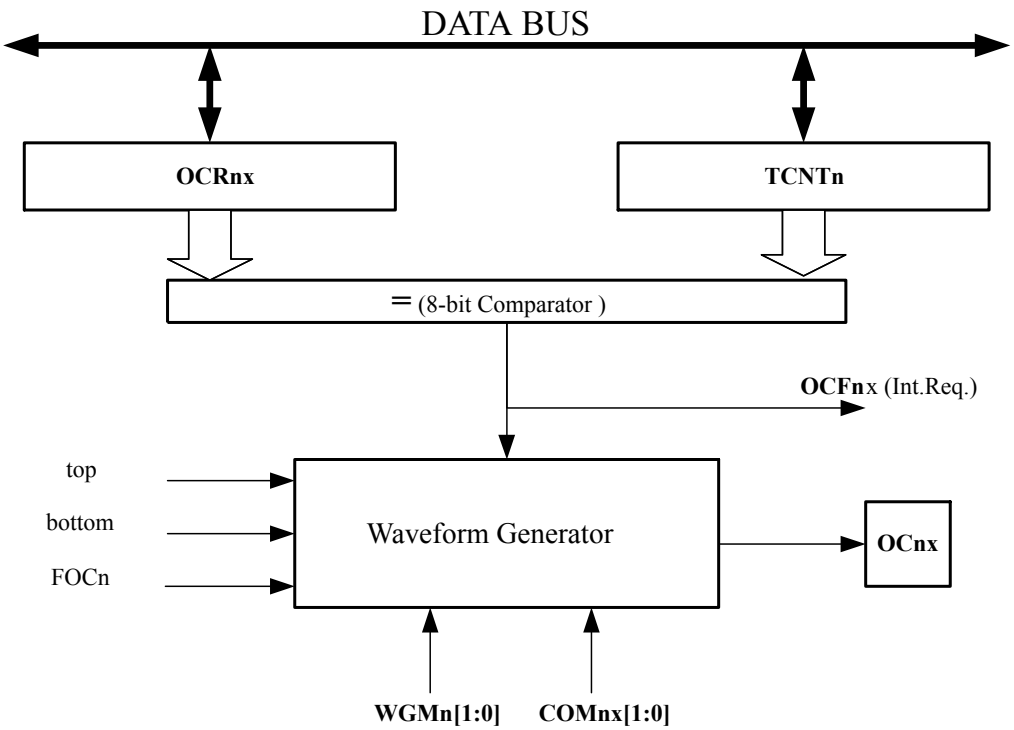
The counting sequence is determined by the setting of the WGM01 and WGM00 bits located in the Timer/Counter Control Register (TCCR0A) and the WGM02 bit located in the Timer/Counter Control Register B (TCCR0B). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare outputs OC0A and OC0B. For more details about advanced counting sequences and waveform generation, see Modes of Operation.

The Timer/Counter Overflow Flag (TOV0) is set according to the mode of operation selected by the WGM0[2:0] bits. TOV0 can be used for generating a CPU interrupt.

## 19.5. 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 '1' to its I/O bit location. The Waveform Generator uses the match signal to generate an output according to operating mode set by the WGM02, WGM01, and WGM00 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.

**Figure 19-3. Output Compare Unit, Block Diagram**




**Note:** The “n” in the register and bit names indicates the device number (n = 0 for Timer/Counter 0), and the “x” indicates Output Compare unit (A/B).

The OCR0x Registers are double buffered when using any of the Pulse Width Modulation (PWM) modes. When double buffering is enabled, the CPU has access to the OCR0x Buffer Register. 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 double buffering is disabled for the normal and Clear Timer on Compare (CTC) modes of operation, and the CPU will access the OCR0x directly.

### 19.5.1. Force Output Compare

In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a '1' to the Force Output Compare (TCCR0C.FOC1x) 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 TCCR1A.COM1x[1:0] bits define whether the OC1x pin is set, cleared or toggled).

### 19.5.2. Compare Match Blocking by TCNT1 Write

All CPU write operations to the TCNT1 Register will block any compare match that occur 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.

### 19.5.3. 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 the Output Compare Unit, independently 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. Similarly, do not write the TCNT1 value equal to BOTTOM when the counter is down counting.

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 (FOC1x) strobe bits in Normal mode. The OC1x Registers keep their values even when changing between Waveform Generation modes.

Be aware that the TCCR1A.COM1x[1:0] bits are not double buffered together with the compare value. Changing the TCCR1A.COM1x[1:0] bits will take effect immediately.

## 19.6. Compare Match Output Unit

The Compare Output mode bits in the Timer/Counter Control Register A (TCCR0A.COM0x) have two functions:

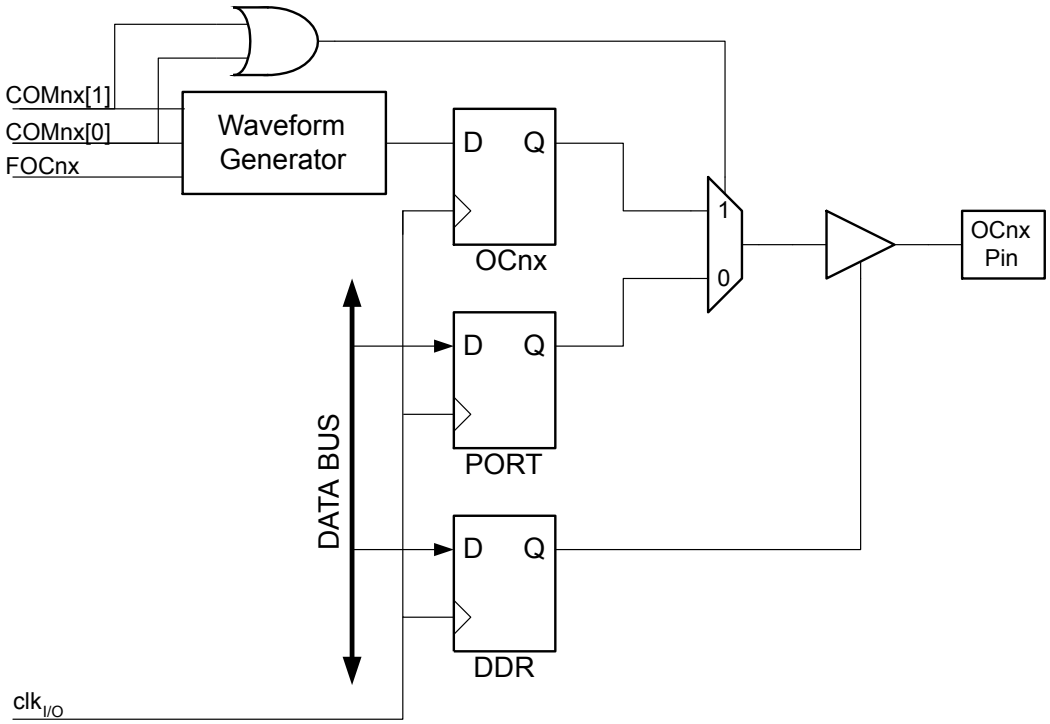
- The Waveform Generator uses the COM0x bits for defining the Output Compare (OC0x) register state at the next compare match.
- The COM0x bits control the OC0x pin output source

The figure below shows a simplified schematic of the logic affected by COM0x. 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 that are affected by the COM0x bits are shown, namely PORT and DDR.

On system reset the OC0x Register is reset to 0x00.

**Note:** 'OC0x state' is always referring to internal OC0x *registers*, not the OC0x *pin*.

**Figure 19-4. Compare Match Output Unit, Schematic**




**Note:** The “n” in the register and bit names indicates the device number (n = 0 for Timer/Counter 0), and the “x” indicates Output Compare unit (A/B).

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. In the Data Direction Register, the bit for the OC1x

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 register state before the output is enabled. Some TCCR0A.COM0x[1:0] bit settings are reserved for certain modes of operation.

The TCCR0A.COM0x[1:0] bits have no effect on the Input Capture unit.

**Related Links**

Register Description on page 144

### 19.6.1. Compare Output Mode and Waveform Generation

The Waveform Generator uses the TCCR0A.COM0x[1:0] bits differently in Normal, CTC, and PWM modes. For all modes, setting the TCCR0A.COM0x[1:0]=0x0 tells the Waveform Generator that no action on the OC0x Register is to be performed on the next compare match. Refer also to the descriptions of the output modes.

A change of the TCCR0A.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 TCCR0C.FOC0x strobe bits.

## 19.7. Modes of Operation

The mode of operation determines the behavior of the Timer/Counter and the Output Compare pins. It is defined by the combination of the Waveform Generation mode bits and Compare Output mode (TCCR0A.WGM0[2:0]) bits in the Timer/Counter control Registers A and B (TCCR0A.COM0x[1:0]). 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 previous section *Compare Match Output Unit*).

For detailed timing information refer to the following section *Timer/Counter Timing Diagrams*.

**Related Links**

Compare Match Output Unit on page 201
Timer/Counter Timing Diagrams on page 142

### 19.7.1. Normal Mode

The simplest mode of operation is the Normal mode (WGM1[2:0] = 0x0). 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 mode operation, the Timer/Counter Overflow Flag (TOV1) will be set in the same clock cycle in which the TCNT1 becomes zero. In this case, the TOV1 Flag behaves like a ninth 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.

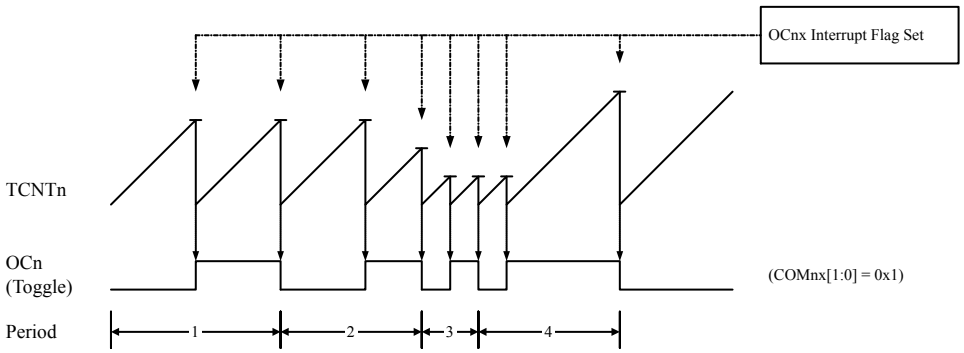
The Output Compare unit can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

### 19.7.2. Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGM0[2:0]=0x2), the OCR0A Register is used to manipulate the counter resolution: 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 counting of external events.

The timing diagram for the CTC mode is shown below. The counter value (TCNT0) increases until a compare match occurs between TCNT0 and OCR0A, and then counter (TCNT0) is cleared.

**Figure 19-5. CTC Mode, Timing Diagram**




An interrupt can be generated each time the counter value reaches the TOP value by setting the OCF0A Flag. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value.

**Note:** Changing TOP to a value close to BOTTOM while the counter is running must be done with care, since the CTC mode does not provide double buffering. 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 count to its maximum value (0xFF for a 8-bit counter, 0xFFFF for a 16-bit counter) and wrap around starting at 0x00 before the compare match will 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 writing the two least significant Compare Output mode bits in the Timer/Counter Control Register A Control to toggle mode (TCCR0A.COM0A[1:0]=0x1). The OC0A value will only be visible on the port pin unless the data direction for the pin is set to output. The waveform generated will have a maximum frequency of $f_{OC0} = f_{clk_I/O}/2$ when OCR0A is written to 0x00. The waveform frequency is defined by the following equation:

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

*N* represents the prescaler factor (1, 8, 64, 256, or 1024).

As for the Normal mode of operation, the Timer/Counter Overflow Flag TOV0 is set in the same clock cycle that the counter wraps from MAX to 0x00.

### 19.7.3. Fast PWM Mode

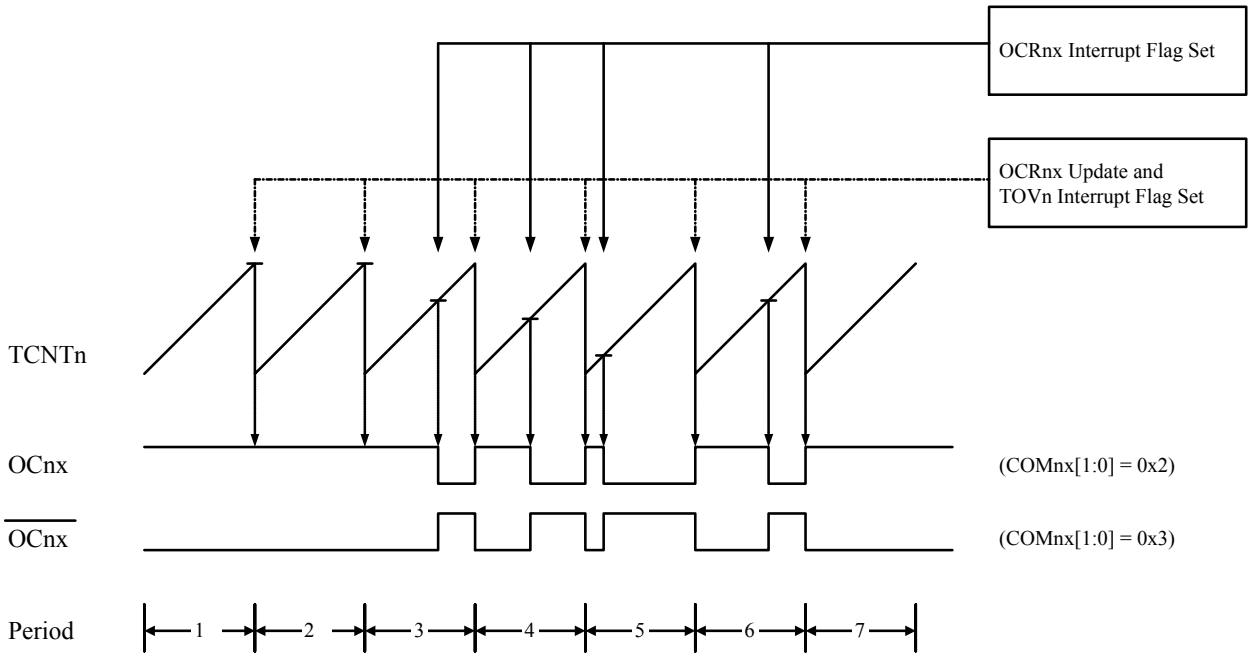
The Fast Pulse Width Modulation or Fast PWM modes (WGM0[2:0]=0x3 or WGM0[2:0]=0x7) provide a high frequency PWM waveform generation option. The Fast PWM modes differ from the other PWM options by their single-slope operation. The counter counts from BOTTOM to TOP, then restarts from BOTTOM. TOP is defined as 0xFF when WGM0[2:0]=0x3. TOP is defined as OCR0A when WGM0[2:0]=0x7.

In non-inverting Compare Output mode, the Output Compare register (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 modes, which 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 below. 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 lines on the TCNT0 slopes mark compare matches between OCR0x and TCNT0.

**Figure 19-6. 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. Writing the TCCR0A.COM0x[1:0] bits to 0x2 will produce a non-inverted PWM; TCCR0A.COM0x[1:0]=0x3 will produce an inverted PWM output. Writing the TCCR0A.COM0A[1:0] bits to 0x1 allows the OC0A pin to toggle on Compare Matches if the TCCRnB.WGMn2 bit is set. This option is not available for the OC0B pin. 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:

$$f_{\mathrm{OCnxPWM}} = \frac{f_{\mathrm{clk_I/O}}}{N \cdot 256}$$

*N* represents the prescale divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR0A register represents special cases for PWM waveform output in the Fast PWM mode: If OCR0A is written equal to BOTTOM, the output will be a narrow spike for each MAX +1 timer clock cycle. Writing OCR0A=MAX will result in a constantly high or low output (depending on the polarity of the output set by the COM0A[1:0] bits.)

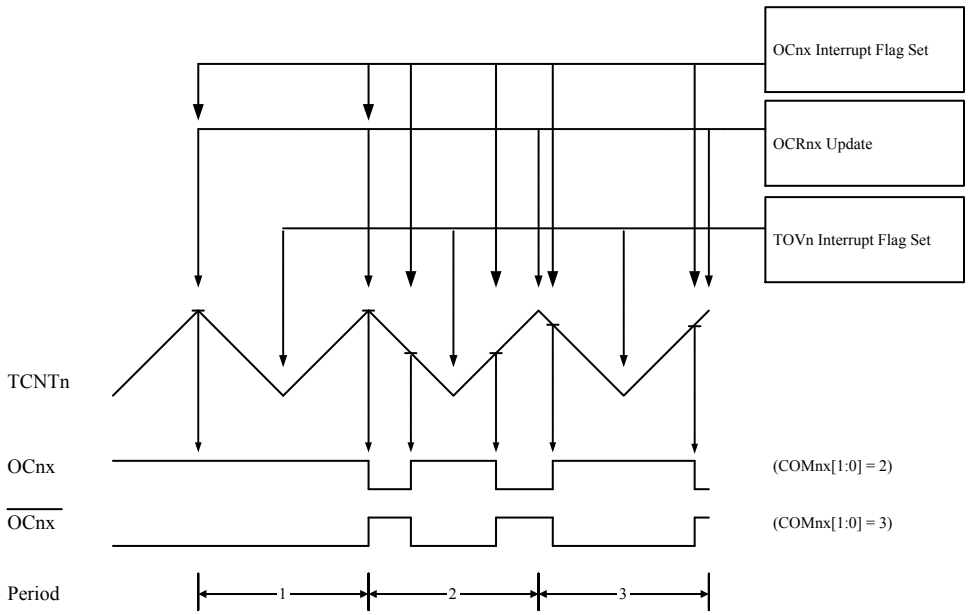
A frequency waveform output with 50% duty cycle can be achieved in Fast PWM mode by selecting OC0x to toggle its logical level on each compare match (COM0x[1:0]=0x1). The waveform generated will have a maximum frequency of $f_{OC0} = f_{clk_I/O}/2$ when OCR0A=0x00. This feature is similar to the OC0A toggle in CTC mode, except double buffering of the Output Compare unit is enabled in the Fast PWM mode.

### 19.7.4. Phase Correct PWM Mode

The Phase Correct PWM mode (WGM0[2:0]=0x1 or WGM0[2:0]=0x5) provides a high resolution, phase correct PWM waveform generation. The Phase Correct PWM mode is based on dual-slope operation: The counter counts repeatedly from BOTTOM to TOP, and then from TOP to BOTTOM. When WGM0[2:0]=0x1 TOP is defined as 0xFF. When WGM0[2:0]=0x5, TOP is defined as OCR0A. In non-inverting Compare Output mode, the Output Compare (OC0x) bit is cleared on compare match between TCNT0 and OCR0x while up-counting, and OC0x is set on the compare match while down-counting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has a lower maximum operation frequency than single slope operation. Due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

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 below. The TCNT0 value is 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.

**Figure 19-7. Phase Correct PWM Mode, Timing Diagram**




**Note:** The “n” in the register and bit names indicates the device number (n = 0 for Timer/Counter 0), and the “x” indicates Output Compare unit (A/B).

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 pin. Writing the COM0x[1:0] bits to 0x2 will produce a non-inverted PWM. An inverted PWM output can be generated by writing COM0x[1:0]=0x3. Setting the Compare Match Output A Mode bit to '1' (TCCR0A.COM0A0) allows the OC0A pin to toggle on Compare Matches if the TCCR0B.WGM02 bit is

set. This option is not available for the OC0B pin. 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:

$$f_{\mathrm{OCnxPCPWM}} = \frac{f_{\mathrm{clk_I/O}}}{N \cdot 510}$$

*N* represents the prescaler factor (1, 8, 64, 256, or 1024).

The extreme values for the OCR0A Register represent special cases when generating a PWM waveform output in the Phase Correct PWM mode: If the OCR0A register is written equal to BOTTOM, the output will be continuously low. If OCR0A is written to MAX, 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 the timing diagram above, OC0x has a transition from high to low even though there is no Compare Match. This transition serves to guarantee symmetry around BOTTOM. There are two cases that give a transition without Compare Match:

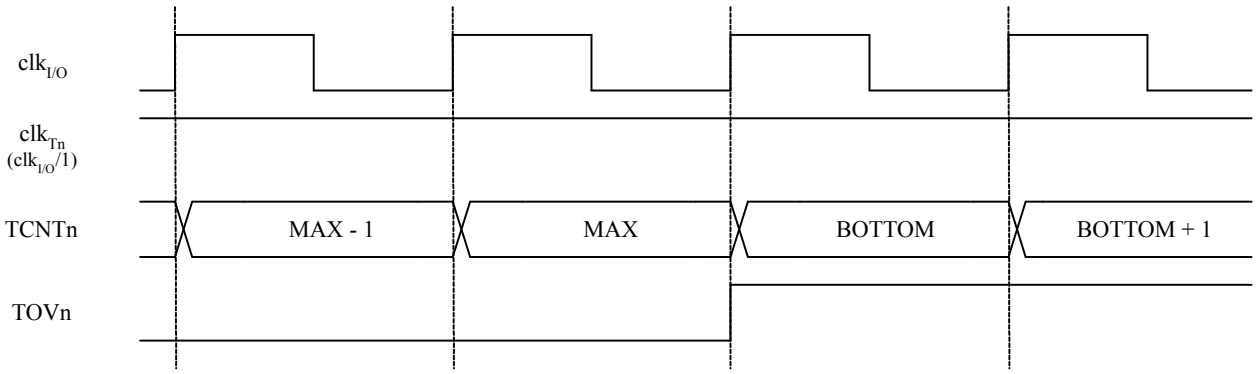
- OCR0x changes its value from MAX, as in the timing diagram. When the OCR0A value is MAX, the OC0 pin value is the same as the result of a down-counting Compare Match. To ensure symmetry around BOTTOM the OC0x value at MAX must correspond to the result of an up-counting Compare Match.
- The timer starts up-counting from a value higher than the one in OCR0x, and for that reason misses the Compare Match and consequently, the OC0x does not undergo the change that would have happened on the way up.

## 19.8. 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. If the given instance of the TC0 supports an asynchronous mode, $clk_{I/O}$ should be replaced by the TC oscillator clock.

The figures include information on when interrupt flags are set. The first figure below illustrates timing data for basic Timer/Counter operation close to the MAX value in all modes other than Phase Correct PWM mode.

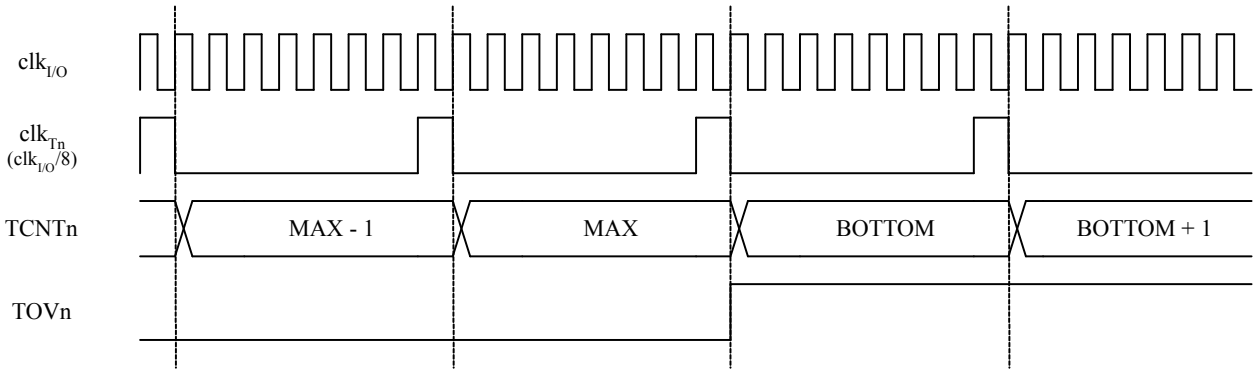
**Figure 19-8. Timer/Counter Timing Diagram, no Prescaling**




**Note:** The “n” in the register and bit names indicates the device number (n = 0 for Timer/Counter 0), and the “x” indicates Output Compare unit (A/B).

The next figure shows the same timing data, but with the prescaler enabled.

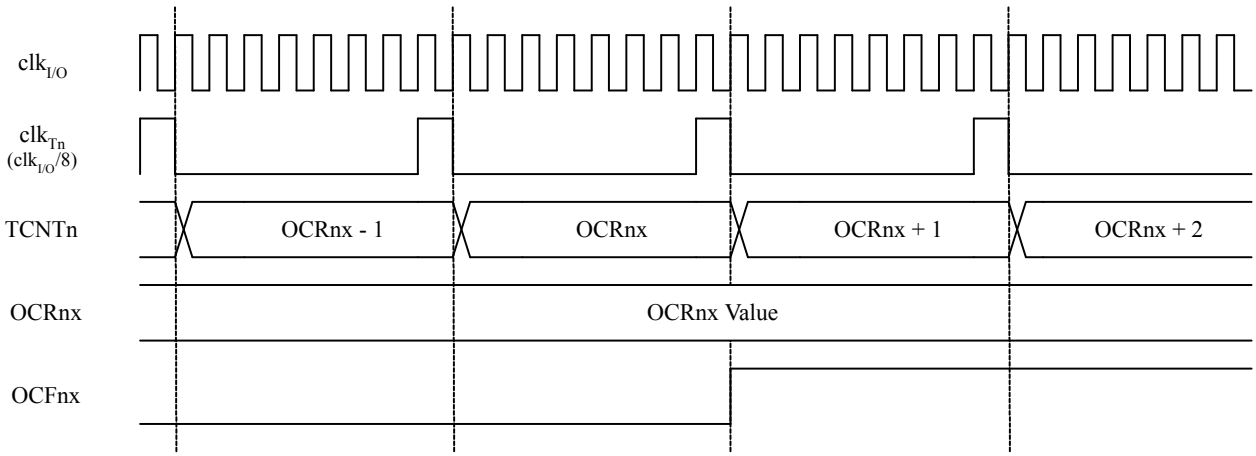
**Figure 19-9. Timer/Counter Timing Diagram, with Prescaler ($f_{clk_I/O}/8$)**




**Note:** The “n” in the register and bit names indicates the device number (n = 0 for Timer/Counter 0), and the “x” indicates Output Compare unit (A/B).

The next figure shows the setting of OCF0B in all modes and OCF0A in all modes (except CTC mode and PWM mode where OCR0A is TOP).

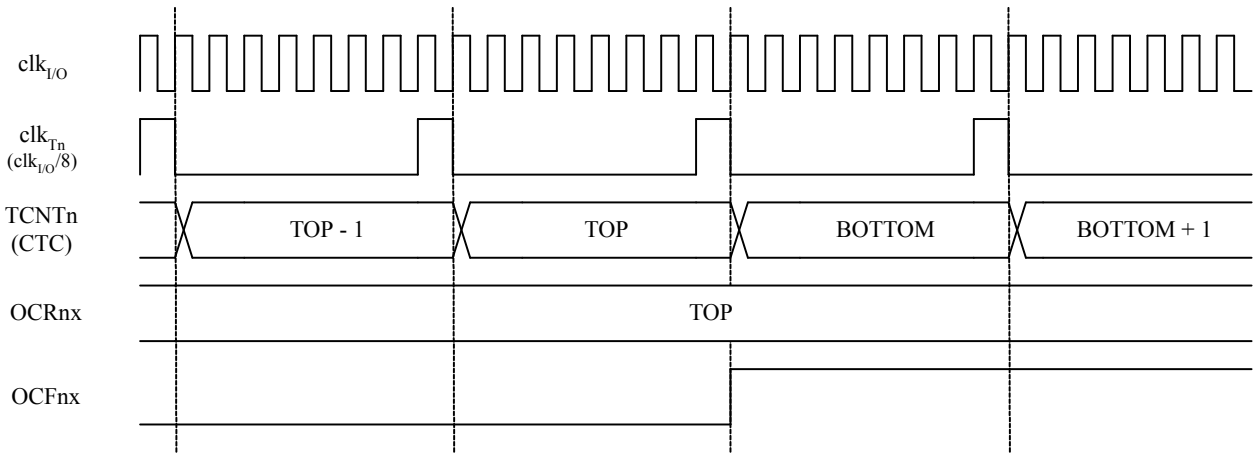
**Figure 19-10. Timer/Counter Timing Diagram, Setting of OCF0x, with Prescaler ($f_{clk_I/O}/8$)**




**Note:** The “n” in the register and bit names indicates the device number (n = 0 for Timer/Counter 0), and the “x” indicates Output Compare unit (A/B).

The next figure shows the setting of OCF0A and the clearing of TCNT0 in CTC mode and fast PWM mode where OCR0A is TOP.

**Figure 19-11. Timer/Counter Timing Diagram, Clear Timer on Compare Match mode, with Prescaler ($f_{clk_I/O}/8$)**



**Note:** The “n” in the register and bit names indicates the device number (n = 0 for Timer/Counter 0), and the “x” indicates Output Compare unit (A/B).

## 19.9. Register Description

### 19.9.1. TC0 Control Register A

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** TCCR0A
**Offset:** 0x44
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x24

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | COM0A1 | COM0A0 | COM0B1 | COM0B0 | | | WGM01 | WGM00 |
| Access | R/W | R/W | R/W | R/W | | | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | | | 0 | 0 |

**Bits 7:6 – COM0An: Compare Output Mode for Channel A [n = 1:0]**
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. The table below shows the COM0A[1:0] bit functionality when the WGM0[2:0] bits are set to a normal or CTC mode (non- PWM).

**Table 19-3. Compare Output Mode, non-PWM**

| 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 . |

The table below shows the COM0A[1:0] bit functionality when the WGM0[1:0] bits are set to fast PWM mode.

**Table 19-4. Compare Output Mode, Fast PWM(1)**

| COM0A1 | COM0A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC0A disconnected. |
| 0 | 1 | WGM02 = 0: Normal Port Operation, OC0A Disconnected<br>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. Refer to Fast PWM Mode for details.

The table below shows the COM0A[1:0] bit functionality when the WGM0[2:0] bits are set to phase correct PWM mode.

**Table 19-5. Compare Output Mode, Phase Correct PWM Mode(1)**

| COM0A1 | COM0A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC0A disconnected. |
| 0 | 1 | WGM02 = 0: Normal Port Operation, OC0A Disconnected.<br>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. 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 TOP. Refer to Phase Correct PWM Mode for details.

**Bits 5:4 – COM0Bn: Compare Output Mode for Channel B [n = 1:0]**

These bits control the Output Compare pin (OC0B) behavior. If one or both of the COM0B[1:0] bits are set, the OC0B 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 OC0B pin must be set in order to enable the output driver.

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

**Table 19-6. Compare Output Mode, non-PWM**

| 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. |

The table below shows the COM0B[1:0] bit functionality when the WGM0[2:0] bits are set to fast PWM mode.

**Table 19-7. Compare Output Mode, Fast PWM(1)**

| COM0B1 | COM0B0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC0B disconnected. |
| 0 | 1 | Reserved |

| COM0B1 | COM0B0 | Description |
|---|---|---|
| 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 TOP. Refer to Fast PWM Mode for details.

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

**Table 19-8. Compare Output Mode, Phase Correct PWM Mode(1)**

| 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. Refer to Phase Correct PWM Mode for details.

**Bits 1:0 – WGM0n: Waveform Generation Mode [n = 1:0]**

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. 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).

**Table 19-9. Waveform Generation Mode Bit Description**

| Mode | WGM02 | WGM01 | WGM00 | Timer/Counter Mode of Operation | TOP | Update of OCR0x at | TOV Flag Set on(1)(2) |
|---|---|---|---|---|---|---|---|
| 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
2. BOTTOM = 0x00

### 19.9.2. TC0 Control Register B

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** TCCR0B
**Offset:** 0x45
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x25

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | FOC0A | FOC0B | | | WGM02 | CS0[2:0] | | |
| Access | R/W | R/W | | | R/W | R/W | R/W | R/W |
| Reset | 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.

To ensure 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. 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 is always read as zero.

**Bit 6 – FOC0B: Force Output Compare B**
The FOC0B bit is only active when the WGM bits specify a non-PWM mode.

To ensure 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. 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 is always read as zero.

**Bit 3 – WGM02: Waveform Generation Mode**
Refer to TCCR0A.

**Bits 2:0 – CS0[2:0]: Clock Select 0 [n = 0..2]**
The three Clock Select bits select the clock source to be used by the Timer/Counter.

**Table 19-10. Clock Select Bit Description**

| CA02 | CA01 | CS00 | 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 | clkI/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.

### 19.9.3. TC0 Interrupt Mask Register

**Name:** TIMSK0
**Offset:** 0x6E
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | OCIEB | OCIEA | TOIE |
| Access | | | | | | R/W | R/W | R/W |
| Reset | | | | | | 0 | 0 | 0 |

**Bit 2 – OCIEB: Timer/Counter0, Output Compare B Match 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 TIFR0.

**Bit 1 – OCIEA: Timer/Counter0, Output Compare A Match 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 TIFR0.

**Bit 0 – TOIE: 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 TIFR0.

### 19.9.4. General Timer/Counter Control Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** GTCCR
**Offset:** 0x43
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x23

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TSM | | | | | | PSRASY | PSRSYNC |
| Access | R/W | | | | | | R/W | R/W |
| Reset | 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 PSRASY and PSRSYNC bits is kept, hence keeping the corresponding prescaler reset signals asserted. This ensures that the corresponding Timer/Counters are halted and can be configured to the same value without the risk of one of them advancing during configuration. When the TSM bit is written to zero, the PSRASY and PSRSYNC bits are cleared by hardware, and the Timer/Counters start counting simultaneously.

**Bit 1 – PSRASY: Prescaler Reset Timer/Counter2**

When this bit is one, the Timer/Counter2 prescaler will be reset. This bit is normally cleared immediately by hardware. If the bit is written when Timer/Counter2 is operating in asynchronous mode, the bit will remain one until the prescaler has been reset. The bit will not be cleared by hardware if the TSM bit is set.

**Bit 0 – PSRSYNC: Prescaler Reset**

When this bit is one, Timer/Counter1 and Timer/Counter0 prescaler will be Reset. This bit is normally cleared immediately by hardware, except if the TSM bit is set. Note that Timer/Counter1 and Timer/Counter0 share the same prescaler and a reset of this prescaler will affect both timers.

### 19.9.5. TC0 Counter Value Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** TCNT0
**Offset:** 0x46
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x26

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TCNT0[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – TCNT0[7:0]: TC0 Counter Value**

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.

### 19.9.6. TC0 Output Compare Register A

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** OCR0A
**Offset:** 0x47
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x27

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OCR0A[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – OCR0A[7:0]: Output Compare 0 A**

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.

### 19.9.7. TC0 Output Compare Register B

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** OCR0B
**Offset:** 0x48
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x28

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OCR0B[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – OCR0B[7:0]: Output Compare 0 B**

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.

### 19.9.8. TC0 Interrupt Flag Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** TIFR0
**Offset:** 0x35
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x15

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | OCFB | OCFA | TOV |
| Access | | | | | | R/W | R/W | R/W |
| Reset | | | | | | 0 | 0 | 0 |

**Bit 2 – OCFB: Timer/Counter0, Output Compare B Match Flag**
The OCF0B bit is set when a Compare Match occurs between the Timer/Counter and the data in OCR0B – Output Compare Register0 B. 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 – OCFA: Timer/Counter0, Output Compare A Match Flag**
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 – TOV: 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 WGM02:0 bit setting. Refer to Table 19-9.

# 20. TC1 - 16-bit Timer/Counter1 with PWM

**Related Links**

Timer/Counter0 and Timer/Counter1 Prescalers on page 193

## 20.1. Overview

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

A block diagram of the 16-bit Timer/Counter is shown below. 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 Register Description. For the actual placement of I/O pins, refer to the *Pin Configurations* description.

**Related Links**

I/O-Ports on page 104

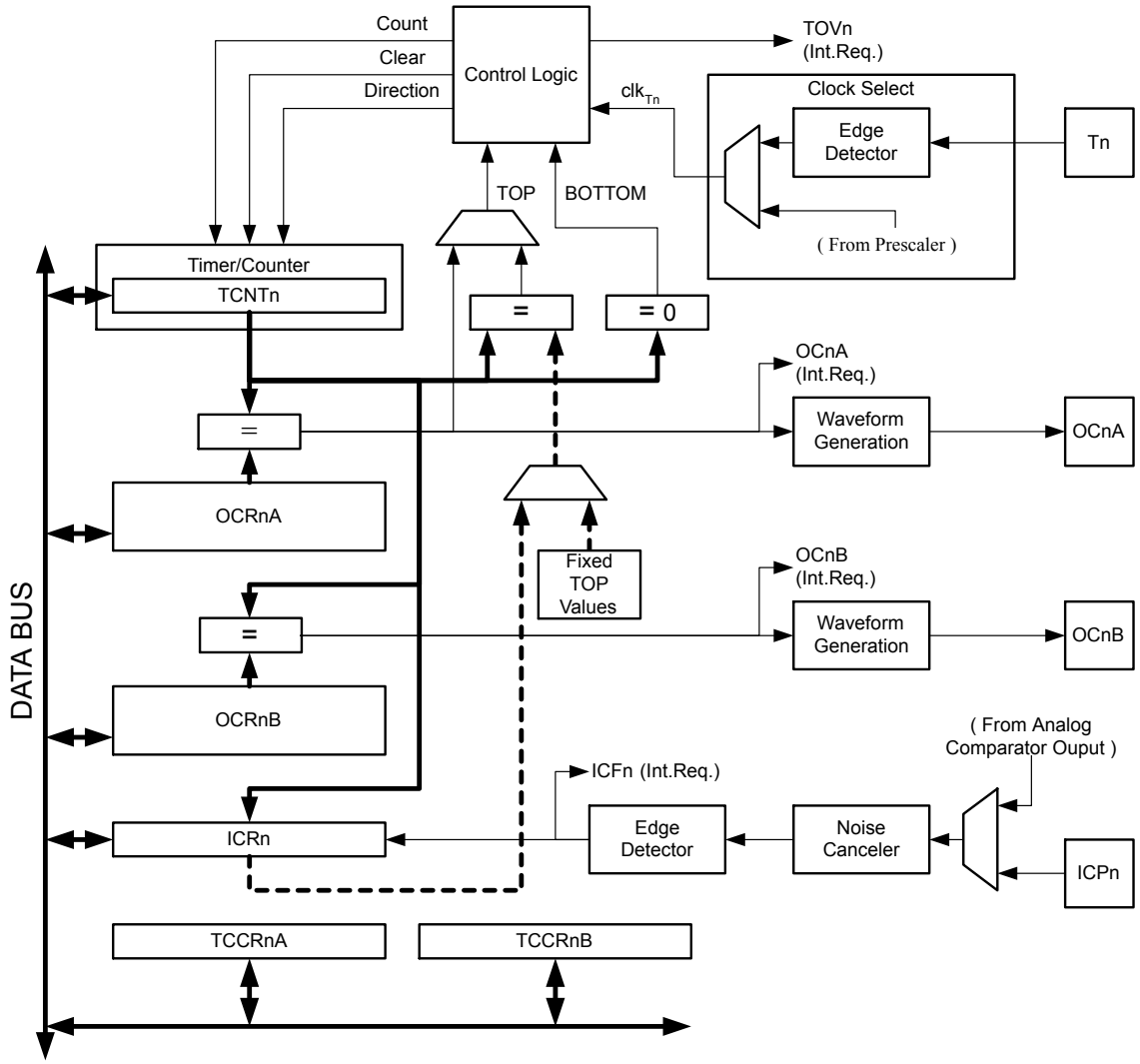
## 20.2. Features

- 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
- Independent interrupt Sources (TOV, OCFA, OCFB, and ICF)

## 20.3. Block Diagram

The Power Reduction TC1 bit in the Power Reduction Register (PRRPRR.PRTIM1) must be written to zero to enable the TC1 module.

**Figure 20-1. 16-bit Timer/Counter Block Diagram**




See the related links for actual pin placement.

## 20.4. Definitions

Many register and bit references in this section are written in general form:

- n=1 represents the Timer/Counter number
- x=A,B represents the Output Compare Unit A or B

However, when using the register or bit definitions in a program, the precise form must be used, i.e., TCNT1 for accessing Timer/Counter1 counter value.

The following definitions are used throughout the section:

**Table 20-1. Definitions**

| Constant | Description |
|---|---|
| BOTTOM | The counter reaches the BOTTOM when it becomes zero (0x00 for 8-bit counters, or 0x0000 for 16-bit counters). |
| MAX | The counter reaches its Maximum when it becomes 0xFF (decimal 255, for 8-bit counters) or 0xFFFF (decimal 65535, for 16-bit counters). |
| 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 MAX or the value stored in the OCR1A Register. The assignment is dependent on the mode of operation. |

## 20.5. Registers

The Timer/Counter (TCNT1), Output Compare Registers (OCRA/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.

The Timer/Counter Control Registers (TCCR1A/B/C) are 8-bit registers and have no CPU access restrictions. Interrupt requests (abbreviated to Int.Req. in the block diagram) signals are all visible in the Timer Interrupt Flag Register (TIFR1). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK1). TIFR1 and TIMSK1 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. 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. 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.

## 20.6. 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 accessed byte-wise, using two read or write operations. Each 16-bit timer has a single 8-bit TEMP 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 that is currently stored in TEMP and the low byte being 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 TEMP register in the same clock cycle as the low byte is read, and must be read subsequently.

**Note:** To perform a 16-bit write operation, 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.

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.

### 16-bit Access

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(1)**

```
   ...
; 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
   ...
```

The assembly code example returns the TCNT**1** value in the r17:r16 register pair.

**C Code Example(1)**

```
unsigned int i;
   ...
/* Set TCNT1 to 0x01FF */
TCNT1 = 0x1FF;
/* Read TCNT1 into i */
i = TCNT1;
   ...
```

**Note:**

1. The example code assumes that the part specific header file is included. For I/O Registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

### Atomic Read

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 perform an atomic read of the TCNT1 Register contents. The OCR1A/B or ICR1 Registers can be ready by using the same principle.

**Assembly Code Example(1)**

```
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
```

The assembly code example returns the TCNT1 value in the r17:r16 register pair.

**C Code Example(1)**

```
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:**

1. The example code assumes that the part specific header file is included. For I/O Registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

### Atomic Write

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(1)**

```
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
```

The assembly code example requires that the r17:r16 register pair contains the value to be written to TCNT1.

**C Code Example(1)**

```
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:**

1. The example code assumes that the part specific header file is included. For I/O Registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

**Related Links**

About Code Examples on page 25

### 20.6.1. 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, the high byte only needs to be written once. However, the same rule of atomic operation described previously also applies in this case.

## 20.7. 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 bits in the Timer/Counter control Register B (TCCR1B.CS[2:0]).

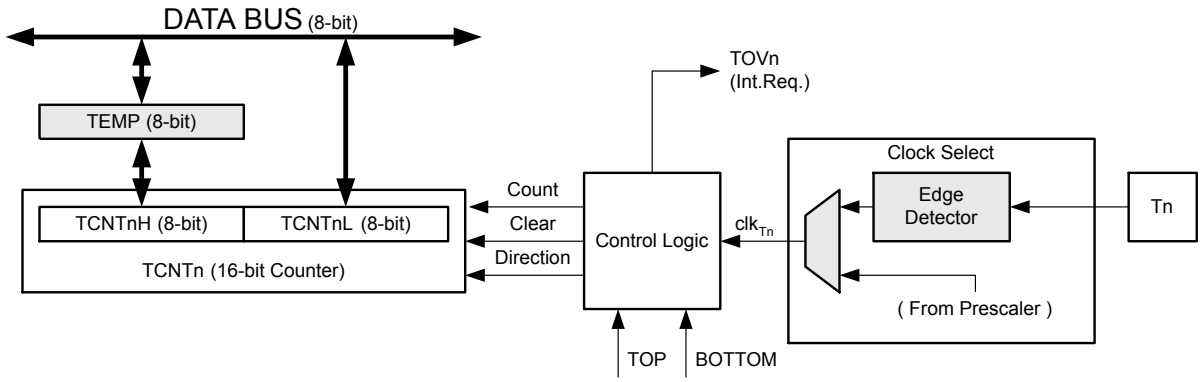
**Related Links**

Timer/Counter 0, 1 Prescalers on page 193

## 20.8. Counter Unit

The main part of the 16-bit Timer/Counter is the programmable 16-bit bi-directional counter unit, as shown in the block diagram:

**Figure 20-2. Counter Unit Block Diagram**




**Note:** The “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).

**Table 20-2. Signal description (internal signals)**

| Signal Name | Description |
|---|---|
| Count | Increment or decrement TCNT1 by 1. |
| Direction | Select between increment and decrement. |
| Clear | Clear TCNT1 (set all bits to zero). |
| $clk_{T1}$ | Timer/Counter 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 accessed indirectly 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.

**Note:** That there are special cases when writing to the TCNT1 Register while the counter is counting will give unpredictable results. These special cases are described in the sections where they are of importance.

Depending on the selected mode of operation, the counter is cleared, incremented, or decremented at each timer clock ($clk_{T1}$). The clock $clk_{T1}$ can be generated from an external or internal clock source, as selected by the Clock Select bits in the Timer/Counter1 Control Register B (TCCR1B.CS[2:0]). When no clock source is selected (CS[2:0]=0x0) 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 (i.e., has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the Waveform Generation mode bits in the Timer/Counter Control Registers A and B (TCCR1B.WGM1[3:2] and TCCR1A.WGM1[1:0]). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare outputs OC0x. For more details about advanced counting sequences and waveform generation, see Modes of Operation.

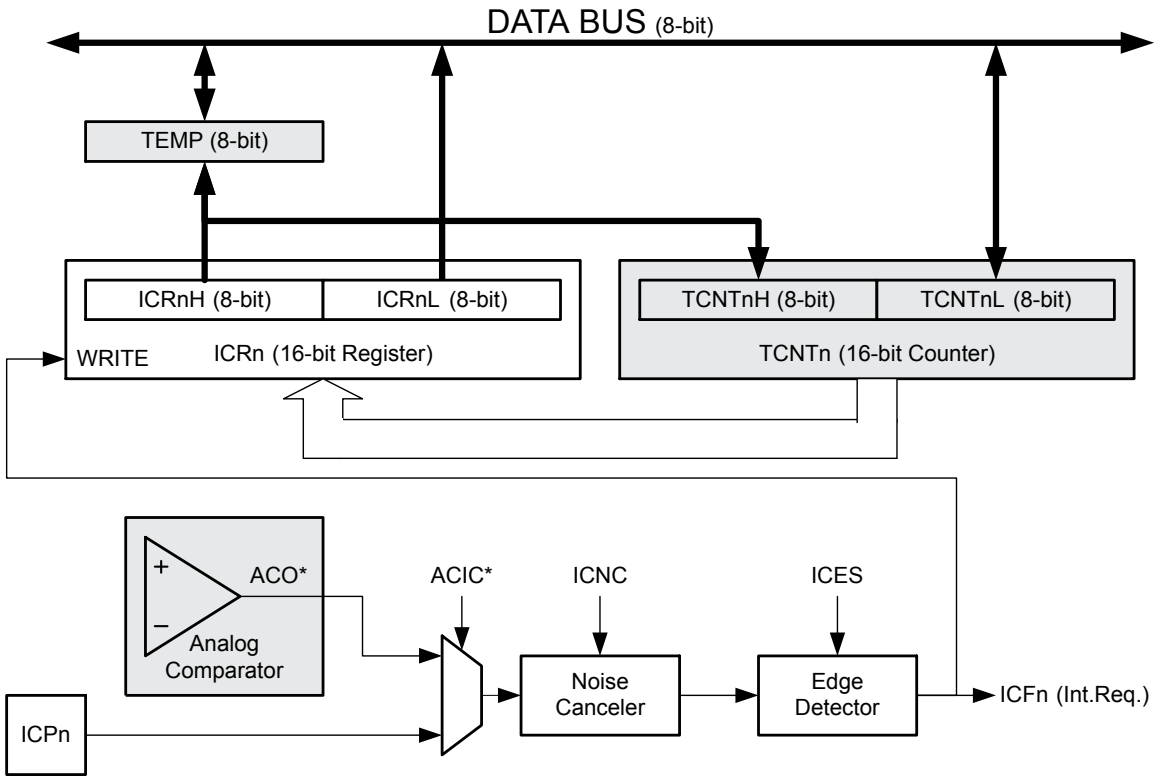
The Timer/Counter Overflow Flag in the TC1 Interrupt Flag Register (TIFR1.TOV) is set according to the mode of operation selected by the WGM1[3:0] bits. TOV can be used for generating a CPU interrupt.

## 20.9. Input Capture Unit

The Timer/Counter1 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.

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

**Figure 20-3. Input Capture Unit Block Diagram for TC1**




**Note:** The “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).

When a change of the logic level (an event) occurs on the Input Capture pin (ICP1), or alternatively on the Analog Comparator output (ACO), and this change confirms to the setting of the edge detector, a capture will be triggered: the 16-bit value of the counter (TCNT1) is written to the Input Capture Register (ICR1). The Input Capture Flag (ICF) is set at the same system clock cycle as the TCNT1 value is copied into the ICR1 Register. If enabled (TIMSK1.ICIE=1), the Input Capture Flag generates an Input Capture interrupt. The ICF1 Flag is automatically cleared when the interrupt is executed. Alternatively the ICF Flag can be cleared by software by writing '1' 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 form ICR1L, 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 bits (WGM1[3:0]) 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.

See also Accessing 16-bit Registers.

### 20.9.1. 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. 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. 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.

**Related Links**

Timer/Counter 0, 1 Prescalers on page 193

### 20.9.2. Noise Canceler

The noise canceler improves noise immunity by using a simple digital filtering scheme. The noise canceler input is monitored over four samples, and all four must be equal for changing the output that in turn is used by the edge detector.

The noise canceler is enabled by setting the Input Capture Noise Canceler bit in the Timer/Counter Control Register B (TCCR1B.ICNC). When enabled, the noise canceler introduces an additional delay of four system clock cycles between a change applied to the input and the update of the ICR1 Register. The noise canceler uses the system clock and is therefore not affected by the prescaler.

### 20.9.3. 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 (ICF) must be cleared by software (writing a logical one to the I/O bit location). For measuring frequency only, the clearing of the ICF Flag is not required (if an interrupt handler is used).

## 20.10. Output Compare Units

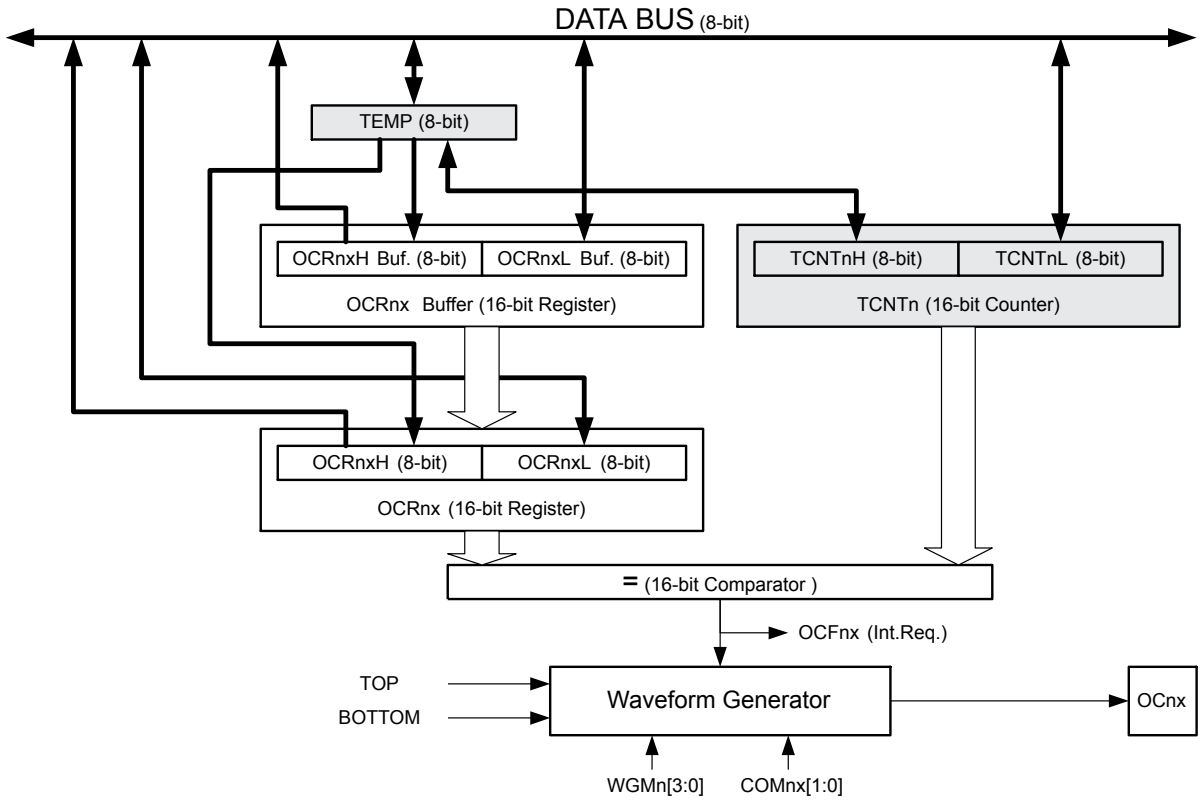
The 16-bit comparator continuously compares TCNT1 with the Output Compare Register (OCR1x). If TCNT equals OCR1x the comparator signals a match. A match will set the Output Compare Flag (TIFR1.OCFx) at the next timer clock cycle. If enabled (TIMSK1.OCIEx = 1), the Output Compare Flag generates an Output Compare interrupt. The OCFx Flag is automatically cleared when the interrupt is executed. Alternatively the OCFx 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, see Modes of Operation.

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.

Below is a block diagram of the Output Compare unit. The elements of the block diagram that are not directly a part of the Output Compare unit are gray shaded.

**Figure 20-4. Output Compare Unit, Block Diagram**




**Note:** The “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 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.

When double buffering is enabled, the CPU has access to the OCR1x Buffer Register. When 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 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.

### 20.10.1. 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 (TCCR1C.FOC1x) 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 TCCR1C.COM1x[1:0] bits settings define whether the OC1x pin is set, cleared or toggled).

### 20.10.2. 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.

### 20.10.3. 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.

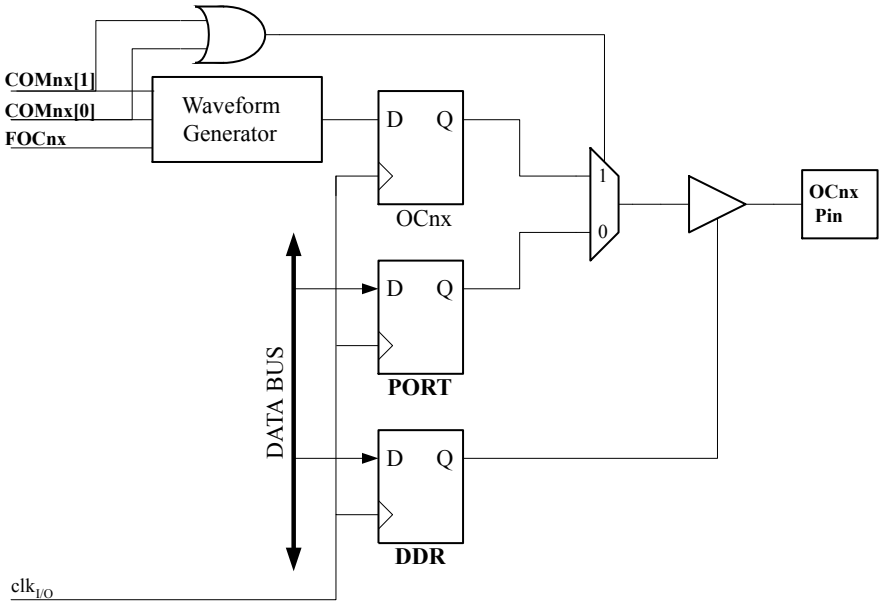
The setup of the OC1x should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC1x value is to use the Force Output Compare (FOC1x) strobe bits in Normal mode. The OC1x Register keeps its value even when changing between Waveform Generation modes.

Be aware that the TCCR1A.COM1x[1:0] bits are not double buffered together with the compare value. Changing the TCCR1A.COM1x[1:0] will take effect immediately.

## 20.11. Compare Match Output Unit

The Compare Output mode (TCCR1A.COM1x[1:0]) bits have two functions. The Waveform Generator uses the TCCR1A.COM1x[1:0] bits for defining the Output Compare (OC1x) state at the next compare match. Secondly the TCCR1A.COM1x[1:0] bits control the OC1x pin output source. The figure below shows a simplified schematic of the logic affected by the TCCR1A.COM1x[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 TCCR1A.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”.

**Figure 20-5. Compare Match Output Unit, Schematic**




**Note:** The “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 general I/O port function is overridden by the Output Compare (OC1x) from the Waveform Generator if either of the TCCR1A.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.

The design of the Output Compare pin logic allows initialization of the OC1x state before the output is enabled. Note that some TCCR1A.COM1x[1:0] bit settings are reserved for certain modes of operation.

The TCCR1A.COM1x[1:0] bits have no effect on the Input Capture unit.

### 20.11.1. Compare Output Mode and Waveform Generation

The Waveform Generator uses the TCCR1A.COM1x[1:0] bits differently in normal, CTC, and PWM modes. For all modes, setting the TCCR1A.COM1x[1:0] = 0 tells the Waveform Generator that no action on the OC1x Register is to be performed on the next compare match. Refer also to the descriptions of the output modes.

A change of the TCCR1A.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 TCCR1C.FOC1x strobe bits.

## 20.12. 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 (TCCR1A.COM1x[1:0]) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The TCCR1A.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 TCCR1A.COM1x[1:0] bits control whether the output should be set, cleared, or toggle at a compare match.

**Related Links**

Timer/Counter Timing Diagrams on page 175
Compare Match Output Unit on page 166

### 20.12.1. Normal Mode

The simplest mode of operation is the Normal mode (TCCR1A.WGM1[3:0]=0x0). 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 BOTTOM=0x0000. In normal operation the Timer/Counter Overflow Flag (TIFR1.TOV) will be set in the same timer clock cycle as the TCNT1 becomes zero. In this case, the TOV Flag in behaves like a 17th bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV 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.

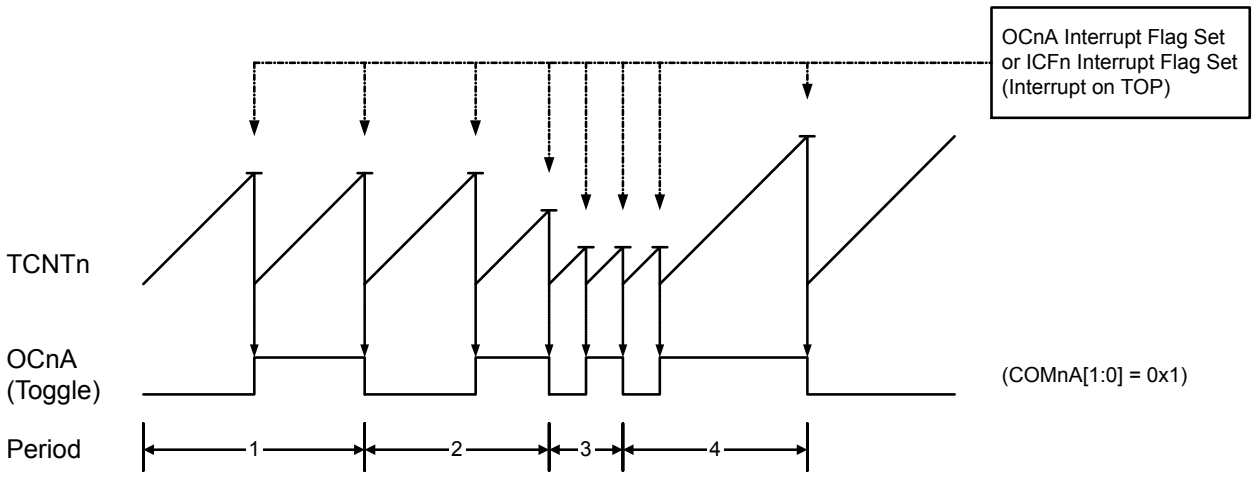
The Output Compare units can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

### 20.12.2. Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC modes (mode 4 or 12, WGM1[3:0]=0x4 or 0xC), the OCR1A or ICR1 registers are used to manipulate the counter resolution: the counter is cleared to ZERO when the counter value (TCNT1) matches either the OCR1A (if WGM1[3:0]=0x4) or the ICR1 (WGM1[3:0]=0xC). 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 below. The counter value (TCNT1) increases until a compare match occurs with either OCR1A or ICR1, and then TCNT1 is cleared.

**Figure 20-6. CTC Mode, Timing Diagram**




**Note:** The “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).

An interrupt can be generated at each time the counter value reaches the TOP value by either using the OCF1A or ICF1 Flag, depending on the actual CTC mode. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value.

**Note:** Changing TOP to a value close to BOTTOM while the counter is running must be done with care, since the CTC mode does not provide double buffering. If the new value written to OCR1A is lower than the current value of TCNT1, the counter will miss the compare match. The counter will then count to its maximum value (0xFF for a 8-bit counter, 0xFFFF for a 16-bit counter) and wrap around starting at 0x00 before the compare match will 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]=0xF), 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]=0x1). 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 $f_{OC1A} = f_{clk_I/O}/2$ when OCR1A is set to ZERO (0x0000). The waveform frequency is defined by the following equation:

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

**Note:**

- The “n” indicates the device number (n = 1 for Timer/Counter 1), and the “x” indicates Output Compare unit (A/B).
- *N* represents the prescaler factor (1, 8, 64, 256, or 1024).

As for the Normal mode of operation, the Timer Counter TOV Flag is set in the same timer clock cycle that the counter counts from MAX to 0x0000.

### 20.12.3. Fast PWM Mode

The Fast Pulse Width Modulation or Fast PWM modes (modes 5, 6, 7, 14,and 15, WGM1[3:0]= 0x5, 0x6, 0x7, 0xE, 0xF) provide 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.

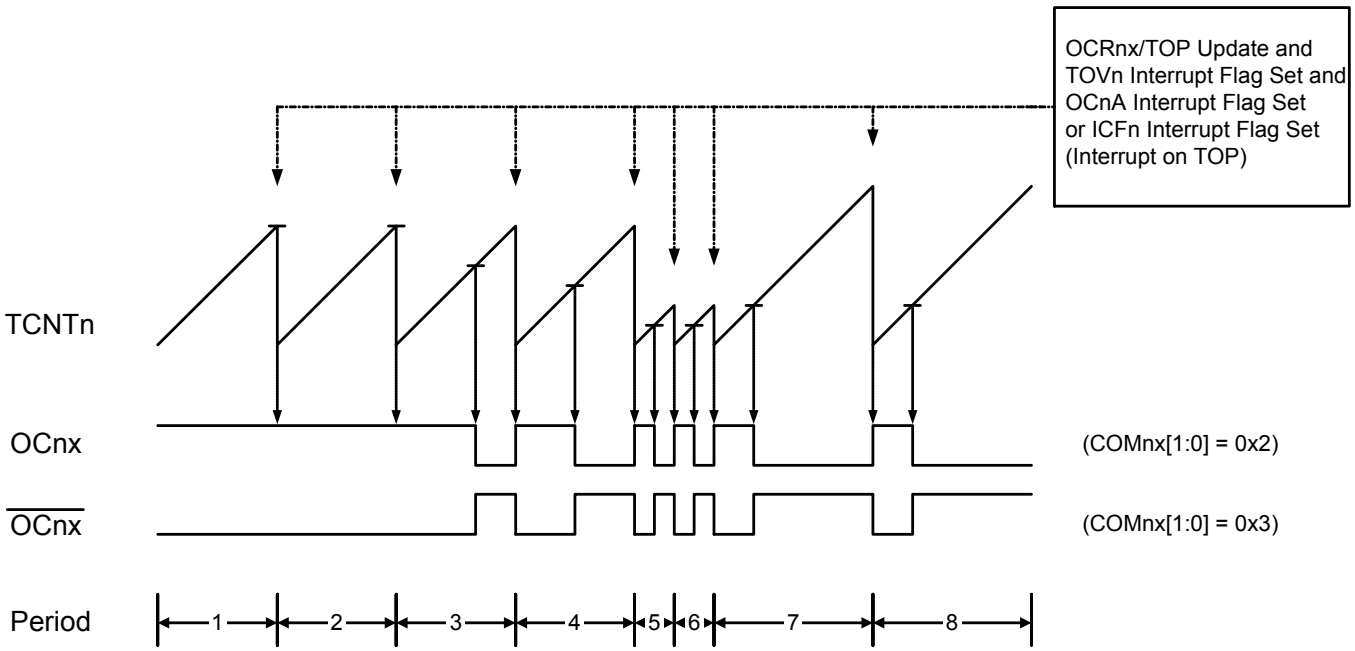
The PWM resolution for Fast PWM can be fixed to 8-, 9-, or 10-bit, or defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or OCR1A register set to 0x0003), and the maximum resolution is 16-bit (ICR1 or OCR1A registers set to MAX). The PWM resolution in bits can be calculated by using the following equation:

$$R_{\mathrm{FPWM}} = \frac{\log(\mathrm{TOP}+1)}{\log(2)}$$

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] = 0x5, 0x6, or 0x7), the value in ICR1 (WGM1[3:0]=0xE), or the value in OCR1A (WGM1[3:0]=0xF). The counter is then cleared at the following timer clock cycle. The timing diagram for the Fast PWM mode using OCR1A or ICR1 to define TOP is shown below. 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 lines on the TCNT1 slopes mark compare matches between OCR1x and TCNT1. The OC1x Interrupt Flag will be set when a compare match occurs.

**Figure 20-7. Fast PWM Mode, Timing Diagram**




**Note:** The “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 Timer/Counter Overflow Flag (TOV1) is set each time the counter reaches TOP. In addition, when either OCR1A or ICR1 is used for defining the TOP value, the OC1A or ICF1 Flag is set at the same timer clock cycle TOV1 is set. 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. As result, 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. Writing the COM1x[1:0] bits to 0x2 will produce an inverted PWM and a non-inverted PWM output can be generated by writing the COM1x[1:0] to 0x3. 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:

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

**Note:**

- The “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).
- *N* represents the prescale divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR1x registers 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 which is controlled by COM1x[1:0]).

A frequency waveform output with 50% duty cycle can be achieved in Fast PWM mode by selecting OC1A to toggle its logical level on each compare match (COM1A[1:0]=0x1). This applies only if OCR1A is used to define the TOP value (WGM1[3:0]=0xF). The waveform generated will have a maximum frequency of $f_{OC1A} = f_{clk_I/O}/2$ 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.

### 20.12.4. Phase Correct PWM Mode

The Phase Correct Pulse Width Modulation or Phase Correct PWM modes (WGM1[3:0]= 0x1, 0x2, 0x3, 0xA, and 0xB) provide 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 up-counting, 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.

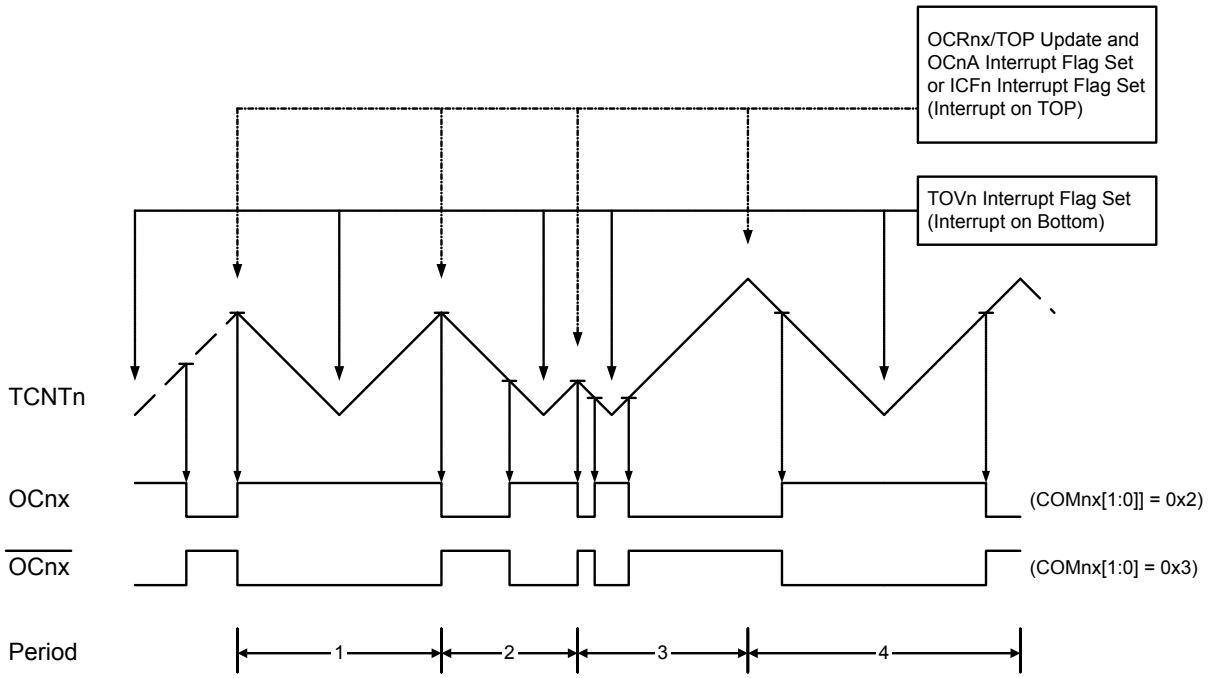
The PWM resolution for the Phase Correct PWM mode can be fixed to 8-, 9-, or 10-bit, or defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or OCR1A set to 0x0003), and the maximum resolution is 16-bit (ICR1 or OCR1A set to MAX). The PWM resolution in bits can be calculated by using the following equation:

$$R_{\mathrm{PCPWM}} = \frac{\log(\mathrm{TOP}+1)}{\log(2)}$$

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]= 0x1, 0x2, or 0x3), the value in ICR1 (WGM1[3:0]=0xA), or the value in OCR1A (WGM1[3:0]=0xB). 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 below, using OCR1A or ICR1 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 lines on the TCNT1 slopes mark compare matches between OCR1x and TCNT1. The OC1x Interrupt Flag will be set when a compare match occurs.

**Figure 20-8. Phase Correct PWM Mode, Timing Diagram**




**Note:** The “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 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 is written. As illustrated by the third period in the timing diagram, 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. Writing COM1x[1:0] bits to 0x2 will produce a non-inverted PWM. An inverted PWM output can be generated by writing the COM1x[1:0] to 0x3. 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:

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

*N* represents the prescale 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. If OCR1A is used to define the TOP value (WGM1[3:0]=0xB) and COM1A[1:0]=0x1, the OC1A output will toggle with a 50% duty cycle.

### 20.12.5. Phase and Frequency Correct PWM Mode

The phase and frequency correct Pulse Width Modulation, or phase and frequency correct PWM mode (WGM1[3:0] = 0x8 or 0x9) 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 up-counting, and set on the compare match while down-counting. 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.

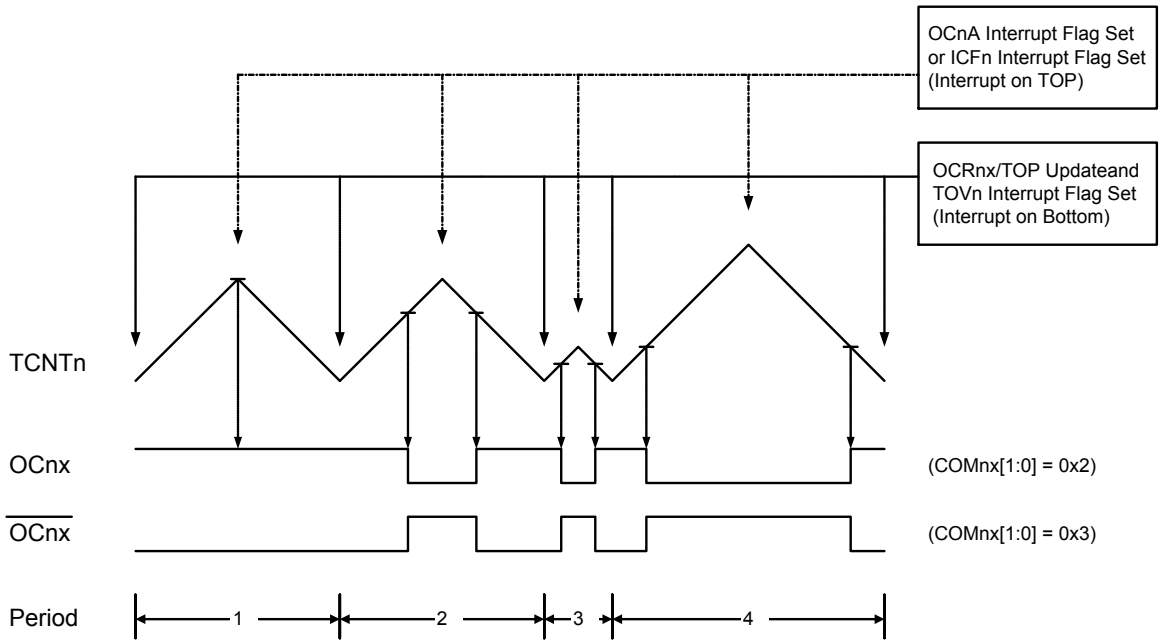
The main difference between the phase correct, and the phase and frequency correct PWM mode is the time the OCR1x Register is updated by the OCR1x Buffer Register, (see Figure 20-8 and the Timing Diagram below).

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:

$$R_{\mathrm{PFCPWM}} = \frac{\log(\mathrm{TOP}+1)}{\log(2)}$$

In phase and frequency correct PWM mode the counter is incremented until the counter value matches either the value in ICR1 (WGM1[3:0]=0x8), or the value in OCR1A (WGM1[3:0]=0x9). 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 below. 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 20-9. Phase and Frequency Correct PWM Mode, Timing Diagram**




**Note:** The “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 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 shown in the timing diagram above, 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 0x2 will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x[1:0] to 0x3 (See description of TCCRA.COM1x). 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:

$$f_{\mathrm{OCnxPFCPWM}} = \frac{f_{\mathrm{clk_I/O}}}{2 \cdot N \cdot \mathrm{TOP}}$$

**Note:**

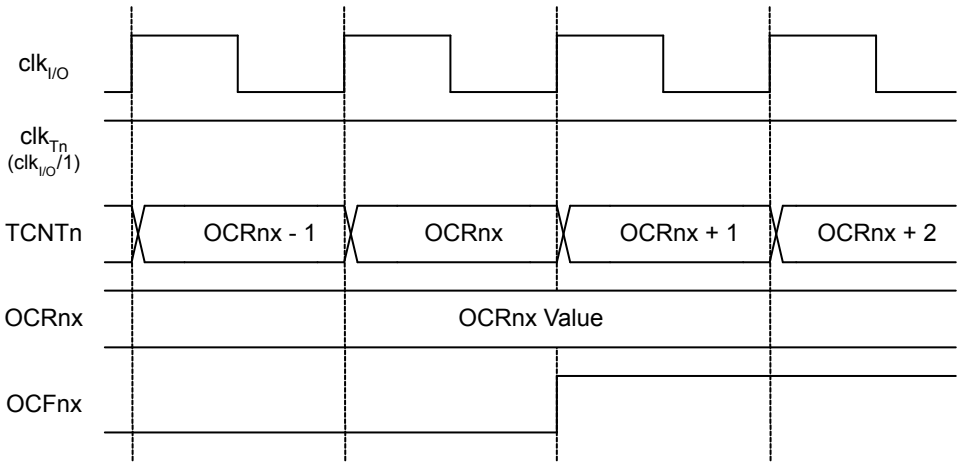
- The “n” in the register and bit names indicates the device number (n = 1 for Timer/Counter 1), and the “x” indicates Output Compare unit (A/B).
- *N* represents the prescale divider (1, 8, 64, 256, or 1024).

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. If OCR1A is used to define the TOP value (WGM1[3:0]=0x9) and COM1A[1:0]=0x1, the OC1A output will toggle with a 50% duty cycle.

## 20.13. 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). The first figure shows a timing diagram for the setting of OCF1x.

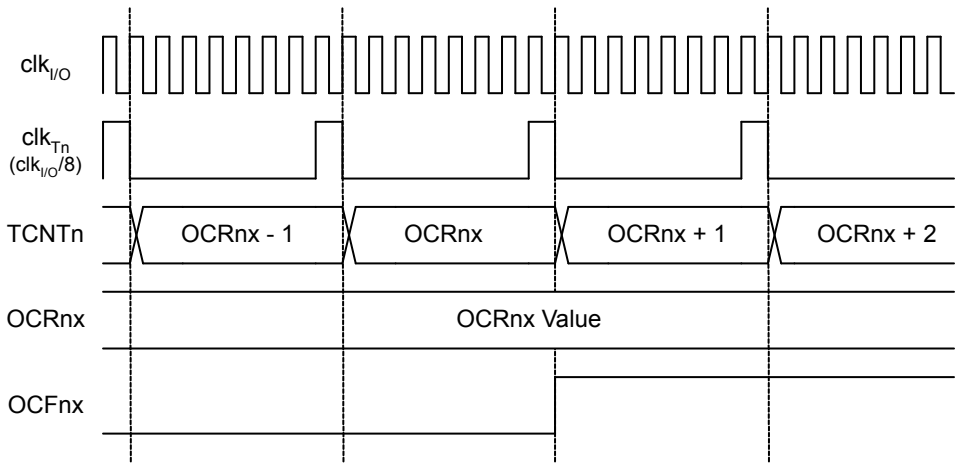
**Figure 20-10. Timer/Counter Timing Diagram, Setting of OCF1x, no Prescaling**




**Note:** The “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 next figure shows the same timing data, but with the prescaler enabled.

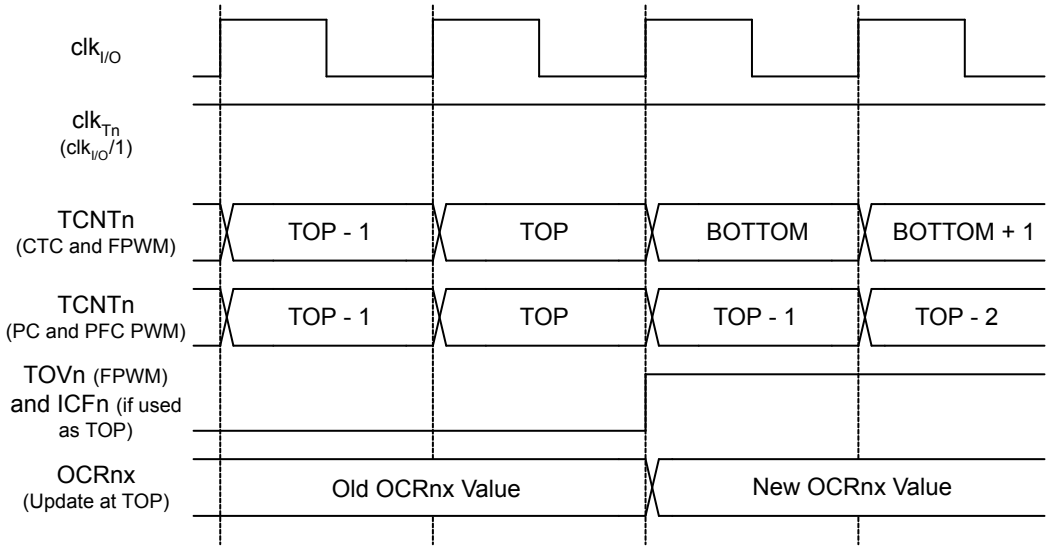
**Figure 20-11. Timer/Counter Timing Diagram, Setting of OCF1x, with Prescaler ($f_{clk_I/O}/8$)**



**Note:** The “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 next figure 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.

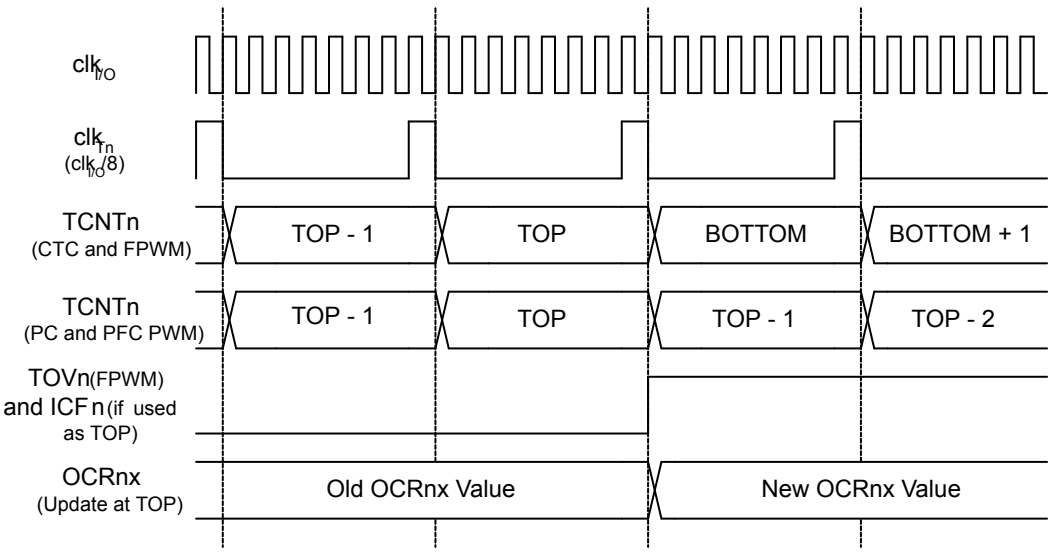
**Figure 20-12. Timer/Counter Timing Diagram, no Prescaling.**




**Note:** The “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 next figure shows the same timing data, but with the prescaler enabled.

**Figure 20-13. Timer/Counter Timing Diagram, with Prescaler ($f_{clk_I/O}$/8)**




**Note:** The “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).

## 20.14. Register Description

### 20.14.1. TC1 Control Register A

**Name:** TCCR1A
**Offset:** 0x80
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | COM1 | COM1 | COM1 | COM1 | | | WGM11 | WGM10 |
| Access | R/W | R/W | R/W | R/W | | | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | | | 0 | 0 |

**Bits 4, 5, 6, 7 – COM1, COM1, COM1, COM1: Compare Output Mode for Channel**
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. The table below shows the COM1x[1:0] bit functionality when the WGM1[3:0] bits are set to a Normal or a CTC mode (non-PWM).

**Table 20-3. 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). |

The table below shows the COM1x[1:0] bit functionality when the WGM1[3:0] bits are set to the fast PWM mode.

**Table 20-4. Compare Output Mode, Fast PWM**

| COM1A1/ COM1B1 | COM1A0/ COM1B0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC1A/OC1B disconnected. |
| 0 | 1 | WGM1[3:0] = 14 or 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected. |

| COM1A1/ COM1B1 | COM1A0/ COM1B0 | Description |
|---|---|---|
| 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:**

1. 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. Refer to Fast PWM Mode for details.

The table below shows the COM1x1:0 bit functionality when the WGM1[3:0] bits are set to the phase correct or the phase and frequency correct, PWM mode.

**Table 20-5. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM**

| COM1A1/ COM1B1 | COM1A0/ COM1B0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC1A/OC1B disconnected. |
| 0 | 1 | WGM1[3:0] = 9 or 11: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected. |
| 1 | 0 | Clear OC1A/OC1B on Compare Match when up-counting. Set OC1A/OC1B on Compare Match when down-counting. |
| 1 | 1 | Set OC1A/OC1B on Compare Match when up-counting. Clear OC1A/OC1B on Compare Match when down-counting. |

**Note:**

1. A special case occurs when OCR1A/OCR1B equals TOP and COM1A1/COM1B1 is set. Refer to Phase Correct PWM Mode for details.

**Bits 0, 1 – WGM10, WGM11: 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. 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. (See Modes of Operation).

**Table 20-6. Waveform Generation Mode Bit Description**

| Mode | WGM13 | WGM12 (CTC1)(1) | WGM11 (PWM11)(1) | WGM10 (PWM10)(1) | Timer/ Counter Mode of Operation | TOP | Update of OCR1x at | TOV1 Flag Set on |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | Normal | 0xFFFF | Immediate | MAX |
| 1 | 0 | 0 | 0 | 1 | PWM, Phase Correct, 8-bit | 0x00FF | TOP | BOTTOM |
| 2 | 0 | 0 | 1 | 0 | PWM, Phase Correct, 9-bit | 0x01FF | TOP | BOTTOM |

| Mode | WGM13 | WGM12 (CTC1)[1] | WGM11 (PWM11)[1] | WGM10 (PWM10)[1] | Timer/ Counter Mode of Operation | TOP | Update of OCR1x at | TOV1 Flag Set on |
|---|---|---|---|---|---|---|---|---|
| 3 | 0 | 0 | 1 | 1 | PWM, Phase Correct, 10-bit | 0x03FF | TOP | BOTTOM |
| 4 | 0 | 1 | 0 | 0 | CTC | OCR1A | Immediate | MAX |
| 5 | 0 | 1 | 0 | 1 | Fast PWM, 8-bit | 0x00FF | BOTTOM | TOP |
| 6 | 0 | 1 | 1 | 0 | Fast PWM, 9-bit | 0x01FF | BOTTOM | TOP |
| 7 | 0 | 1 | 1 | 1 | Fast PWM, 10-bit | 0x03FF | BOTTOM | TOP |
| 8 | 1 | 0 | 0 | 0 | PWM, Phase and Frequency Correct | ICR1 | BOTTOM | BOTTOM |
| 9 | 1 | 0 | 0 | 1 | PWM, Phase and Frequency Correct | OCR1A | BOTTOM | BOTTOM |
| 10 | 1 | 0 | 1 | 0 | PWM, Phase Correct | ICR1 | TOP | BOTTOM |
| 11 | 1 | 0 | 1 | 1 | PWM, Phase Correct | OCR1A | TOP | BOTTOM |
| 12 | 1 | 1 | 0 | 0 | CTC | ICR1 | Immediate | MAX |
| 13 | 1 | 1 | 0 | 1 | Reserved | - | - | - |
| 14 | 1 | 1 | 1 | 0 | Fast PWM | ICR1 | BOTTOM | TOP |
| 15 | 1 | 1 | 1 | 1 | Fast PWM | OCR1A | BOTTOM | TOP |

**Note:**

1. The CTC1 and PWM1[1:0] bit definition names are obsolete. Use the WGM1[3:0] definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

### 20.14.2. TC1 Control Register B

**Name:** TCCR1B
**Offset:** 0x81
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ICNC1 | ICES1 | | WGM13 | WGM12 | CS12 | CS11 | CS10 |
| Access | R/W | R/W | | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | | 0 | 0 | 0 | 0 | 0 |

**Bit 7 – ICNC1: Input Capture Noise Canceler**
Writing this bit to '1' 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 '1', 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.

**Bits 3, 4 – WGM12, WGM13: Waveform Generation Mode**
Refer to TCCR1A.

**Bits 0, 1, 2 – CS10, CS11, CS12: Clock Select 1 [n = 0..2]**
The three Clock Select bits select the clock source to be used by the Timer/Counter. Refer to Figure 20-10 and Figure 20-11.

**Table 20-7. Clock Select Bit Description**

| CS12 | CS11 | CS10 | Description |
|---|---|---|---|
| 0 | 0 | 0 | No clock source (Timer/Counter stopped). |
| 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 | clkI/O/256 (From prescaler) |
| 1 | 0 | 1 | $clk_{I/O}$/1024 (From prescaler) |

| CS12 | CS11 | CS10 | Description |
|---|---|---|---|
| 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. |

### 20.14.3. TC1 Control Register C

**Name:** TCCR1C
**Offset:** 0x82
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | FOC1A | FOC1B | | | | | | |
| Access | R/W | R/W | | | | | | |
| Reset | 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. 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.

### 20.14.4. TC1 Counter Value Low byte

**Name:** TCNT1L
**Offset:** 0x84
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TCNT1L[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – TCNT1L[7:0]: Timer/Counter 1 Counter Value Low byte**
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. Refer to Accessing 16-bit Registers for details.

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.

### 20.14.5. TC1 Counter High byte

**Name:** TCNT1H
**Offset:** 0x85
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TCNT1H[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – TCNT1H[7:0]: Timer/Counter 1 High byte**
Refer to TCNT1L.

### 20.14.6. Input Capture Register 1 Low byte

**Name:** ICR1L
**Offset:** 0x86
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ICR1L[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – ICR1L[7:0]: Input Capture 1 Low byte**
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. Refer to Accessing 16-bit Registers for details.

### 20.14.7. Input Capture Register 1 High byte

**Name:** ICR1H
**Offset:** 0x87
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ICR1H[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – ICR1H[7:0]: Input Capture 1 High byte**
Refer to ICR1L.

### 20.14.8. Output Compare Register 1 A Low byte

**Name:** OCR1AL
**Offset:** 0x88
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OCR1AL[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – OCR1AL[7:0]: Output Compare 1 A Low byte**
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. Refer to Accessing 16-bit Registers for details.

### 20.14.9. Output Compare Register 1 A High byte

**Name:** OCR1AH
**Offset:** 0x89
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OCR1AH[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – OCR1AH[7:0]: Output Compare 1 A High byte**
Refer to OCR1AL.

### 20.14.10. Output Compare Register 1 B Low byte

**Name:** OCR1BL
**Offset:** 0x8A
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OCR1BL[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – OCR1BL[7:0]: Output Compare 1 B Low byte**
Refer to OCR1AL.

### 20.14.11. Output Compare Register 1 B High byte

**Name:** OCR1BH
**Offset:** 0x8B
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OCR1BH[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – OCR1BH[7:0]: Output Compare 1 B High byte**
Refer to OCR1AL.

### 20.14.12. Timer/Counter 1 Interrupt Mask Register

**Name:** TIMSK1
**Offset:** 0x6F
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | ICIE | | | OCIEB | OCIEA | TOIE |
| Access | | | R/W | | | R/W | R/W | R/W |
| Reset | | | 0 | | | 0 | 0 | 0 |

**Bit 5 – ICIE: Input Capture Interrupt Enable**
When this bit is written to '1', and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Input Capture interrupt is enabled. The corresponding Interrupt Vector is executed when the ICF Flag, located in TIFR1, is set.

**Bit 2 – OCIEB: Output Compare B Match Interrupt Enable**
When this bit is written to '1', and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter Output Compare B Match interrupt is enabled. The corresponding Interrupt Vector is executed when the OCFB Flag, located in TIFR1, is set.

**Bit 1 – OCIEA: Output Compare A Match Interrupt Enable**
When this bit is written to '1', and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter Output Compare A Match interrupt is enabled. The corresponding Interrupt Vector is executed when the OCFA Flag, located in TIFR1, is set.

**Bit 0 – TOIE: Overflow Interrupt Enable**
When this bit is written to '1', and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter 1 Overflow interrupt is enabled. The corresponding Interrupt Vector is executed when the TOV Flag, located in TIFR1, is set.

### 20.14.13. TC1 Interrupt Flag Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** TIFR1
**Offset:** 0x36
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x16

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | ICF | | | OCFB | OCFA | TOV |
| Access | | | R/W | | | R/W | R/W | R/W |
| Reset | | | 0 | | | 0 | 0 | 0 |

**Bit 5 – ICF: 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 ICF Flag is set when the counter reaches the TOP value.

ICF is automatically cleared when the Input Capture Interrupt Vector is executed. Alternatively, ICF can be cleared by writing a logic one to its bit location.

**Bit 2 – OCFB: 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 (FOCB) strobe will not set the OCF1B Flag.

OCFB 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 – OCFA: 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 (FOCA) strobe will not set the OCF1A Flag.

OCFA 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 – TOV: 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. Refer to the Waveform Generation Mode bit description for the TOV Flag behavior when using another WGM1[3:0] bit setting.

TOV1 is automatically cleared when the Timer/Counter 1 Overflow Interrupt Vector is executed. Alternatively, TOV1 can be cleared by writing a logic one to its bit location.

# 21. Timer/Counter 0, 1 Prescalers

The 8-bit Timer/Counter0 (TC0) , 16-bit Timer/Counters 1 (TC1) share the same prescaler module, but the Timer/Counters can have different prescaler settings. The following description applies to: TC0 , TC1 .

**Related Links**

8-bit Timer/Counter0 with PWM on page 132
16-bit Timer/Counter1 with PWM on page 156

## 21.1. Internal Clock Source

The Timer/Counter can be clocked directly by the system clock (by setting the CSn[2:0]=0x1). 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 $f_{CLK_I/O}/1024$.

## 21.2. Prescaler Reset

The prescaler is free running, i.e., operates independently of the Clock Select logic of the Timer/Counter, and it is shared by Timer/Counter1 and Timer/Counter0. 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 (0x6 > CSn[2:0] > 0x1). 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).

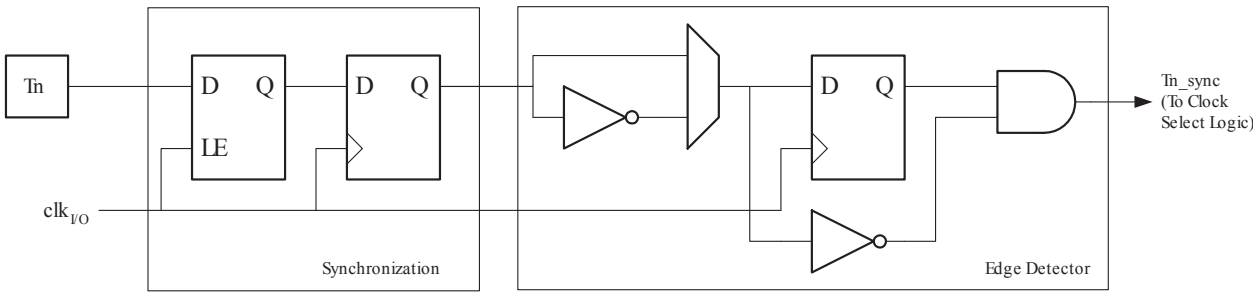
It is possible to use the prescaler reset for synchronizing the Timer/Counter to program execution. However, care must be taken if the other Timer/Counter that shares the same prescaler also uses prescaling. A prescaler reset will affect the prescaler period for all Timer/Counters it is connected to.

## 21.3. External Clock Source

An external clock source applied to the T1/T0 pin can be used as Timer/Counter clock ($clk_{T1}$/$clk_{T0}$). The T1/T0 pin is sampled once every system clock cycle by the pin synchronization logic. The synchronized (sampled) signal is then passed through the edge detector. See also the block diagram of the T1/T0 synchronization and edge detector logic below. 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 $clk_{T1}$/$clk_{T0}$ pulse for each positive (CSn[2:0]=0x7) or negative (CSn[2:0]=0x6) edge it detects.

**Figure 21-1. T1/T0 Pin Sampling**



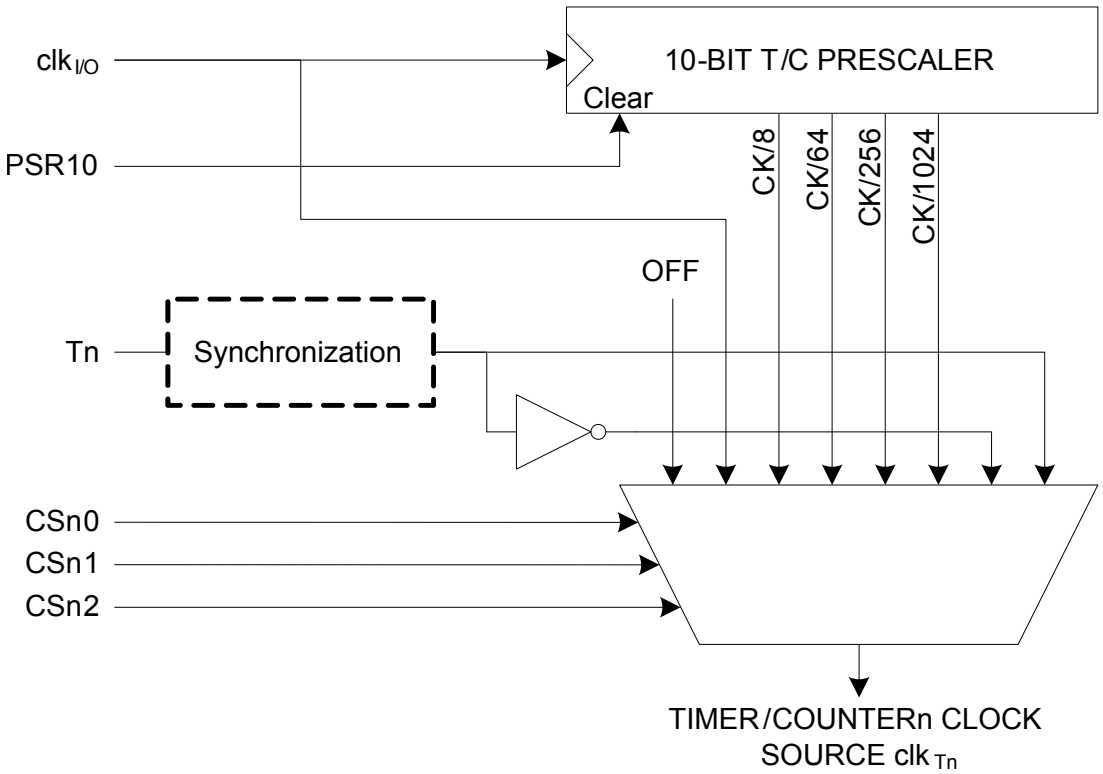
The synchronization and edge detector logic introduces a delay of 2.5 to 3.5 system clock cycles from an edge has been applied to the T1/T0 pin to the counter is updated.

Enabling and disabling of the clock input must be done when T1/T0 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 ($f_{Tn} < f_{clk_I/O}/2$) given a 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 the tolerances of the oscillator source (crystal, resonator, and capacitors), 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 21-2. Prescaler for Timer/Counter0 and Timer/Counter1(1)**




**Note:** 1. The synchronization logic on the input pins (T1/T0) is shown in the block diagram above.

## 21.4. Register Description

### 21.4.1. General Timer/Counter Control Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** GTCCR
**Offset:** 0x43
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x23

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TSM | | | | | | PSRASY | PSRSYNC |
| Access | R/W | | | | | | R/W | R/W |
| Reset | 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 PSRASY and PSRSYNC bits is kept, hence keeping the corresponding prescaler reset signals asserted. This ensures that the corresponding Timer/Counters are halted and can be configured to the same value without the risk of one of them advancing during configuration. When the TSM bit is written to zero, the PSRASY and PSRSYNC bits are cleared by hardware, and the Timer/Counters start counting simultaneously.

**Bit 1 – PSRASY: Prescaler Reset Timer/Counter2**

When this bit is one, the Timer/Counter2 prescaler will be reset. This bit is normally cleared immediately by hardware. If the bit is written when Timer/Counter2 is operating in asynchronous mode, the bit will remain one until the prescaler has been reset. The bit will not be cleared by hardware if the TSM bit is set.

**Bit 0 – PSRSYNC: Prescaler Reset**

When this bit is one, Timer/Counter1 and Timer/Counter0 prescaler will be Reset. This bit is normally cleared immediately by hardware, except if the TSM bit is set. Note that Timer/Counter1 and Timer/Counter0 share the same prescaler and a reset of this prescaler will affect both timers.

# 22. TC2 - 8-bit Timer/Counter2 with PWM and Asynchronous Operation

## 22.1. Features

- Channel Counter
- Clear Timer on Compare Match (Auto Reload)
- Glitch-free, Phase Correct Pulse Width Modulator (PWM)
- Frequency Generator
- 10-bit Clock Prescaler
- Overflow and Compare Match Interrupt Sources (TOV2, OCF2A, and OCF2B)
- Allows Clocking from External 32kHz Watch Crystal Independent of the I/O Clock

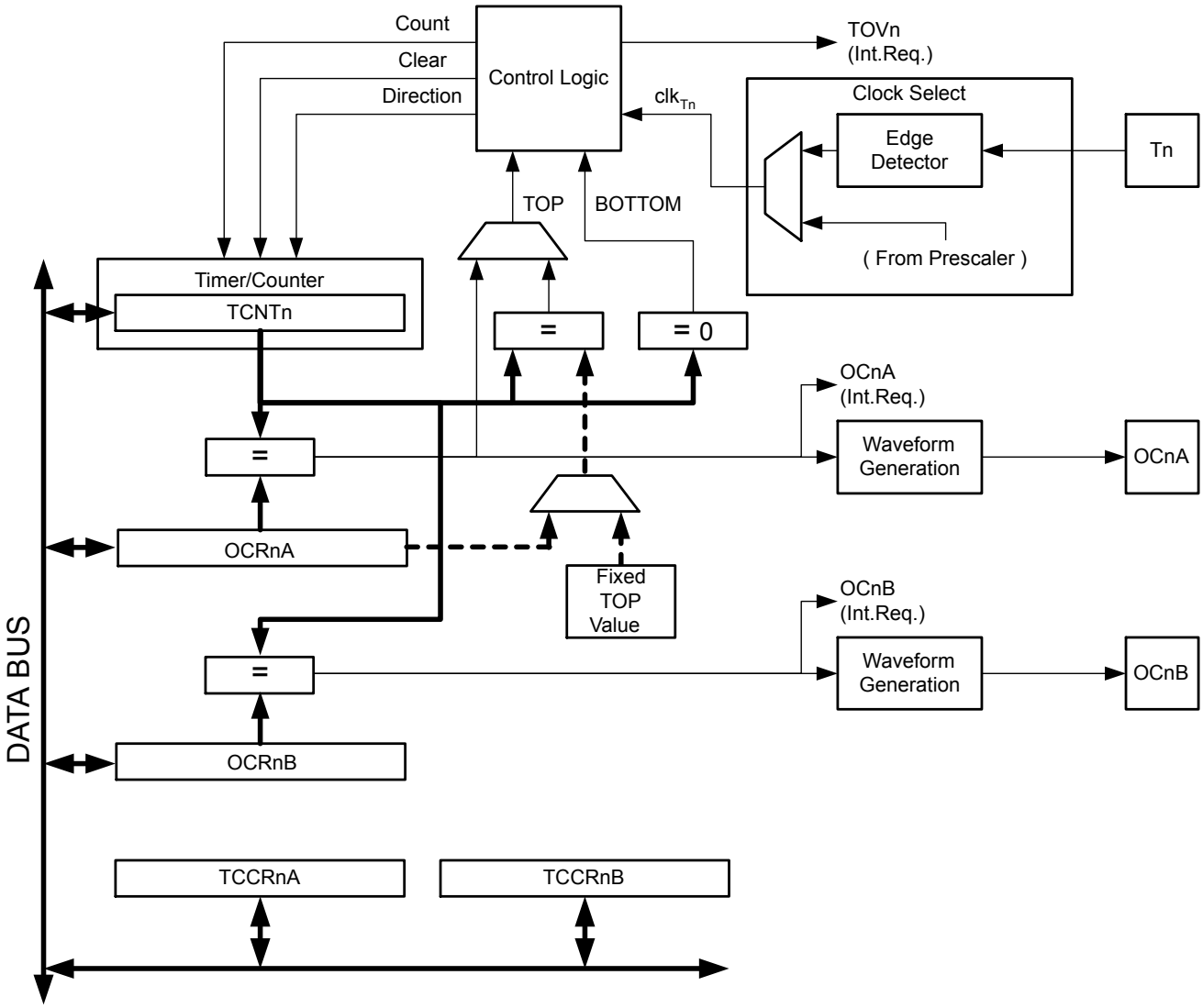
## 22.2. Overview

Timer/Counter2 (TC2) is a general purpose, channel, 8-bit Timer/Counter module.

A simplified block diagram of the 8-bit Timer/Counter is shown below. 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 following Register Description. For the actual placement of I/O pins, refer to the pinout diagram.

The TC2 is enabled when the PRTIM2 bit in the Power Reduction Register (PRR.PRTIM2) is written to '1'.

**Figure 22-1. 8-bit Timer/Counter Block Diagram**




**Related Links**

Pin Configurations on page 15

### 22.2.1. Definitions

Many register and bit references in this section are written in general form:

- n=2 represents the Timer/Counter number
- x=A,B represents the Output Compare Unit A or B

However, when using the register or bit definitions in a program, the precise form must be used, i.e., TCNT2 for accessing Timer/Counter2 counter value.

The following definitions are used throughout the section:

**Table 22-1. Definitions**

| Constant | Description |
|---|---|
| BOTTOM | The counter reaches the BOTTOM when it becomes zero (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 OCR2A Register. The assignment is dependent on the mode of operation. |

### 22.2.2. Registers

The Timer/Counter (TCNT2) and Output Compare Register (OCR2A and OCR2B) are 8-bit registers. Interrupt request (shorten as Int.Req.) signals are all visible in the Timer Interrupt Flag Register (TIFR2). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK2). TIFR2 and TIMSK2 are not shown in the figure.

The Timer/Counter can be clocked internally, via the prescaler, or asynchronously clocked from the TOSC1/2 pins, as detailed later in this section. The asynchronous operation is controlled by the Asynchronous Status Register (ASSR). The Clock Select logic block controls which clock source he 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 ($clk_{T2}$).

The double buffered Output Compare Register (OCR2A and OCR2B) are 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 (OC2A and OC2B). See Output Compare Unit for details. The compare match event will also set the Compare Flag (OCF2A or OCF2B) which can be used to generate an Output Compare interrupt request.

## 22.3. Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal synchronous or an external asynchronous clock source:

The clock source $clk_{T2}$ is by default equal/synchronous to the MCU clock, $clk_{I/O}$.

When the Asynchronous TC2 bit in the Asynchronous Status Register (ASSR.AS2) is written to '1', the clock source is taken from the Timer/Counter Oscillator connected to TOSC1 and TOSC2.

For details on asynchronous operation, see the description of the ASSR. For details on clock sources and prescaler, see Timer/Counter Prescaler.

## 22.4. Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Below is the block diagram of the counter and its surroundings.

**Figure 22-2. Counter Unit Block Diagram**

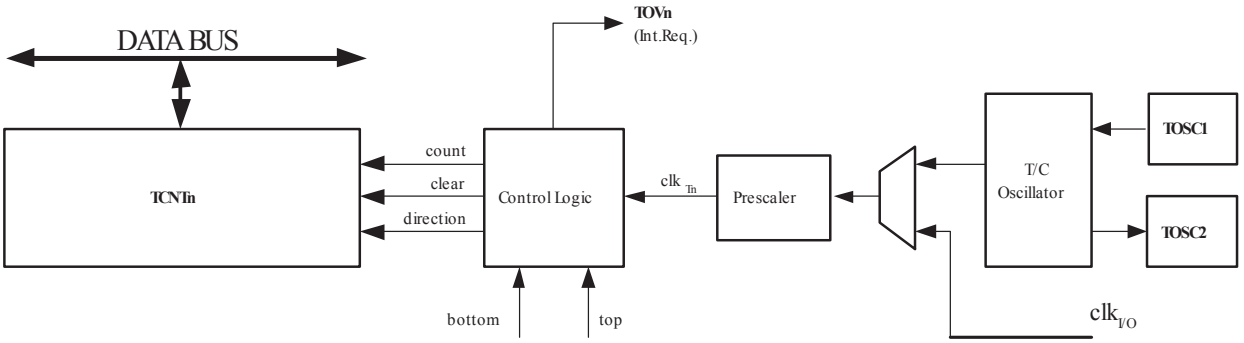



**Table 22-2. Signal description (internal signals):**

| Signal name | Description |
|---|---|
| count | Increment or decrement TCNT2 by 1. |
| direction | Selects between increment and decrement. |
| clear | Clear TCNT2 (set all bits to zero). |
| $clk_{Tn}$ | Timer/Counter clock, referred to as $clk_{T2}$ in the following. |
| top | Signalizes that TCNT2 has reached maximum value. |
| bottom | Signalizes that TCNT2 has reached minimum value (zero). |

Depending on the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock ($clk_{T2}$). $clk_{T2}$ can be generated from an external or internal clock source, selected by the Clock Select bits (CS2[2:0]). When no clock source is selected (CS2[2:0]=0x0) the timer is stopped. However, the TCNT2 value can be accessed by the CPU, regardless of whether $clk_{T2}$ 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 WGM21 and WGM20 bits located in the Timer/Counter Control Register (TCCR2A) and the WGM22 bit located in the Timer/Counter Control Register B (TCCR2B). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare outputs OC2A and OC2B. For more details about advanced counting sequences and waveform generation, see "Modes of Operation".

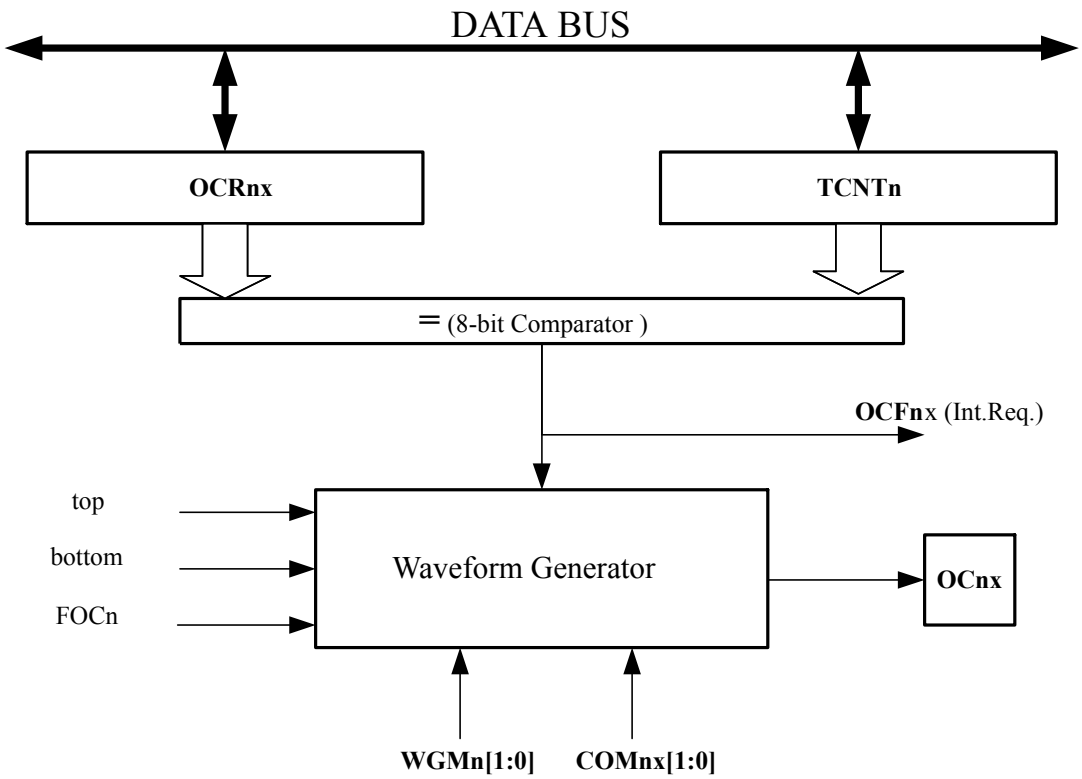
The Timer/Counter Overflow Flag (TOV2) is set according to the mode of operation selected by the TCC2B.WGM2[2:0] bits. TOV2 can be used for generating a CPU interrupt.

## 22.5. Output Compare Unit

The 8-bit comparator continuously compares TCNT2 with the Output Compare Register (OCR2A and OCR2B). Whenever TCNT2 equals OCR2A or OCR2B, the comparator signals a match. A match will set the Output Compare Flag (OCF2A or OCF2B) 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 Output Compare 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 WGM2[2:0] bits and Compare Output mode (COM2x[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).

The following figure shows a block diagram of the Output Compare unit.

**Figure 22-3. Output Compare Unit, Block Diagram**




The OCR2x Register is 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 OCR2x 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.

The OCR2x Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR2x Buffer Register, and if double buffering is disabled the CPU will access the OCR2x directly.

**Related Links**

Modes of Operation on page 138

### 22.5.1. 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 (FOC2x) bit. Forcing compare match will not set the OCF2x Flag or reload/clear the timer, but the OC2x pin will be updated as if a real compare match had occurred (the COM2x[1:0] bits settings define whether the OC2x pin is set, cleared or toggled).

### 22.5.2. Compare Match Blocking by TCNT2 Write

All CPU write operations to the TCNT2 Register will block any compare match that occurs in the next timer clock cycle, even when the timer is stopped. This feature allows OCR2x to be initialized to the same value as TCNT2 without triggering an interrupt when the Timer/Counter clock is enabled.

### 22.5.3. Using the Output Compare Unit

Since writing TCNT2 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT2 when using the Output Compare channel, independently of whether the Timer/Counter is running or not. If the value written to TCNT2 equals the OCR2x value, the

compare match will be missed, resulting in incorrect waveform generation. Similarly, do not write the TCNT2 value equal to BOTTOM when the counter is downcounting.

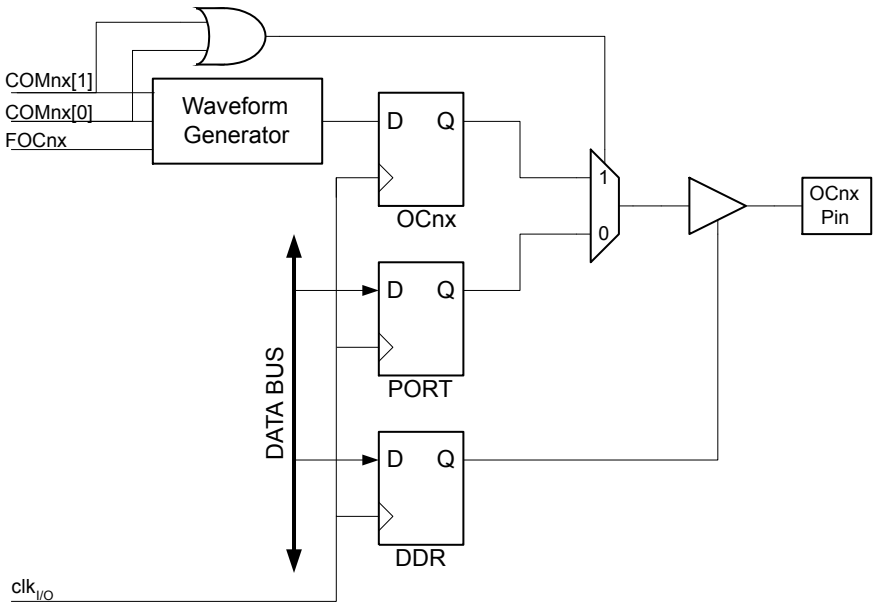
The setup of the OC2x should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC2x value is to use the Force Output Compare (FOC2x) strobe bit in Normal mode. The OC2x Register keeps its value even when changing between Waveform Generation modes.

Be aware that the COM2x[1:0] bits are not double buffered together with the compare value. Changing the COM2x[1:0] bits will take effect immediately.

## 22.6. Compare Match Output Unit

The Compare Output mode (COM2x[1:0]) bits have two functions. The Waveform Generator uses the COM2x[1:0] bits for defining the Output Compare (OC2x) state at the next compare match. Also, the COM2x[1:0] bits control the OC2x pin output source. The following figure shows a simplified schematic of the logic affected by the COM2x[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 COM2x[1:0] bits are shown. When referring to the OC2x state, the reference is for the internal OC2x Register, not the OC2x pin.

**Figure 22-4. Compare Match Output Unit, Schematic**




The general I/O port function is overridden by the Output Compare (OC2x) from the Waveform Generator if either of the COM2x1:0 bits are set. However, the OC2x 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 OC2x pin (DDR_OC2x) must be set as output before the OC2x 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 OC2x state before the output is enabled. Note that some COM2x[1:0] bit settings are reserved for certain modes of operation. See Register Description.

**Related Links**

Modes of Operation on page 138

### 22.6.1. Compare Output Mode and Waveform Generation

The Waveform Generator uses the COM2x[1:0] bits differently in normal, CTC, and PWM modes. For all modes, setting the COM2x[1:0] = 0 tells the Waveform Generator that no action on the OC2x Register is to be performed on the next compare match. Refer also to the descriptions of the output modes.

A change of the COM2x[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 FOC2x strobe bits.

## 22.7. 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 (WGM2[2:0]) and Compare Output mode (COM2x[1:0]) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM2x[1:0] bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM2x[1:0] bits control whether the output should be set, cleared, or toggled at a compare match (See Compare Match Output Unit).

For detailed timing information refer to Timer/Counter Timing Diagrams.

### 22.7.1. Normal Mode

The simplest mode of operation is the Normal mode (WGM2[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 (TOV2) will be set in the same timer clock cycle as the TCNT2 becomes zero. The TOV2 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 TOV2 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.

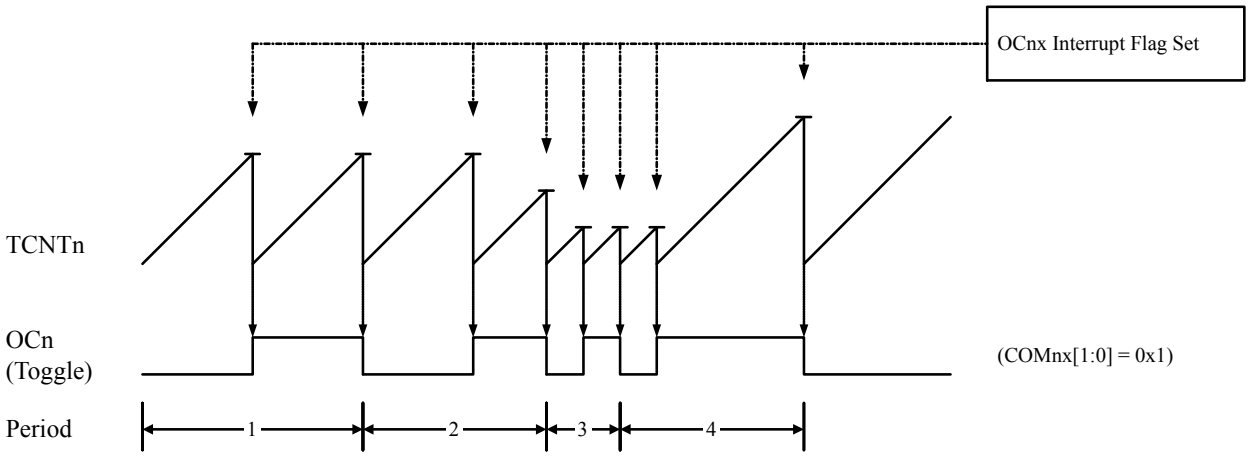
The Output Compare unit can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

### 22.7.2. Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGM2[2:0] = 2), the OCR2A Register is used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT2) matches the OCR2A. The OCR2A 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 as follows. The counter value (TCNT2) increases until a compare match occurs between TCNT2 and OCR2A, and then counter (TCNT2) is cleared.

**Figure 22-5. CTC Mode, Timing Diagram**




An interrupt can be generated each time the counter value reaches the TOP value by using the OCF2A 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 OCR2A is lower than the current value of TCNT2, 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 OC2A output can be set to toggle its logical level on each compare match by setting the Compare Output mode bits to toggle mode (COM2A[1:0] = 1). The OC2A value will not be visible on the port pin unless the data direction for the pin is set to output. The waveform generated will have a maximum frequency of $f_{OC2A} = f_{clk_I/O}/2$ when OCR2A is set to zero (0x00). The waveform frequency is defined by the following equation:

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

The N variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).

As for the Normal mode of operation, the TOV2 Flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

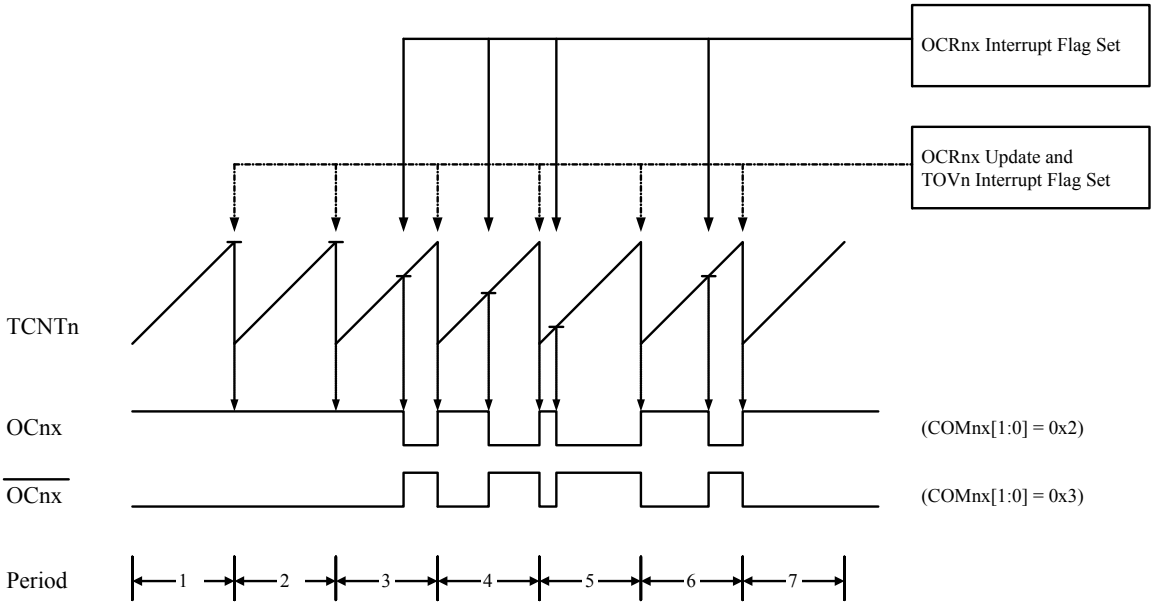
### 22.7.3. Fast PWM Mode

The fast Pulse Width Modulation or fast PWM mode (WGM2[2:0] = 0x3 or 0x7) 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 WGM2[2:0] = 0x3, and OCR2A when WGM2[2:0] = 0x7. In non-inverting Compare Output mode, the Output Compare (OC2x) is cleared on the compare match between TCNT2 and OCR2x, 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 uses 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 depicted in the following figure. The TCNT2 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 TCNT2 slopes represent compare matches between OCR2x and TCNT2.

**Figure 22-6. Fast PWM Mode, Timing Diagram**




The Timer/Counter Overflow Flag (TOV2) 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 OC2x pin. Setting the COM2x1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM2x[1:0] to three. TOP is defined as 0xFF when WGM2[2:0] = 0x3, and OCR2A when MGM2[2:0] = 0x7. The actual OC2x 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 OC2x Register at the compare match between OCR2x and TCNT2, and clearing (or setting) the OC2x 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:

$$f_{\mathrm{OCnxPWM}} = \frac{f_{\mathrm{clk_I/O}}}{N \cdot 256}$$

The N variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).

The extreme values for the OCR2A Register represent special cases when generating a PWM waveform output in the fast PWM mode. If the OCR2A is set equal to BOTTOM, the output will be a narrow spike for each MAX+1 timer clock cycle. Setting the OCR2A equal to MAX will result in a constantly high or low output (depending on the polarity of the output set by the COM2A[1:0] bits.)

A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC2x to toggle its logical level on each compare match (COM2x[1:0] = 1). The waveform generated will have a maximum frequency of $f_{oc2} = f_{clk_I/O}/2$ when OCR2A is set to zero. This feature is similar to the OC2A toggle in CTC mode, except the double buffer feature of the Output Compare unit is enabled in the fast PWM mode.

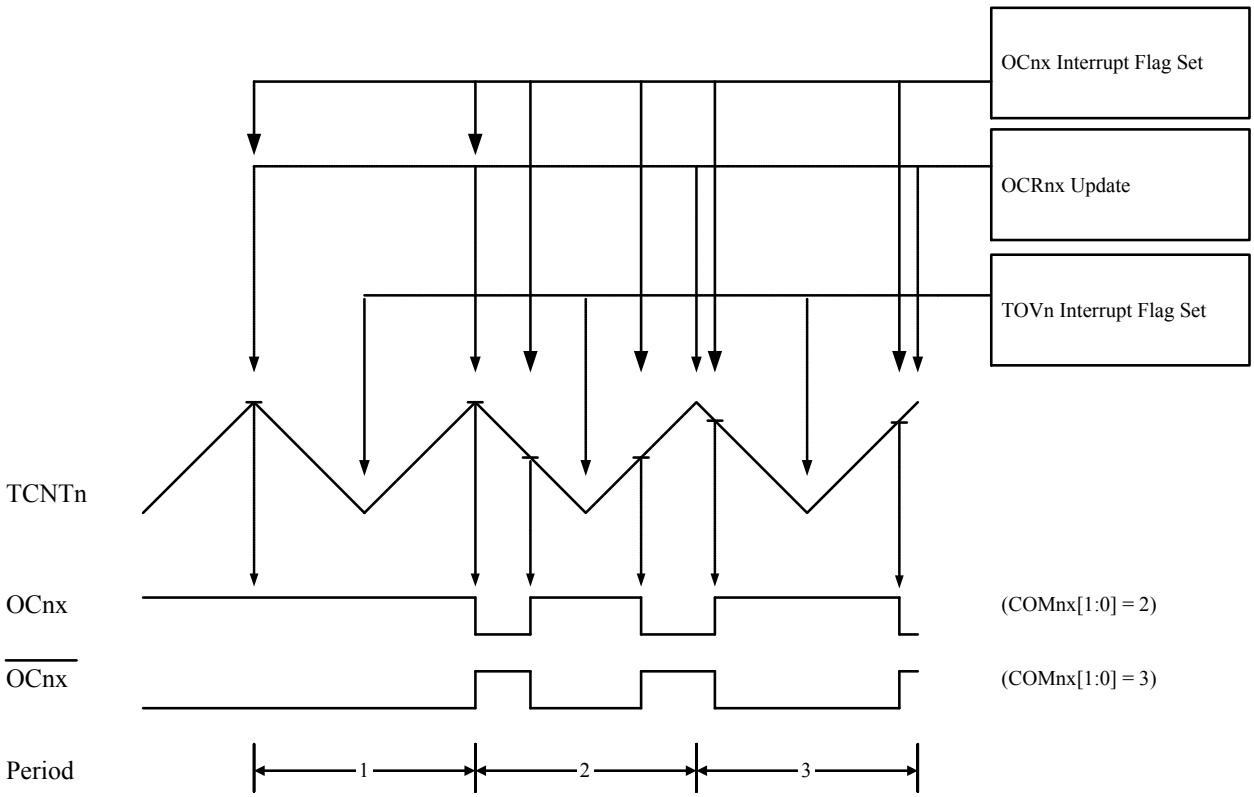
### 22.7.4. Phase Correct PWM Mode

The phase correct PWM mode (WGM2[2:0] = 0x1 or 0x5) 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 WGM2[2:0] = 0x3, and OCR2A when MGM2[2:0] = 7. In non-inverting Compare Output mode, the Output Compare (OC2x) is cleared on the compare match between TCNT2 and OCR2x 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.

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 TCNT2 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 22-7. The TCNT2 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 TCNT2 slopes represent compare matches between OCR2x and TCNT2.

**Figure 22-7. Phase Correct PWM Mode, Timing Diagram**




The Timer/Counter Overflow Flag (TOV2) 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 OC2x pin. Setting the COM2x[1:0] bits to two will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM2x[1:0] to three. TOP is defined as 0xFF when WGM2[2:0] = 0x3, and OCR2A when WGM2[2:0] = 7. The actual OC2x 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 OC2x Register at the compare match between OCR2x and TCNT2 when the counter increments, and setting (or clearing) the OC2x Register at compare match between OCR2x and TCNT2 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{\text{OCnxPCPWM}} = \frac{f_{\text{clk_I/O}}}{N \cdot 510}$$

The N variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).

The extreme values for the OCR2A Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR2A is set equal to BOTTOM, the output will be continuously low and if set equal to MAX the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

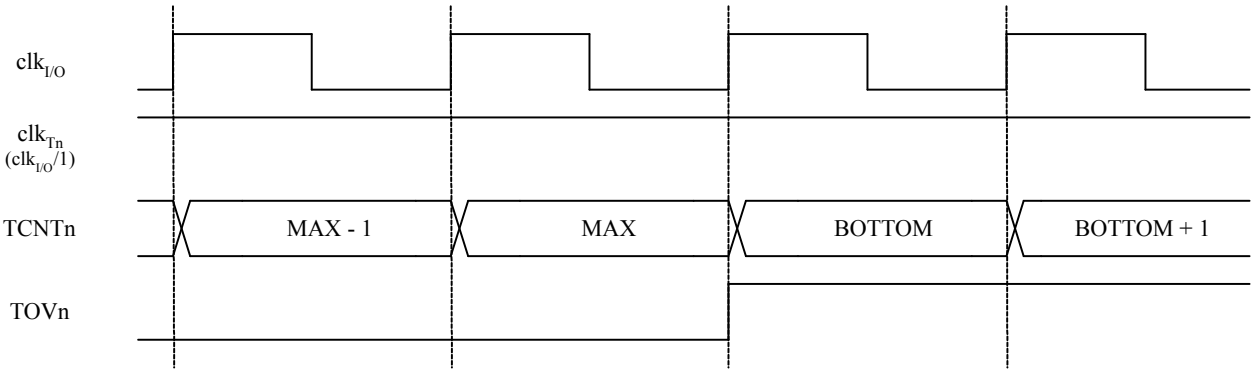
At the very start of period 2 in the above figure OC2x has a transition from high to low even though there is no Compare Match. The point of this transition is to guarantee symmetry around BOTTOM. There are two cases that give a transition without Compare Match.

- OCR2A changes its value from MAX, as shown in the preceeding figure. When the OCR2A value is MAX the OC2 pin value is the same as the result of a down-counting compare match. To ensure symmetry around BOTTOM the OC2 value at MAX must correspond to the result of an up-counting Compare Match.
- The timer starts counting from a value higher than the one in OCR2A, and for that reason misses the Compare Match and hence the OC2 change that would have happened on the way up.

## 22.8. Timer/Counter Timing Diagrams

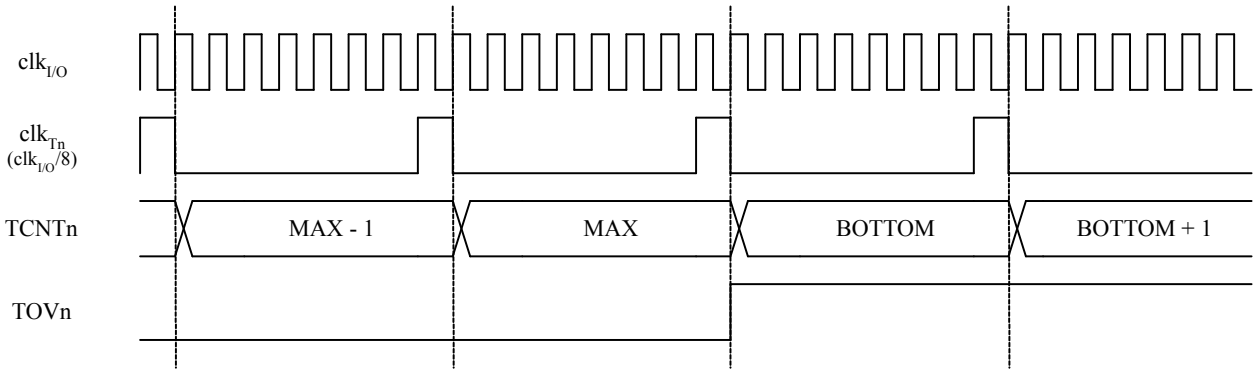
The following figures show the Timer/Counter in synchronous mode, and the timer clock ($clk_{T2}$) is therefore shown as a clock enable signal. In asynchronous mode, $clk_{I/O}$ should be replaced by the Timer/Counter Oscillator clock. The figures include information on when Interrupt Flags are set. The following figure 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 22-8. Timer/Counter Timing Diagram, no Prescaling**



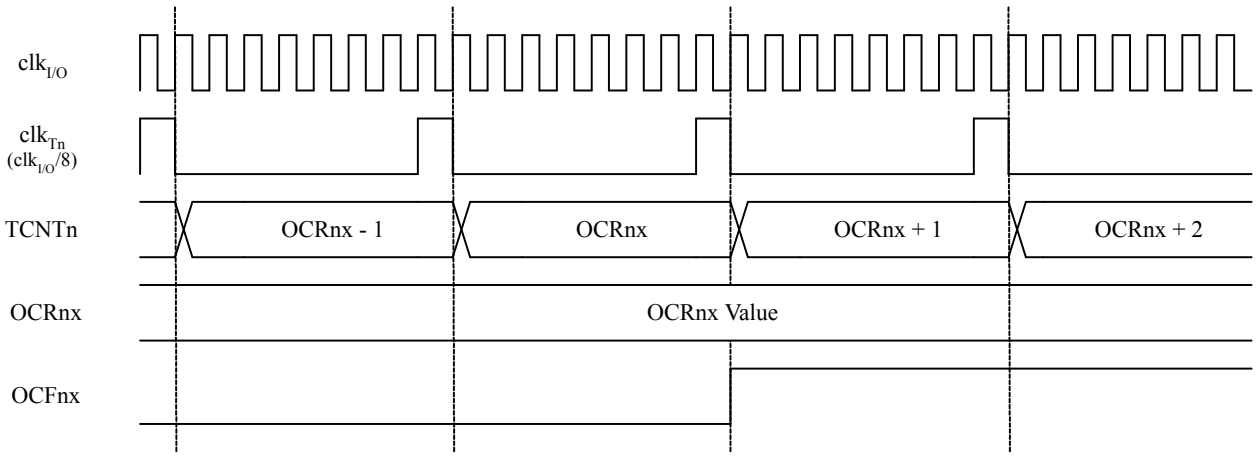

The following figure shows the same timing data, but with the prescaler enabled.

**Figure 22-9. Timer/Counter Timing Diagram, with Prescaler ($f_{clk_I/O}/8$)**



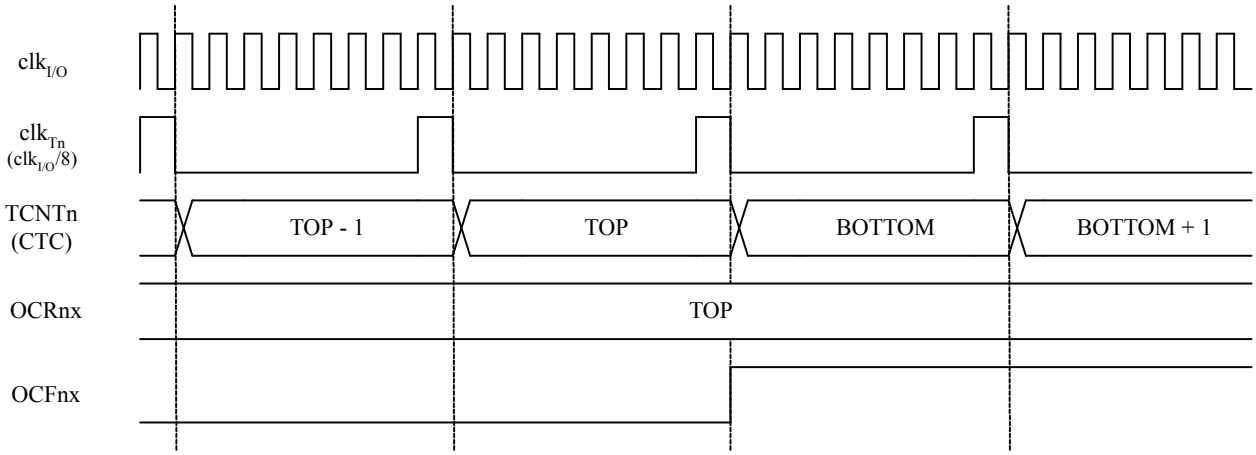

The following figure shows the setting of OCF2A in all modes except CTC mode.

**Figure 22-10. Timer/Counter Timing Diagram, Setting of OCF2A, with Prescaler ($f_{clk_I/O}/8$)**




The following figure shows the setting of OCF2A and the clearing of TCNT2 in CTC mode.

**Figure 22-11. Timer/Counter Timing Diagram, Clear Timer on Compare Match mode, with Prescaler ($f_{clk_I/O}/8$)**




## 22.9. Asynchronous Operation of Timer/Counter2

When TC2 operates asynchronously, some considerations must be taken:

- When switching between asynchronous and synchronous clocking of TC2, the registers TCNT2, OCR2x, and TCCR2x might be corrupted. A safe procedure for switching clock source is:

1. Disable the TC2 interrupts by clearing OCIE2x and TOIE2.
2. Select clock source by setting AS2 as appropriate.
3. Write new values to TCNT2, OCR2x, and TCCR2x.
4. To switch to asynchronous operation: Wait for TCN2xUB, OCR2xUB, and TCR2xUB.
5. Clear the TC2 Interrupt Flags.
6. Enable interrupts, if needed.

- The CPU main clock frequency must be more than four times the oscillator frequency.
- When writing to one of the registers TCNT2, OCR2x, or TCCR2x, the value is transferred to a temporary register, and latched after two positive edges on TOSC1. The user should not write a new value before the contents of the temporary register have been transferred to its destination. Each of the five mentioned registers has its individual temporary register, which means that e.g. writing to TCNT2 does not disturb an OCR2x write in progress. The Asynchronous Status Register (ASSR) indicates that a transfer to the destination register has taken place.
- When entering Power-save or ADC Noise Reduction mode after having written to TCNT2, OCR2x, or TCCR2x, the user must wait until the written register has been updated if TC2 is used to wake up the device. Otherwise, the MCU will enter sleep mode before the changes are effective. This is particularly important if any of the Output Compare2 interrupts is used to wake up the device, since the Output Compare function is disabled during writing to OCR2x or TCNT2. If the write cycle is not finished, and the MCU enters sleep mode before the corresponding OCR2xUB bit returns to zero, the device will never receive a compare match interrupt, and the MCU will not wake up.
- If TC2 is used to wake the device up from Power-save or ADC Noise Reduction mode, precautions must be taken if the user wants to re-enter one of these modes: If re-entering sleep mode within the TOSC1 cycle, the interrupt will immediately occur and the device wake up again. The result is multiple interrupts and wake-ups within one TOSC1 cycle from the first interrupt. If the user is in doubt whether the time before re-entering Power-save or ADC Noise Reduction mode is sufficient, the following algorithm can be used to ensure that one TOSC1 cycle has elapsed:
  1. Write a value to TCCR2x, TCNT2, or OCR2x.
  2. Wait until the corresponding Update Busy Flag in ASSR returns to zero.
  3. Enter Power-save or ADC Noise Reduction mode.
- When the asynchronous operation is selected, the 32.768kHz oscillator for TC2 is always running, except in Power-down and Standby modes. After a Power-up Reset or wake-up from Power-down or Standby mode, the user should be aware of the fact that this oscillator might take as long as one second to stabilize. The user is advised to wait for at least one second before using TC2 after power-up or wake-up from Power-down or Standby mode. The contents of all TC2 Registers must be considered lost after a wake-up from Power-down or Standby mode due to unstable clock signal upon start-up, no matter whether the Oscillator is in use or a clock signal is applied to the TOSC1 pin.
- Description of wake up from Power-save or ADC Noise Reduction mode when the timer is clocked asynchronously: When the interrupt condition is met, the wake up process is started on the following cycle of the timer clock, that is, the timer is always advanced by at least one before the processor can read the counter value. After wake-up, the MCU is halted for four cycles, it executes the interrupt routine, and resumes execution from the instruction following SLEEP.
- Reading of the TCNT2 Register shortly after wake-up from Power-save may give an incorrect result. Since TCNT2 is clocked on the asynchronous TOSC clock, reading TCNT2 must be done through a register synchronized to the internal I/O clock domain. Synchronization takes place for every rising TOSC1 edge. When waking up from Power-save mode, and the I/O clock ($clk_{I/O}$) again becomes active, TCNT2 will read as the previous value (before entering sleep) until the next rising TOSC1 edge. The phase of the TOSC clock after waking up from Power-save mode is essentially

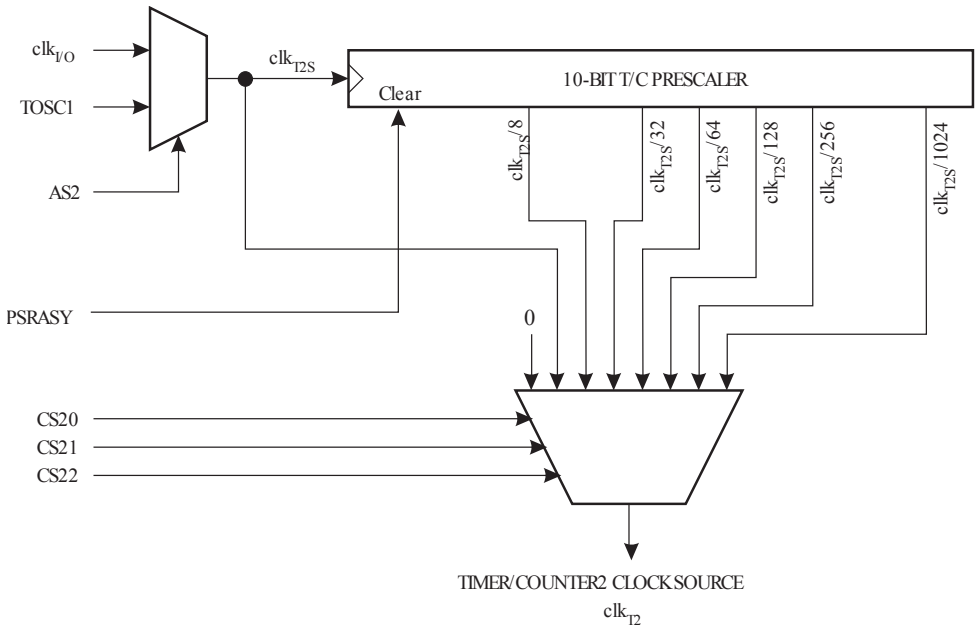
unpredictable, as it depends on the wake-up time. The recommended procedure for reading TCNT2 is thus as follows:

1. Wait for the corresponding Update Busy Flag to be cleared.
2. Read TCNT2.

- During asynchronous operation, the synchronization of the Interrupt Flags for the asynchronous timer takes 3 processor cycles plus one timer cycle. The timer is therefore advanced by at least one before the processor can read the timer value causing the setting of the Interrupt Flag. The Output Compare pin is changed on the timer clock and is not synchronized to the processor clock.

## 22.10. Timer/Counter Prescaler

**Figure 22-12. Prescaler for TC2**




The clock source for TC2 is named $clk_{T2S}$. It is by default connected to the main system I/O clock $clk_{I/O}$. By writing a '1' to the Asynchronous TC2 bit in the Asynchronous Status Register (ASSR.AS2), TC2 is asynchronously clocked from the TOSC1 pin. This enables use of TC2 as a Real Time Counter (RTC). When AS2 is set, pins TOSC1 and TOSC2 are disconnected from Port B. A crystal can then be connected between the TOSC1 and TOSC2 pins to serve as an independent clock source for TC2. The Oscillator is optimized for use with a 32.768kHz crystal.

For TC2, the possible prescaled selections are: $clk_{T2S}/8$, $clk_{T2S}/32$, $clk_{T2S}/64$, $clk_{T2S}/128$, $clk_{T2S}/256$, and $clk_{T2S}/1024$. Additionally, $clk_{T2S}$ as well as 0 (stop) may be selected. The prescaler is reset by writing a '1' to the Prescaler Reset TC2 bit in the General TC2 Control Register (GTCCR.PSRASY). This allows the user to operate with a defined prescaler.

## 22.11. Register Description

### 22.11.1. TC2 Control Register A

**Name:** TCCR2A
**Offset:** 0xB0
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | COM2A1 | COM2A0 | COM2B1 | COM2B0 | | | WGM21 | WGM20 |
| Access | R/W | R/W | R/W | R/W | | | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | | | 0 | 0 |

**Bits 7:6 – COM2An: Compare Output Mode for Channel A [n = 1:0]**
These bits control the Output Compare pin (OC2A) behavior. If one or both of the COM2A[1:0] bits are set, the OC2A 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 OC2A pin must be set in order to enable the output driver.

When OC2A is connected to the pin, the function of the COM2A[1:0] bits depends on the WGM2[2:0] bit setting. The table below shows the COM2A[1:0] bit functionality when the WGM2[2:0] bits are set to a normal or CTC mode (non- PWM).

**Table 22-3. Compare Output Mode, non-PWM**

| COM2A1 | COM2A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC2A disconnected. |
| 0 | 1 | Toggle OC2A on Compare Match. |
| 1 | 0 | Clear OC2A on Compare Match. |
| 1 | 1 | Set OC2A on Compare Match . |

The table below shows the COM2A[1:0] bit functionality when the WGM2[1:0] bits are set to fast PWM mode.

**Table 22-4. Compare Output Mode, Fast PWM(1)**

| COM2A1 | COM2A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC2A disconnected. |
| 0 | 1 | WGM22 = 0: Normal Port Operation, OC2A Disconnected<br>WGM22 = 1: Toggle OC2A on Compare Match |
| 1 | 0 | Clear OC2A on Compare Match, set OC2A at BOTTOM (non-inverting mode) |
| 1 | 1 | Set OC2A on Compare Match, clear OC2A at BOTTOM (inverting mode) |

**Note:**

1. A special case occurs when OCR2A equals TOP and COM2A1 is set. In this case the compare match is ignored, but the set or clear is done at BOTTOM. Refer to Fast PWM Mode for details.

The table below shows the COM2A[1:0] bit functionality when the WGM2[2:0] bits are set to phase correct PWM mode.

**Table 22-5. Compare Output Mode, Phase Correct PWM Mode(1)**

| COM2A1 | COM2A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC2A disconnected. |
| 0 | 1 | WGM22 = 0: Normal Port Operation, OC2A Disconnected.<br>WGM22 = 1: Toggle OC2A on Compare Match. |
| 1 | 0 | Clear OC2A on Compare Match when up-counting. Set OC2A on Compare Match when down-counting. |
| 1 | 1 | Set OC2A on Compare Match when up-counting. Clear OC2A on Compare Match when down-counting. |

**Note:**

1. A special case occurs when OCR2A equals TOP and COM2A1 is set. In this case, the Compare Match is ignored, but the set or clear is done at TOP. Refer to Phase Correct PWM Mode for details.

**Bits 5:4 – COM2Bn: Compare Output Mode for Channel B [n = 1:0]**

These bits control the Output Compare pin (OC2B) behavior. If one or both of the COM2B[1:0] bits are set, the OC2B 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 OC2B pin must be set in order to enable the output driver.

When OC2B is connected to the pin, the function of the COM2B[1:0] bits depends on the WGM2[2:0] bit setting. The table shows the COM2B[1:0] bit functionality when the WGM2[2:0] bits are set to a normal or CTC mode (non- PWM).

**Table 22-6. Compare Output Mode, non-PWM**

| COM2B1 | COM2B0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC2B disconnected. |
| 0 | 1 | Toggle OC2B on Compare Match. |
| 1 | 0 | Clear OC2B on Compare Match. |
| 1 | 1 | Set OC2B on Compare Match. |

The table below shows the COM0B[1:0] bit functionality when the WGM0[2:0] bits are set to fast PWM mode.

**Table 22-7. Compare Output Mode, Fast PWM(1)**

| 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 OCR2B equals TOP and COM2B[1] is set. In this case, the Compare Match is ignored, but the set or clear is done at TOP. Refer to Fast PWM Mode for details.

The table below shows the COM2B[1:0] bit functionality when the WGM2[2:0] bits are set to phase correct PWM mode.

**Table 22-8. Compare Output Mode, Phase Correct PWM Mode(1)**

| COM2B1 | COM2B0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC2B disconnected. |
| 0 | 1 | Reserved |
| 1 | 0 | Clear OC2B on Compare Match when up-counting. Set OC2B on Compare Match when down-counting. |
| 1 | 1 | Set OC2B on Compare Match when up-counting. Clear OC2B on Compare Match when down-counting. |

**Note:**

1. A special case occurs when OCR2B equals TOP and COM2B1 is set. In this case, the Compare Match is ignored, but the set or clear is done at TOP. Refer to Phase Correct PWM Mode for details.

**Bits 1:0 – WGM2n: Waveform Generation Mode [n = 1:0]**

Combined with the WGM22 bit found in the TCCR2B 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. 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).

**Table 22-9. Waveform Generation Mode Bit Description**

| Mode | WGM22 | WGM21 | WGM20 | Timer/Counter Mode of Operation | TOP | Update of OCR0x 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
2. BOTTOM = 0x00

### 22.11.2. TC2 Control Register B

**Name:** TCCR2B
**Offset:** 0xB1
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | FOC2A | FOC2B | | | WGM22 | CS2[2:0] | | |
| Access | R/W | R/W | | | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | | | 0 | 0 | 0 | 0 |

**Bit 7 – FOC2A: Force Output Compare A**
The FOC2A bit is only active when the WGM bits specify a non-PWM mode.

To ensure compatibility with future devices, this bit must be set to zero when TCCR2B is written when operating in PWM mode. When writing a logical one to the FOC2A bit, an immediate Compare Match is forced on the Waveform Generation unit. The OC2A output is changed according to its COM2A[1:0] bits setting. Note that the FOC2A bit is implemented as a strobe. Therefore it is the value present in the COM2A[1:0] bits that determines the effect of the forced compare.

A FOC2A strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR2A as TOP.

The FOC2A bit is always read as zero.

**Bit 6 – FOC2B: Force Output Compare B**
The FOC2B bit is only active when the WGM bits specify a non-PWM mode.

To ensure compatibility with future devices, this bit must be set to zero when TCCR2B is written when operating in PWM mode. When writing a logical one to the FOC2B bit, an immediate Compare Match is forced on the Waveform Generation unit. The OC2B output is changed according to its COM2B[1:0] bits setting. Note that the FOC2B bit is implemented as a strobe. Therefore it is the value present in the COM2B[1:0] bits that determines the effect of the forced compare.

A FOC2B strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR2B as TOP.

The FOC2B bit is always read as zero.

**Bit 3 – WGM22: Waveform Generation Mode**
Refer to TCCR2A.

**Bits 2:0 – CS2[2:0]: Clock Select 2 [n = 0..2]**
The three Clock Select bits select the clock source to be used by the Timer/Counter.

**Table 22-10. Clock Select Bit Description**

| CA22 | CA21 | CS20 | Description |
|---|---|---|---|
| 0 | 0 | 0 | No clock source (Timer/Counter stopped). |
| 0 | | 1 | $clk_{I/O}$/1 (No prescaling) |
| 0 | 1 | 0 | $clk_{I/O}$/8 (From prescaler) |

| CA22 | CA21 | CS20 | Description |
|---|---|---|---|
| 0 | 1 | 1 | $clk_{I/O}$/32 (From prescaler) |
| 1 | 0 | 0 | clkI/O/64 (From prescaler) |
| 1 | 0 | 1 | $clk_{I/O}$/128 (From prescaler) |
| 1 | 1 | 0 | $clk_{I/O}$/256 (From prescaler) |
| 1 | 1 | 1 | $clk_{I/O}$/1024 (From prescaler) |

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.

### 22.11.3. TC2 Counter Value Register

**Name:** TCNT2
**Offset:** 0xB2
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TCNT2[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – TCNT2[7:0]: Timer/Counter 2 Counter Value**
The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNT2 Register blocks (removes) the Compare Match on the following timer clock. Modifying the counter (TCNT2) while the counter is running, introduces a risk of missing a Compare Match between TCNT2 and the OCR2x Registers.

### 22.11.4. TC2 Output Compare Register A

**Name:** OCR2A
**Offset:** 0xB3
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OCR2A[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – OCR2A[7:0]: Output Compare 2 A**
The Output Compare Register A contains an 8-bit value that is continuously compared with the counter value (TCNT2). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC2A pin.

### 22.11.5. TC2 Output Compare Register B

**Name:** OCR2B
**Offset:** 0xB4
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OCR2B[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – OCR2B[7:0]: Output Compare 2 B**
The Output Compare Register B contains an 8-bit value that is continuously compared with the counter value (TCNT2). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC2B pin.

### 22.11.6. TC2 Interrupt Mask Register

**Name:** TIMSK2
**Offset:** 0x70
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | OCIEB | OCIEA | TOIE |
| Access | | | | | | R/W | R/W | R/W |
| Reset | | | | | | 0 | 0 | 0 |

**Bit 2 – OCIEB: Timer/Counter2, Output Compare B Match Interrupt Enable**
When the OCIEB bit is written to '1' and the I-bit in the Status Register is set (one), the Timer/Counter2 Compare Match B interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter2 occurs, i.e., when the OCFB bit is set in TIFR2.

**Bit 1 – OCIEA: Timer/Counter2, Output Compare A Match Interrupt Enable**
When the OCIEA bit is written to '1' and the I-bit in the Status Register is set (one), the Timer/Counter2 Compare Match A interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter2 occurs, i.e., when the OCFA bit is set in TIFR2.

**Bit 0 – TOIE: Timer/Counter2, Overflow Interrupt Enable**
When the TOIE bit is written to '1' and the I-bit in the Status Register is set (one), the Timer/Counter2 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter2 occurs, i.e., when the TOV bit is set in TIFR2.

### 22.11.7. TC2 Interrupt Flag Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** TIFR2
**Offset:** 0x37
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x17

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | OCFB | OCFA | TOV |
| Access | | | | | | R/W | R/W | R/W |
| Reset | | | | | | 0 | 0 | 0 |

**Bit 2 – OCFB: Timer/Counter2, Output Compare B Match Flag**
The OCFB bit is set (one) when a compare match occurs between the Timer/Counter2 and the data in OCRB – Output Compare Register2. OCFB is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCFB is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIEB (Timer/Counter2 Compare match Interrupt Enable), and OCFB are set (one), the Timer/Counter2 Compare match Interrupt is executed.

**Bit 1 – OCFA: Timer/Counter2, Output Compare A Match Flag**
The OCFA bit is set (one) when a compare match occurs between the Timer/Counter2 and the data in OCRA – Output Compare Register2. OCFA is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCFA is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIEA (Timer/Counter2 Compare match Interrupt Enable), and OCFA are set (one), the Timer/Counter2 Compare match Interrupt is executed.

**Bit 0 – TOV: Timer/Counter2, Overflow Flag**
The TOV bit is set (one) when an overflow occurs in Timer/Counter2. TOV is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TOV is cleared by writing a logic one to the flag. When the SREG I-bit, TOIEA (Timer/Counter2 Overflow Interrupt Enable), and TOV are set (one), the Timer/Counter2 Overflow interrupt is executed. In PWM mode, this bit is set when Timer/Counter2 changes counting direction at 0x00.

### 22.11.8. Asynchronous Status Register

**Name:** ASSR
**Offset:** 0xB6
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | EXCLK | AS2 | TCN2UB | OCR2AUB | OCR2BUB | TCR2AUB | TCR2BUB |
| Access | | R | R | R | R | R | R | R |
| Reset | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bit 6 – EXCLK: Enable External Clock Input**
When EXCLK is written to one, and asynchronous clock is selected, the external clock input buffer is enabled and an external clock can be input on Timer Oscillator 1 (TOSC1) pin instead of a 32kHz crystal. Writing to EXCLK should be done before asynchronous operation is selected. Note that the crystal Oscillator will only run when this bit is zero.

**Bit 5 – AS2: Asynchronous Timer/Counter2**
When AS2 is written to zero, Timer/Counter2 is clocked from the I/O clock, clkI/O. When AS2 is written to one, Timer/Counter2 is clocked from a crystal Oscillator connected to the Timer Oscillator 1 (TOSC1) pin. When the value of AS2 is changed, the contents of TCNT2, OCR2A, OCR2B, TCCR2A and TCCR2B might be corrupted.

**Bit 4 – TCN2UB: Timer/Counter2 Update Busy**
When Timer/Counter2 operates asynchronously and TCNT2 is written, this bit becomes set. When TCNT2 has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that TCNT2 is ready to be updated with a new value.

**Bit 3 – OCR2AUB: Enable External Clock Input**
When Timer/Counter2 operates asynchronously and OCR2A is written, this bit becomes set. When OCR2A has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that OCR2A is ready to be updated with a new value.

**Bit 2 – OCR2BUB: Output Compare Register2 Update Busy**
When Timer/Counter2 operates asynchronously and OCR2B is written, this bit becomes set. When OCR2B has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that OCR2B is ready to be updated with a new value.

**Bit 1 – TCR2AUB: Timer/Counter Control Register2 Update Busy**
When Timer/Counter2 operates asynchronously and TCCR2A is written, this bit becomes set. When TCCR2A has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that TCCR2A is ready to be updated with a new value.

**Bit 0 – TCR2BUB: Timer/Counter Control Register2 Update Busy**
When Timer/Counter2 operates asynchronously and TCCR2B is written, this bit becomes set. When TCCR2B has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that TCCR2B is ready to be updated with a new value.

If a write is performed to any of the five Timer/Counter2 Registers while its update busy flag is set, the updated value might get corrupted and cause an unintentional interrupt to occur.

### 22.11.9. General Timer/Counter Control Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** GTCCR
**Offset:** 0x43
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x23

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TSM | | | | | | PSRASY | PSRSYNC |
| Access | R/W | | | | | | R/W | R/W |
| Reset | 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 PSRASY and PSRSYNC bits is kept, hence keeping the corresponding prescaler reset signals asserted. This ensures that the corresponding Timer/Counters are halted and can be configured to the same value without the risk of one of them advancing during configuration. When the TSM bit is written to zero, the PSRASY and PSRSYNC bits are cleared by hardware, and the Timer/Counters start counting simultaneously.

**Bit 1 – PSRASY: Prescaler Reset Timer/Counter2**

When this bit is one, the Timer/Counter2 prescaler will be reset. This bit is normally cleared immediately by hardware. If the bit is written when Timer/Counter2 is operating in asynchronous mode, the bit will remain one until the prescaler has been reset. The bit will not be cleared by hardware if the TSM bit is set.

**Bit 0 – PSRSYNC: Prescaler Reset**

When this bit is one, Timer/Counter1 and Timer/Counter0 prescaler will be Reset. This bit is normally cleared immediately by hardware, except if the TSM bit is set. Note that Timer/Counter1 and Timer/Counter0 share the same prescaler and a reset of this prescaler will affect both timers.

# 23. SPI – Serial Peripheral Interface

## 23.1. 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

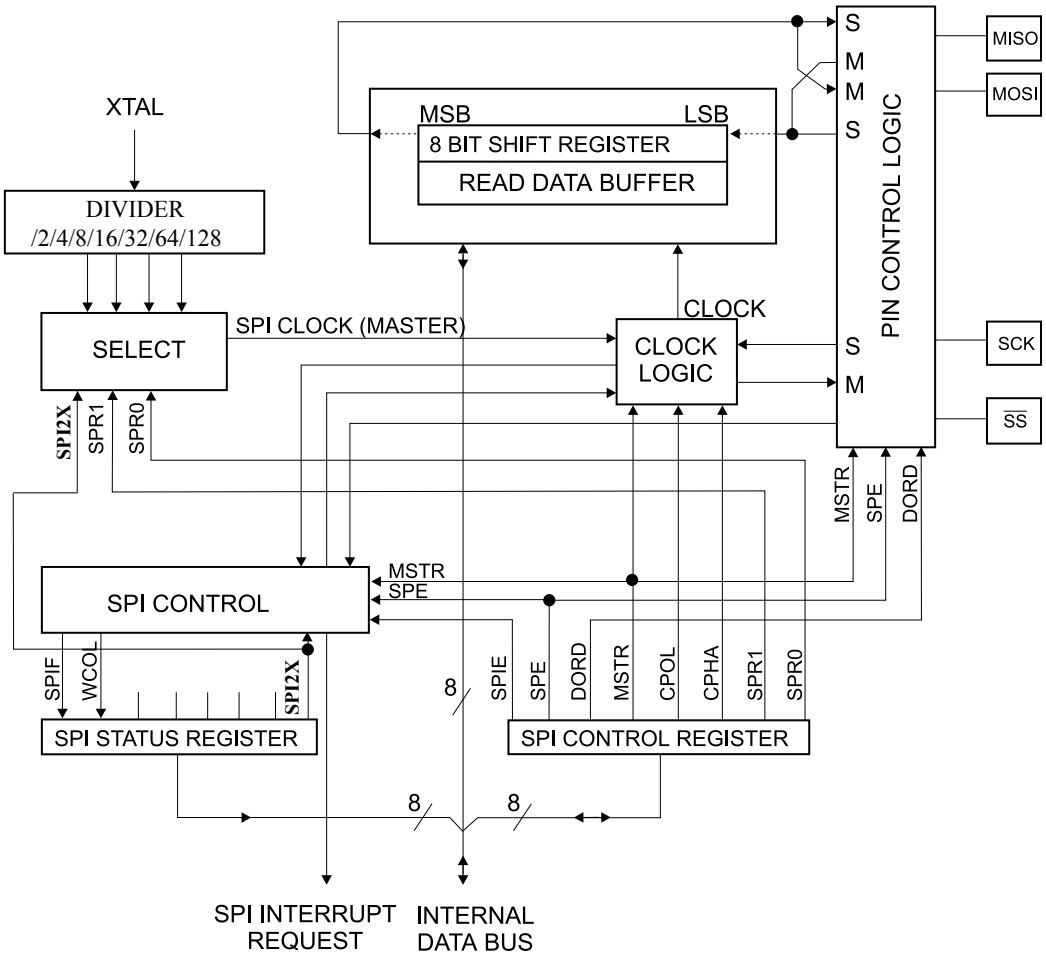
## 23.2. Overview

The Serial Peripheral Interface (SPI) allows high-speed synchronous data transfer between the device and peripheral units, or between several AVR devices.

The USART can also be used in Master SPI mode, please refer to *USART in SPI Mode* chapter.

To enable the SPI module, Power Reduction Serial Peripheral Interface bit in the Power Reduction Register (PRR.PRSPI0) must be written to '0'.

**Figure 23-1. SPI Block Diagram**




**Note:** Refer to the pin-out description and the IO Port description for SPI pin placement.

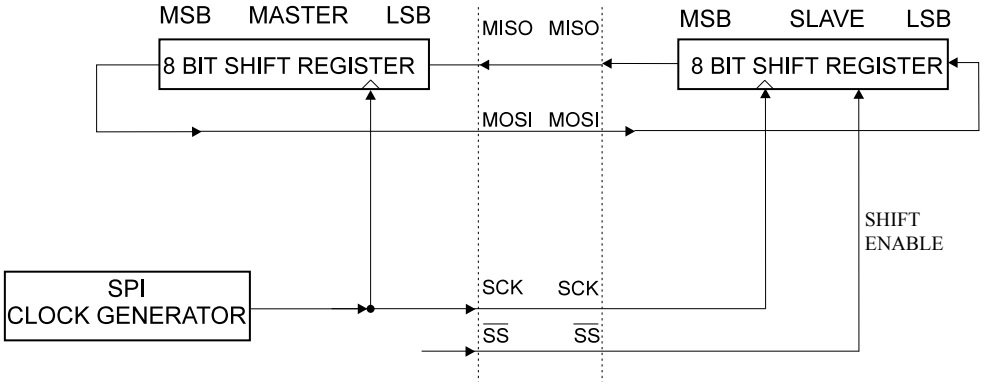
The interconnection between Master and Slave CPUs with SPI is shown in the figure below. The system consists of two shift registers, and a Master Clock generator. The SPI Master initiates the communication cycle when pulling low the Slave Select $\overline{SS}$ pin of the desired Slave. Master and Slave 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 23-2. SPI Master-slave Interconnection**




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

In SPI Slave mode, the control logic will sample the incoming signal of the SCK pin. To ensure correct sampling of the clock signal, the minimum low and high periods should be longer than two CPU clock cycles.

When the SPI is enabled, the data direction of the MOSI, MISO, SCK, and $\overline{SS}$ pins is overridden according to the table below. For more details on automatic port overrides, refer to the IO Port description.

**Table 23-1. 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:** 1. See the IO Port description for how to define the SPI pin directions.

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)))
      ;
}
```

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;
}
```

**Related Links**

Pin Descriptions on page 19
USARTSPI - USART in SPI Mode on page 261
PM - Power Management and Sleep Modes on page 65
I/O-Ports on page 104
About Code Examples on page 25

## 23.3. $\overline{SS}$ Pin Functionality

### 23.3.1. 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. The SPI logic will be reset once the 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.

### 23.3.2. 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.

## 23.4. 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. Data bits are shifted out and latched in on opposite edges of the SCK signal, ensuring sufficient time for data signals to stabilize. The following table, summarizes SPCR.CPOL and SPCR.CPHA settings.

**Table 23-2. 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) |

| SPI Mode | Conditions | Leading Edge | Trailing Edge |
|---|---|---|---|
| 2 | CPOL=1, CPHA=0 | Sample (Falling) | Setup (Rising) |
| 3 | CPOL=1, CPHA=1 | Setup (Falling) | Sample (Rising) |

The SPI data transfer formats are shown in the following figure.

**Figure 23-3. SPI Transfer Format with CPHA = 0**

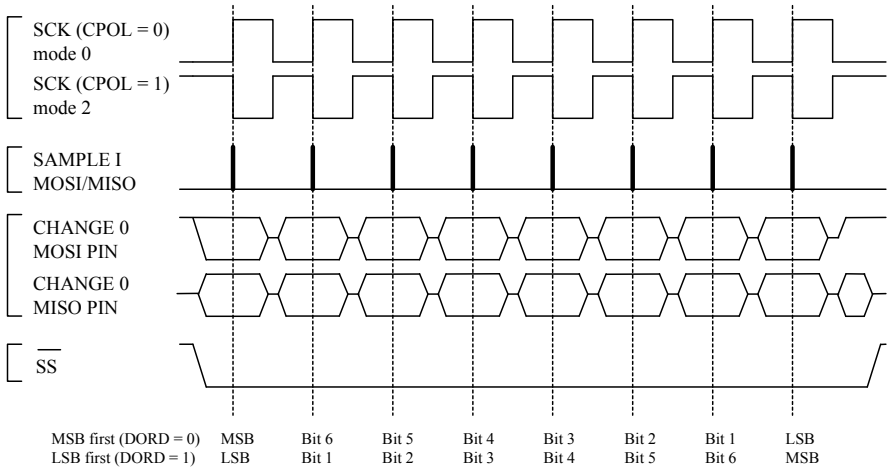
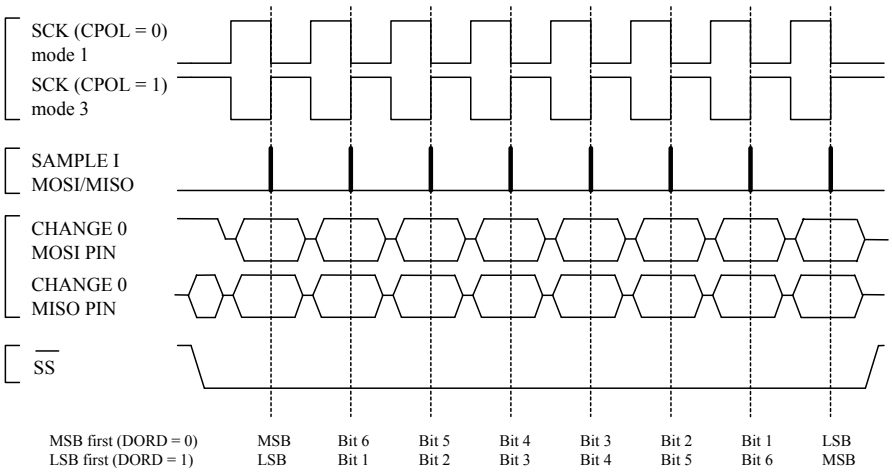



**Figure 23-4. SPI Transfer Format with CPHA = 1**




## 23.5. Register Description

### 23.5.1. SPI Control Register 0

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** SPCR0
**Offset:** 0x4C
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x2C

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | SPIE0 | SPE0 | DORD0 | MSTR0 | CPOL0 | CPHA0 | SPR01 | SPR00 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bit 7 – SPIE0: SPI0 Interrupt Enable**
This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR Register is set and if the Global Interrupt Enable bit in SREG is set.

**Bit 6 – SPE0: SPI0 Enable**
When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

**Bit 5 – DORD0: Data0 Order**
When the DORD bit is written to one, the LSB of the data word is transmitted first.

When the DORD bit is written to zero, the MSB of the data word is transmitted first.

**Bit 4 – MSTR0: Master/Slave0 Select**
This bit selects Master SPI mode when written to one, and Slave SPI mode when written logic zero. If 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 – CPOL0: Clock0 Polarity**
When this bit is written to one, SCK is high when idle. When CPOL is written to zero, SCK is low when idle. Refer to Figure 23-3 and Figure 23-4 for an example. The CPOL functionality is summarized below:

**Table 23-3. CPOL0 Functionality**

| CPOL0 | Leading Edge | Trailing Edge |
|---|---|---|
| 0 | Rising | Falling |
| 1 | Falling | Rising |

**Bit 2 – CPHA0: Clock0 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 23-3 and Figure 23-4 for an example. The CPHA functionality is summarized below:

**Table 23-4. CPHA0 Functionality**

| CPHA0 | Leading Edge | Trailing Edge |
|---|---|---|
| 0 | Sample | Setup |
| 1 | Setup | Sample |

**Bits 1:0 – SPR0n: SPI0 Clock Rate Select n [n = 1:0]**

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 Oscillator Clock frequency $f_{osc}$ is shown in the table below.

**Table 23-5. Relationship between SCK and Oscillator Frequency**

| SPI2X | SPR01 | SPR00 | SCK Frequency |
|---|---|---|---|
| 0 | 0 | 0 | $f_{osc}/4$ |
| 0 | 0 | 1 | $f_{osc}/16$ |
| 0 | 1 | 0 | $f_{osc}/64$ |
| 0 | 1 | 1 | $f_{osc}/128$ |
| 1 | 0 | 0 | $f_{osc}/2$ |
| 1 | 0 | 1 | $f_{osc}/8$ |
| 1 | 1 | 0 | $f_{osc}/32$ |
| 1 | 1 | 1 | $f_{osc}/64$ |

### 23.5.2. SPI Status Register 0

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** SPSR0
**Offset:** 0x4D
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x2D

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | SPIF0 | WCOL0 | | | | | | SPI2X0 |
| Access | R | R | | | | | | R/W |
| Reset | 0 | 0 | | | | | | 0 |

**Bit 7 – SPIF0: SPI Interrupt Flag**

When a serial transfer is complete, the SPIF Flag is set. 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 – WCOL0: Write Collision Flag**

The WCOL 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.

**Bit 0 – SPI2X0: Double SPI Speed Bit**

When this bit is written logic one the SPI speed (SCK Frequency) will be doubled when the SPI is in Master mode (refer to Table 23-5). This means that the minimum SCK period will be two CPU clock periods. When the SPI is configured as Slave, the SPI is only guaranteed to work at fosc/4 or lower.

The SPI interface is also used for program memory and EEPROM downloading or uploading. See *Serial Downloading* for serial programming and verification.

### 23.5.3. SPI Data Register 0

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** SPDR0
**Offset:** 0x4E
**Reset:** 0xXX
**Property:** When addressing as I/O Register: address offset is 0x2E

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | SPID[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | x | x | x | x | x | x | x | x |

**Bits 7:0 – SPID[7:0]: SPI Data**

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.

# 24. USART - Universal Synchronous Asynchronous Receiver Transceiver

## 24.1. 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

## 24.2. Overview

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device.

The USART can also be used in Master SPI mode. The Power Reduction USART bit in the Power Reduction Register (PRR.PRUSARTn) must be written to '0' in order to enable USARTn. USART 0 is in PRR.

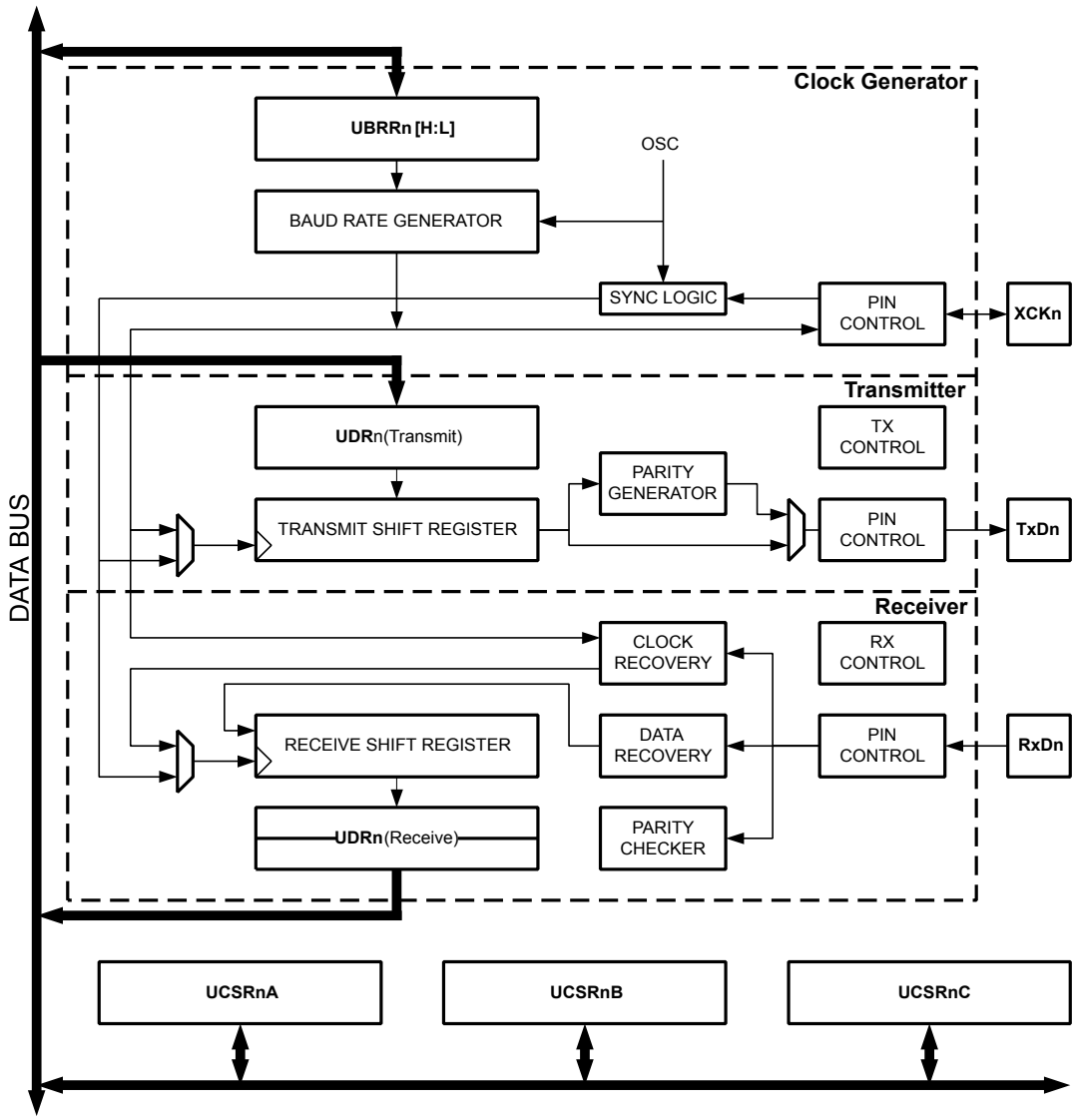
**Related Links**

USARTSPI - USART in SPI Mode on page 261
I/O-Ports on page 104

## 24.3. Block Diagram

In the USART Block Diagram, the CPU accessible I/O Registers and I/O pins are shown in bold. The dashed boxes in the block diagram separate the three main parts of the USART (listed from the top): Clock Generator, Transmitter, and Receiver. Control Registers are shared by all units. The Clock Generation logic consists of synchronization logic for external clock input used by synchronous slave operation, and the baud rate generator. The XCKn (Transfer Clock) pin is only used by synchronous transfer mode. The Transmitter consists of a single write buffer, a serial Shift Register, Parity Generator, and Control logic for handling different serial frame formats. The write buffer allows a continuous transfer of data without any 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 Shift Register, and a two level receive buffer (UDRn). The Receiver supports the same frame formats as the Transmitter, and can detect Frame Error, Data OverRun, and Parity Errors.

**Figure 24-1. USART Block Diagram**



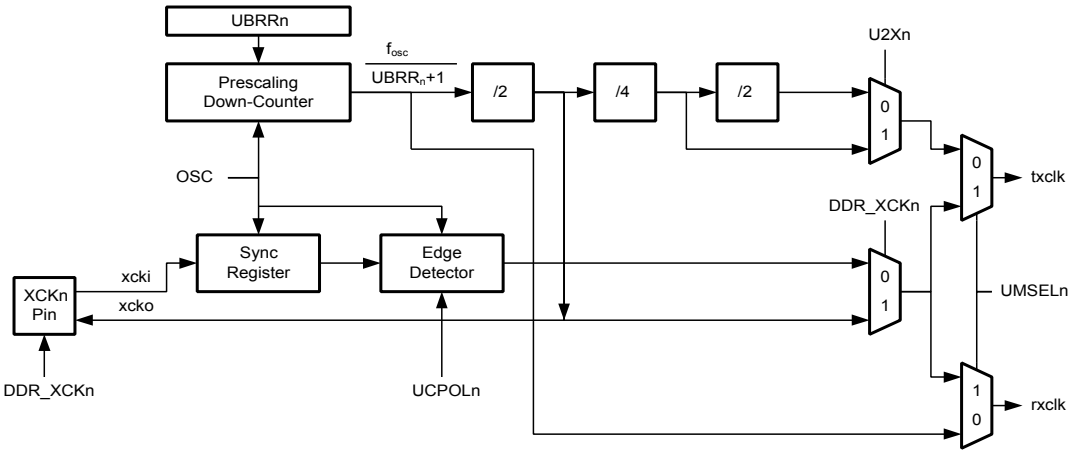

**Note:** Refer to the *Pin Configurations* and the *I/O-Ports* description for USART pin placement.

## 24.4. Clock Generation

The Clock Generation logic generates the base clock for the Transmitter and Receiver. The USART supports four modes of clock operation: Normal asynchronous, Double Speed asynchronous, Master synchronous and Slave synchronous mode. The USART Mode Select bit 0 in the USART Control and Status Register n C (UCSRnC.UMSELn0) selects between asynchronous and synchronous operation. Double Speed (asynchronous mode only) is controlled by the U2Xn found in the UCSRnA Register. When using synchronous mode (UMSELn0=1), the Data Direction Register for the XCKn pin (DDR_XCKn) controls whether the clock source is internal (Master mode) or external (Slave mode). The XCKn pin is only active when using synchronous mode.

Below is a block diagram of the clock generation logic.

**Figure 24-2. Clock Generation Logic, Block Diagram**




Signal description:

- txclk: Transmitter clock (internal signal).
- rxclk: Receiver base clock (internal signal).
- xcki: Input from XCKn pin (internal signal). Used for synchronous slave operation.
- xcko: Clock output to XCKn pin (internal signal). Used for synchronous master operation.
- $f_{osc}$: System clock frequency.

### 24.4.1. Internal Clock Generation – The Baud Rate Generator

Internal clock generation is used for the asynchronous and the synchronous master modes of operation. The description in this section refers to the Clock Generation Logic block diagram in the previous section..

The USART Baud Rate Register (UBRRn) 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 UBRRn value each time the counter has counted down to zero or when the UBRRnL Register is written. A clock is generated each time the counter reaches zero. This clock is the baud rate generator clock output (= $f_{osc}$/(UBRRn+1)). The Transmitter divides the baud rate generator clock output by 2, 8, or 16 depending on mode. 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 the state of the UMSEL, U2Xn and DDR_XCK bits.

The table below contains equations for calculating the baud rate (in bits per second) and for calculating the UBRRn value for each mode of operation using an internally generated clock source.

**Table 24-1. Equations for Calculating Baud Rate Register Setting**

| Operating Mode | Equation for Calculating Baud Rate(1) | Equation for Calculating UBRRn Value |
|---|---|---|
| Asynchronous Normal mode (U2Xn = 0) | $\mathrm{BAUD} = \frac{f_{\mathrm{OSC}}}{16(\mathbf{UBRR}n+1)}$ | $\mathbf{UBRR}n = \frac{f_{\mathrm{OSC}}}{16\mathrm{BAUD}} - 1$ |
| Asynchronous Double Speed mode (U2Xn = 1) | $\mathrm{BAUD} = \frac{f_{\mathrm{OSC}}}{8(\mathbf{UBRR}n+1)}$ | $\mathbf{UBRR}n = \frac{f_{\mathrm{OSC}}}{8\mathrm{BAUD}} - 1$ |
| Synchronous Master mode | $\mathrm{BAUD} = \frac{f_{\mathrm{OSC}}}{2(\mathbf{UBRR}n+1)}$ | $\mathbf{UBRR}n = \frac{f_{\mathrm{OSC}}}{2\mathrm{BAUD}} - 1$ |

**Note:** 1. The baud rate is defined to be 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 the UBRRnH and UBRRnL Registers, (0-4095).
Some examples of UBRRn values for some system clock frequencies are found in Examples of Baud Rate Settings.

### 24.4.2. Double Speed Operation (U2Xn)

The transfer rate can be doubled by setting the U2Xn bit in UCSRnA. Setting this bit only has effect for the asynchronous operation. Set this bit to zero when using synchronous operation.

Setting this bit will reduce the divisor of the baud rate divider from 16 to 8, effectively doubling the transfer rate for asynchronous communication. However, in this case, the Receiver will only use half the number of samples (reduced from 16 to 8) for data sampling and clock recovery, and therefore a more accurate baud rate setting and system clock are required when this mode is used.

For the Transmitter, there are no downsides.

### 24.4.3. External Clock

External clocking is used by the synchronous slave modes of operation. The description in this section refers to the Clock Generation Logic block diagram in the previous section.

External clock input from the XCKn pin is sampled by a synchronization register to minimize the chance of meta-stability. The output from the synchronization register must then pass through an edge detector before it can be used by the Transmitter and Receiver. This process introduces a two CPU clock period delay and therefore the maximum external XCKn clock frequency is limited by the following equation:

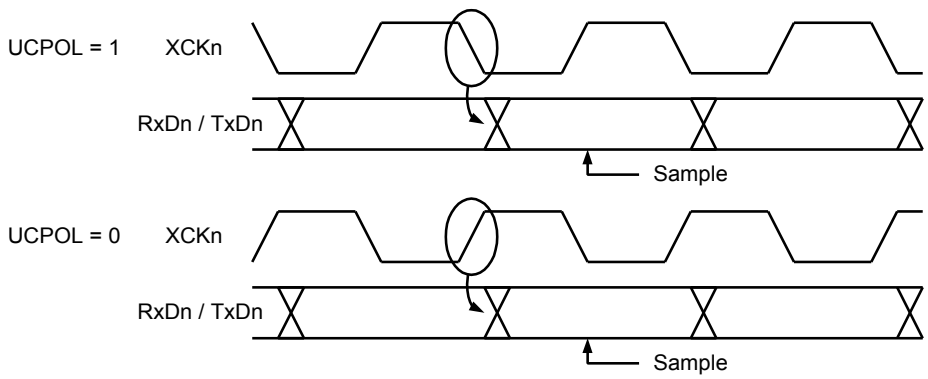
$$f_{XCKn} < \frac{f_{OSC}}{4}$$

The value of $f_{osc}$ depends on the stability of the system clock source. It is therefore recommended to add some margin to avoid possible loss of data due to frequency variations.

### 24.4.4. Synchronous Clock Operation

When synchronous mode is used (UMSEL = 1), the XCKn pin will be used as either clock input (Slave) or clock output (Master). The dependency between the clock edges and data sampling or data change is the same. The basic principle is that data input (on RxDn) is sampled at the opposite XCKn clock edge of the edge the data output (TxDn) is changed.

**Figure 24-3. Synchronous Mode XCKn Timing**




The UCPOL bit UCRSC selects which XCKn clock edge is used for data sampling and which is used for data change. As the above timing diagram shows, when UCPOL is zero, the data will be changed at

rising XCKn edge and sampled at falling XCKn edge. If UCPOL is set, the data will be changed at falling XCKn edge and sampled at rising XCKn edge.

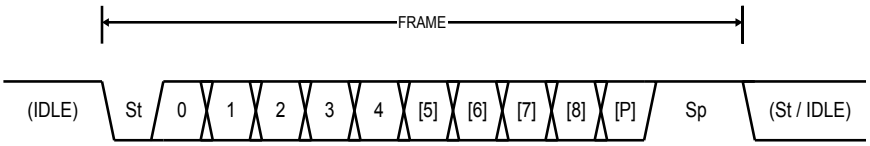
## 24.5. Frame Formats

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:

- 1 start bit
- 5, 6, 7, 8, or 9 data bits
- no, even or odd parity bit
- 1 or 2 stop bits

A frame starts with the start bit, followed by the data bits (from five up to nine data bits in total): first the least significant data bit, then the next data bits ending with the most significant bit. If enabled, the parity bit is inserted after the data bits, before the one or two stop bits. When a complete frame is transmitted, it can be directly followed by a new frame, or the communication line can be set to an idle (high) state. the figure below illustrates the possible combinations of the frame formats. Bits inside brackets are optional.

**Figure 24-4. Frame Formats**




| | |
|---|---|
| St | Start bit, always low. |
| (n) | Data bits (0 to 8). |
| P | Parity bit. Can be odd or even. |
| Sp | Stop bit, always high. |
| IDLE | No transfers on the communication line (RxDn or TxDn). An IDLE line must be high. |

The frame format used by the USART is set by:

- Character Size bits (UCSRnC.UCSZn[2:0]) select the number of data bits in the frame.
- Parity Mode bits (UCSRnC.UPMn[1:0]) enable and set the type of parity bit.
- Stop Bit Select bit (UCSRnC.USBSn) select the number of stop bits. The Receiver ignores the second stop bit.

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. An FE (Frame Error) will only be detected in cases where the first stop bit is zero.

### 24.5.1. 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:

$$P_{\text{even}} = d_{n+-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0 P_{\text{odd}} = d_{n+-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

$P_{even}$ 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 first stop bit of a serial frame.

## 24.6. 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 enabling the Transmitter or the Receiver depending on the usage. For interrupt driven USART operation, the Global Interrupt Flag should be cleared (and interrupts globally disabled) when doing the initialization.

Before doing a re-initialization with changed baud rate or frame format, be sure that there are no ongoing transmissions during the period the registers are changed. The TXC Flag (UCSRnA.TXC) 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. The UCSRnA.TXC must be cleared before each transmission (before UDRn 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 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 the r17, r16 Registers.

**Assembly Code Example**

```
USART_Init:
   ; Set baud rate to UBRR0
   out    UBRR0H, r17
   out    UBRR0L, r16
   ; Enable receiver and transmitter
   ldi    r16, (1<<RXEN0)|(1<<TXEN0)
   out    UCSR0B,r16
   ; Set frame format: 8data, 2stop bit
   ldi    r16, (1<<USBS0)|(3<<UCSZ00)
   out    UCSR0C,r16
   ret
```

**C Code Example**

```
#define FOSC 1843200 // Clock Speed
#define BAUD 9600
#define MYUBRR FOSC/16/BAUD-1
void main( void )
{
   ...
   USART_Init(MYUBRR)
   ...
}
void USART_Init( unsigned int ubrr)
{
   /*Set baud rate */
   UBRR0H = (unsigned char)(ubrr>>8);
   UBRR0L = (unsigned char)ubrr;
   Enable receiver and transmitter */
   UCSR0B = (1<<RXEN0)|(1<<TXEN0);
   /* Set frame format: 8data, 2stop bit */
   UCSR0C = (1<<USBS0)|(3<<UCSZ00);
}
```

More advanced initialization routines can be written to 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.

**Related Links**

About Code Examples on page 25

## 24.7. Data Transmission – The USART Transmitter

The USART Transmitter is enabled by setting the Transmit Enable (TXEN) bit in the UCSRnB Register. 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 XCKn pin will be overridden and used as transmission clock.

### 24.7.1. 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 UDRn I/O location. The buffered data in the transmit buffer will be moved to the Shift Register when the Shift Register 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 Register, U2Xn bit or by XCKn depending on mode of operation.

The following code examples show a simple USART transmit function based on polling of the Data Register Empty (UDRE) Flag. When using frames with less than eight bits, the most significant bits written to the UDR0 are ignored. The USART 0 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 R17.

**Assembly Code Example**

```
USART_Transmit:
   ; Wait for empty transmit buffer
   in      r17, UCSR0A
   sbrs    r17, UDRE
   rjmp    USART_Transmit
   ; Put data (r16) into buffer, sends the data
   out    UDR0,r16
   ret
```

**C Code Example**

```
void USART_Transmit( unsigned char data )
{
   /* Wait for empty transmit buffer */
   while ( !( UCSR0A & (1<<UDRE)) )
      ;
   /* Put data into buffer, sends the data */
   UDR0 = data;
}
```

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.

**Related Links**

About Code Examples on page 25

### 24.7.2. Sending Frames with 9 Data Bit

If 9-bit characters are used (UCSZn = 7), the ninth bit must be written to the TXB8 bit in UCSRnB before the low byte of the character is written to UDRn.

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.

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**

```
USART_Transmit:
   ; Wait for empty transmit buffer
   in      r18, UCSR0A
   sbrs    r18, UDRE
   rjmp    USART_Transmit
   ; Copy 9th bit from r17 to TXB8
   cbi     UCSR0B,TXB8
   sbrc    r17,0
   sbi     UCSR0B,TXB8
   ; Put LSB data (r16) into buffer, sends the data
   out    UDR0,r16
   ret
```

**C Code Example**

```
void USART_Transmit( unsigned int data )
{
   /* Wait for empty transmit buffer */
   while ( !( UCSR0A & (1<<UDRE)) )
      ;
   /* Copy 9th bit to TXB8 */
   UCSR0B &= ~(1<<TXB8);
   if ( data & 0x0100 )
      UCSR0B |= (1<<TXB8);
   /* Put data into buffer, sends the data */
   UDR0 = data;
}
```

**Note:** These transmit functions are written to be general functions. They can be optimized if the contents of the UCSRnB is static. For example, only the TXB8 bit of the UCSRnB Register is used after initialization.

**Related Links**

About Code Examples on page 25

### 24.7.3. 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 (UDRE) Flag 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 the UCSRnA Register.

When the Data Register Empty Interrupt Enable (UDRIE) bit in UCSRnB is written to '1', the USART Data Register Empty Interrupt will be executed as long as UDRE is set (provided that global interrupts are enabled). UDRE is cleared by writing UDRn. When interrupt-driven data transmission is used, the Data Register Empty interrupt routine must either write new data to UDRn 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 (TXC) 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. The TXC Flag bit is either automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a '1' to its bit 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 (TXCIE) bit in UCSRnB is written to '1', the USART Transmit Complete Interrupt will be executed when the TXC Flag becomes set (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, this is done automatically when the interrupt is executed.

### 24.7.4. Parity Generator

The Parity Generator calculates the parity bit for the serial frame data. When parity bit is enabled (UCSRnC.UPM[1]=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.

### 24.7.5. Disabling the Transmitter

When writing the TX Enable bit in the USART Control and Status Register n B (UCSRnB.TXEN) to zero, the disabling of the Transmitter will not become effective until ongoing and pending transmissions are completed, i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When disabled, the Transmitter will no longer override the TxDn pin.

## 24.8. Data Reception – The USART Receiver

The USART Receiver is enabled by writing the Receive Enable (RXEN) bit in the UCSRnB Register to '1'. When the Receiver is enabled, the normal pin operation of the RxDn 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 XCKn pin will be used as transfer clock.

### 24.8.1. 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 XCKn clock, and shifted into the Receive Shift Register until the first stop bit of a frame is received. A second stop bit will be ignored by the Receiver. When the first stop bit is received, i.e., a complete serial frame is present in the Receive Shift Register, the contents of the Shift Register will be moved into the receive buffer. The receive buffer can then be read by reading the UDRn I/O location.

The following code example shows a simple USART receive function based on polling of the Receive Complete (RXC) Flag. When using frames with less than eight bits the most significant bits of the data read from the UDR0 will be masked to zero. The USART 0 has to be initialized before the function can be used. For the assembly code, the received data will be stored in R16 after the code completes.

**Assembly Code Example**

```
USART_Receive:
   ; Wait for data to be received
   in    r17, UCSR0A
   sbrs  r17, RXC
```

```
   rjmp  USART_Receive
   ; Get and return received data from buffer
   in    r16, UDR0
   ret
```

**C Code Example**

```
unsigned char USART_Receive( void )
{
   /* Wait for data to be received */
   while ( !(UCSR0A & (1<<RXC)) )
      ;
   /* Get and return received data from buffer */
   return UDR0;
}
```

For I/O Registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

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.

**Related Links**

About Code Examples on page 25

### 24.8.2. Receiving Frames with 9 Data Bits

If 9-bit characters are used (UCSZn=7) the ninth bit must be read from the RXB8 bit in UCSRnB before reading the low bits from the UDRn. This rule applies to the FE, DOR and UPE Status Flags as well. Read status from UCSRnA, then data from UDRn. Reading the UDRn I/O location will change the state of the receive buffer FIFO and consequently the TXB8, FE, DOR and UPE bits, which all are stored in the FIFO, will change.

The following code example shows a simple receive function for USART0 that handles both nine bit characters and the status bits. For the assembly code, the received data will be stored in R17:R16 after the code completes.

**Assembly Code Example**

```
USART_Receive:
   ; Wait for data to be received
   in     r16, UCSR0A
   sbrs   r16, RXC
   rjmp   USART_Receive
   ; Get status and 9th bit, then data from buffer
   in     r18, UCSR0A
   in     r17, UCSR0B
   in     r16, UDR0
   ; 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**

```
unsigned int USART_Receive( void )
{
```

```
    unsigned char status, resh, resl;
    /* Wait for data to be received */
    while ( !(UCSR0A & (1<<RXC)) )
        ;
    /* Get status and 9th bit, then data */
    /* from buffer */
    status = UCSR0A;
    resh = UCSR0B;
    resl = UDR0;
    /* 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);
}
```

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.

**Related Links**

About Code Examples on page 25

### 24.8.3. Receive Compete Flag and Interrupt

The USART Receiver has one flag that indicates the Receiver state.

The Receive Complete (RXC) Flag indicates if there are unread data present in the receive buffer. This flag is one when unread data exist in the receive buffer, and zero when the receive buffer is empty (i.e., does not contain any unread data). If the Receiver is disabled (RXEN = 0), the receive buffer will be flushed and consequently the RXCn bit will become zero.

When the Receive Complete Interrupt Enable (RXCIE) in UCSRnB is set, the USART Receive Complete interrupt will be executed as long as the RXC Flag is set (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.

### 24.8.4. Receiver Error Flags

The USART Receiver has three Error Flags: Frame Error (FE), Data OverRun (DOR) and Parity Error (UPE). All can be accessed by reading UCSRnA. Common for the Error Flags is that they are located in the receive buffer together with the frame for which they indicate the error status. Due to the buffering of the Error Flags, the UCSRnA must be read before the receive buffer (UDRn), since reading the UDRn I/O location changes the buffer read location. Another equality for the Error Flags is that they can not be altered by software doing a write to the flag location. However, all flags must be set to zero when the UCSRnA is written for upward compatibility of future USART implementations. None of the Error Flags can generate interrupts.

The Frame Error (FE) Flag indicates the state of the first stop bit of the next readable frame stored in the receive buffer. The FE Flag is zero when the stop bit was correctly read as '1', and the FE Flag will be one when the stop bit was incorrect (zero). This flag can be used for detecting out-of-sync conditions, detecting break conditions and protocol handling. The FE Flag is not affected by the setting of the USBS bit in UCSRnC since the Receiver ignores all, except for the first, stop bits. For compatibility with future devices, always set this bit to zero when writing to UCSRnA.

The Data OverRun (DOR) Flag indicates data loss due to a receiver buffer full condition. A Data OverRun occurs when the receive buffer is full (two characters), a new character is waiting in the Receive Shift Register, and a new start bit is detected. If the DOR Flag is set, one or more serial frames were lost between the last frame read from UDR, and the next frame read from UDR. For compatibility with future

devices, always write this bit to zero when writing to UCSRnA. The DOR Flag is cleared when the frame received was successfully moved from the Shift Register to the receive buffer.

The Parity Error (UPE) Flag indicates that the next frame in the receive buffer had a Parity Error when received. If Parity Check is not enabled the UPE bit will always read '0'. For compatibility with future devices, always set this bit to zero when writing to UCSRnA. For more details see Parity Bit Calculation and 'Parity Checker' below.

### 24.8.5. Parity Checker

The Parity Checker is active when the high USART Parity Mode bit 1 in the USART Control and Status Register n C (UCSRnC.UPM[1]) is written to '1'. The type of Parity Check to be performed (odd or even) is selected by the UCSRnC.UPM[0] 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 USART Parity Error Flag in the USART Control and Status Register n A (UCSRnA.UPE) can then be read by software to check if the frame had a Parity Error.

The UPEn bit is set if the next character that can be read from the receive buffer had a Parity Error when received and the Parity Checking was enabled at that point (UPM[1] = 1). This bit is valid until the receive buffer (UDRn) is read.

### 24.8.6. Disabling the Receiver

In contrast to the Transmitter, disabling of the Receiver will be immediate. Data from ongoing receptions will therefore be lost. When disabled (i.e., UCSRnB.RXEN is written to zero) the Receiver will no longer override the normal function of the RxDn port pin. The Receiver buffer FIFO will be flushed when the Receiver is disabled. Remaining data in the buffer will be lost.

### 24.8.7. 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. If the buffer has to be flushed during normal operation, due to for instance an error condition, read the UDRn I/O location until the RXCn Flag is cleared.

The following code shows how to flush the receive buffer of USART0.

**Assembly Code Example**

```
USART_Flush:
   in      r16, UCSR0A
   sbrs    r16, RXC
   ret
   in      r16, UDR0
   rjmp    USART_Flush
```

**C Code Example**

```
void USART_Flush( void )
{
   unsigned char dummy;
   while ( UCSR0A & (1<<RXC) ) dummy = UDR0;
}
```

**Related Links**

About Code Examples on page 25

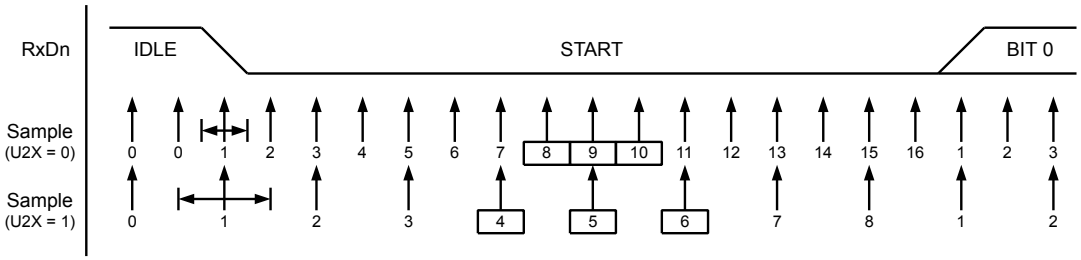
## 24.9. Asynchronous Data Reception

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 RxDn 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.

### 24.9.1. Asynchronous Clock Recovery

The clock recovery logic synchronizes internal clock to the incoming serial frames. The figure below illustrates the sampling process of the start bit of an incoming frame. The sample rate is 16-times the baud rate for Normal mode, and 8 times the baud rate for Double Speed mode. The horizontal arrows illustrate the synchronization variation due to the sampling process. Note the larger time variation when using the Double Speed mode (UCSRnA.U2Xn=1) of operation. Samples denoted '0' are samples taken while the RxDn line is idle (i.e., no communication activity).

**Figure 24-5. Start Bit Sampling**

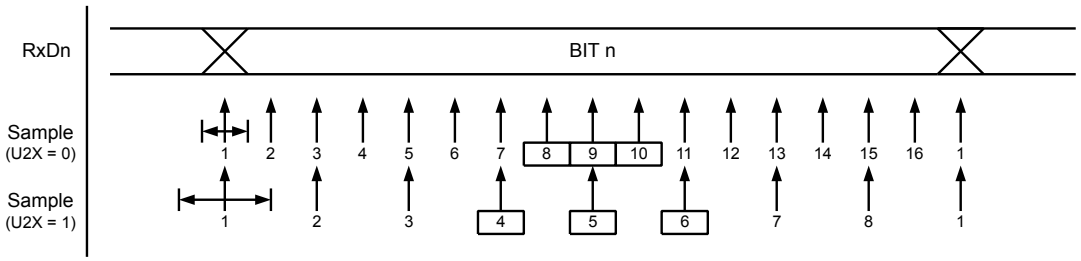



When the clock recovery logic detects a high (idle) to low (start) transition on the RxDn line, the start bit detection sequence is initiated. Let sample 1 denote the first zero-sample as shown in the figure. The clock recovery logic then uses samples 8, 9, and 10 for Normal mode, and samples 4, 5, and 6 for Double Speed mode (indicated with sample numbers inside boxes on the figure), 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 on RxDn. 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.

### 24.9.2. 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. The figure below 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 24-6. Sampling of Data and Parity Bit**

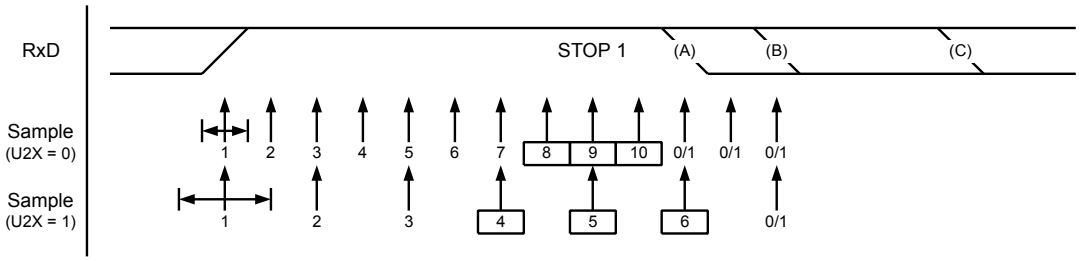



The decision of the logic level of the received bit is taken by doing a majority voting of the logic value to the three samples in the center of the received bit: If two or all three center samples (those marked by

their sample number inside boxes) have high levels, the received bit is registered to be a logic '1'. If two or all three samples have low levels, the received bit is registered to be a logic '0'. This majority voting process acts as a low pass filter for the incoming signal on the RxDn pin. The recovery process is then repeated until a complete frame is received, including the first stop bit. The Receiver only uses the first stop bit of a frame.

The following figure shows the sampling of the stop bit and the earliest possible beginning of the start bit of the next frame.

**Figure 24-7. Stop Bit Sampling and Next Start Bit Sampling**




The same majority voting is done to the stop bit as done for the other bits in the frame. If the stop bit is registered to have a logic '0' value, the Frame Error (UCSRnA.FE) Flag 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. For Normal Speed mode, the first low level sample can be taken at point marked (A) in the figure above. For Double Speed mode, the first low level must be delayed to (B). (C) marks a stop bit of full length. The early start bit detection influences the operational range of the Receiver.

### 24.9.3. Asynchronous Operational Range

The operational range of the Receiver is dependent on the mismatch between the received bit rate and the internally generated baud rate. If 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 recommendations below), 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.

| | |
|---|---|
| $R_{\text{slow}} = \frac{(D+1)S}{S-1+D\cdot S+S_F}$ | $R_{\text{fast}} = \frac{(D+2)S}{(D+1)S+S_M}$ |

- $D$: Sum of character size and parity size (D = 5 to 10 bit)
- $S$: Samples per bit. S = 16 for Normal Speed mode and S = 8 for Double Speed mode.
- $S_F$: First sample number used for majority voting. $S_F$ = 8 for normal speed and $S_F$ = 4 for Double Speed mode.
- $S_M$: Middle sample number used for majority voting. $S_M$ = 9 for normal speed and $S_M$ = 5 for Double Speed mode.
- $R_{slow}$ : is the ratio of the slowest incoming data rate that can be accepted in relation to the receiver baud rate. $R_{fast}$ is the ratio of the fastest incoming data rate that can be accepted in relation to the receiver baud rate.

The following tables list the maximum receiver baud rate error that can be tolerated. Note that Normal Speed mode has higher toleration of baud rate variations.

**Table 24-2. Recommended Maximum Receiver Baud Rate Error for Normal Speed Mode (U2Xn = 0)**

| 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 24-3. Recommended Maximum Receiver Baud Rate Error for Double Speed Mode (U2Xn = 1)**

| 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 was made under the assumption that the Receiver and Transmitter equally divides the maximum total error.

There are two possible sources for the receivers baud rate error. The Receiver's system clock (EXTCLK) will always have some minor instability over the supply voltage range and the temperature range. When using a crystal to generate the system clock, this is rarely a problem, but for a resonator, the system clock may differ more than 2% depending of the resonator's tolerance. The second source for the 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 UBRRn value that gives an acceptable low error can be used if possible.

## 24.10. Multi-Processor Communication Mode

Setting the Multi-Processor Communication mode (MPCMn) bit in UCSRnA 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. This effectively reduces the number of incoming frames that has to be handled by the CPU, in a system with multiple MCUs that communicate via the same serial bus. The Transmitter is unaffected by the MPCMn setting, 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 9 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 '1', the frame contains an address. When the frame type bit is '0', 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.

### 24.10.1. Using MPCMn

For an MCU to act as a master MCU, it can use a 9-bit character frame format (UCSZ1=7). The ninth bit (TXB8) must be set when an address frame (TXB8=1) or cleared when a data frame (TXB=0) is being transmitted. The slave MCUs must in this case 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 in Multi-Processor Communication mode (MPCM in UCSRnA is set).
2. The Master MCU sends an address frame, and all slaves receive and read this frame. In the Slave MCUs, the RXC Flag in UCSRnA will be set as normal.
3. Each Slave MCU reads the UDRn Register and determines if it has been selected. If so, it clears the MPCM bit in UCSRnA, otherwise 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, the addressed MCU sets the MPCM bit and waits for a new address frame from master. The process then repeats from step 2.

Using any of the 5- to 8-bit character frame formats is possible, 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 uses the same character size setting. If 5- to 8-bit character frames are used, the Transmitter must be set to use two stop bit (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.

## 24.11. Examples of Baud Rate Setting

For standard crystal and resonator frequencies, the most commonly used baud rates for asynchronous operation can be generated by using the UBRRn settings as listed in the table below.

UBRRn values which yield an actual baud rate differing less than 0.5% from the target baud rate, are bold in the table. 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 also section *Asynchronous Operational Range*). The error values are calculated using the following equation:

$$Error\left[\%\right] = \left(\frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}} - 1\right)^2 \ 100\,\%$$

**Table 24-4. Examples of UBRRn 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 | |
| | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | 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) | 62.5kbps | | 125kbps | | 115.2kbps | | 230.4kbps | | 125kbps | | 250kbps | |

Note: 1. UBRRn = 0, Error = 0.0%

**Table 24-5. Examples of UBRRn 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 | |
| | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | 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% |
| 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% |

| 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 | |
| | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error |
| 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) | 230.4kbps | | 460.8kbps | | 250kbps | | 0.5Mbps | | 460.8kbps | | 921.6kbps | |

(1) UBRRn = 0, Error = 0.0%

**Table 24-6. Examples of UBRRn 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 | |
| | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | 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% |
| 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) | 0.5Mbps | | 1Mbps | | 691.2kbps | | 1.3824Mbps | | 921.6kbps | | 1.8432Mbps | |

(1) UBRRn = 0, Error = 0.0%

**Table 24-7. Examples of UBRRn 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 | |
| | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | 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) | 1Mbps | | 2Mbps | | 1.152Mbps | | 2.304Mbps | | 1.25Mbps | | 2.5Mbps | |

(1) UBRRn = 0, Error = 0.0%

**Related Links**

Asynchronous Operational Range on page 245

## 24.12. Register Description

### 24.12.1. USART I/O Data Register 0

The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDR0. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDR0 Register location. Reading the UDR0 Register location will return 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 UDRE0 Flag in the UCSR0A Register is set. Data written to UDR0 when the UDRE0 Flag is not set, will be ignored by the USART Transmitter. When data is written to the transmit buffer, and the Transmitter is enabled, the Transmitter will load the data into the Transmit Shift Register when the Shift Register is empty. Then the data will be serially transmitted on the TxD0 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, do not use Read-Modify-Write instructions (SBI and CBI) on this location. Be careful when using bit test instructions (SBIC and SBIS), since these also will change the state of the FIFO.

**Name:** UDR0
**Offset:** 0xC6
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TXB / RXB[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – TXB / RXB[7:0]: USART Transmit / Receive Data Buffer**

### 24.12.2. USART Control and Status Register 0 A

**Name:** UCSR0A
**Offset:** 0xC0
**Reset:** 0x20
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RXC0 | TXC0 | UDRE0 | FE0 | DOR0 | UPE0 | U2X0 | MPCM0 |
| Access | R | R/W | R | R | R | R | R/W | R/W |
| Reset | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

**Bit 7 – RXC0: 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 RXC0 bit will become zero. The RXC0 Flag can be used to generate a Receive Complete interrupt (see description of the RXCIE0 bit).

**Bit 6 – TXC0: 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 (UDR0). The TXC0 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 TXC0 Flag can generate a Transmit Complete interrupt (see description of the TXCIE0 bit).

**Bit 5 – UDRE0: USART Data Register Empty**
The UDRE0 Flag indicates if the transmit buffer (UDR0) is ready to receive new data. If UDRE0 is one, the buffer is empty, and therefore ready to be written. The UDRE0 Flag can generate a Data Register Empty interrupt (see description of the UDRIE0 bit). UDRE0 is set after a reset to indicate that the Transmitter is ready.

**Bit 4 – FE0: 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 (UDR0) is read. The FEn bit is zero when the stop bit of received data is one. Always set this bit to zero when writing to UCSR0A.

This bit is reserved in Master SPI Mode (MSPIM).

**Bit 3 – DOR0: 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), it 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 (UDR0) is read. Always set this bit to zero when writing to UCSR0A.

This bit is reserved in Master SPI Mode (MSPIM).

**Bit 2 – UPE0: 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 (UPM01 = 1). This bit is valid until the receive buffer (UDR0) is read. Always set this bit to zero when writing to UCSR0A.

This bit is reserved in Master SPI Mode (MSPIM).

**Bit 1 – U2X0: 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.

This bit is reserved in Master SPI Mode (MSPIM).

**Bit 0 – MPCM0: Multi-processor Communication Mode**
This bit enables the Multi-processor Communication mode. When the MPCMn 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 MPCM0 setting. Refer to Multi-Processor Communication Mode for details.

This bit is reserved in Master SPI Mode (MSPIM).

### 24.12.3. USART Control and Status Register 0 B

**Name:** UCSR0B
**Offset:** 0xC1
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RXCIE0 | TXCIE0 | UDRIE0 | RXEN0 | TXEN0 | UCSZ02 | RXB80 | TXB80 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bit 7 – RXCIE0: RX Complete Interrupt Enable 0**
Writing this bit to one enables interrupt on the RXC0 Flag. A USART Receive Complete interrupt will be generated only if the RXCIE0 bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXC0 bit in UCSR0A is set.

**Bit 6 – TXCIE0: TX Complete Interrupt Enable 0**
Writing this bit to one enables interrupt on the TXC0 Flag. A USART Transmit Complete interrupt will be generated only if the TXCIE0 bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXC0 bit in UCSR0A is set.

**Bit 5 – UDRIE0: USART Data Register Empty Interrupt Enable 0**
Writing this bit to one enables interrupt on the UDRE0 Flag. A Data Register Empty interrupt will be generated only if the UDRIE0 bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDRE0 bit in UCSR0A is set.

**Bit 4 – RXEN0: Receiver Enable 0**
Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxDn pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FE0, DOR0, and UPE0 Flags.

**Bit 3 – TXEN0: Transmitter Enable 0**
Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxD0 pin when enabled. The disabling of the Transmitter (writing TXEN0 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 TxD0 port.

**Bit 2 – UCSZ02: Character Size 0**
The UCSZ02 bits combined with the UCSZ0[1:0] bit in UCSR0C sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use.

This bit is reserved in Master SPI Mode (MSPIM).

**Bit 1 – RXB80: Receive Data Bit 8 0**
RXB80 is the ninth data bit of the received character when operating with serial frames with nine data bits. Must be read before reading the low bits from UDR0.

This bit is reserved in Master SPI Mode (MSPIM).

**Bit 0 – TXB80: Transmit Data Bit 8 0**

TXB80 is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. Must be written before writing the low bits to UDR0.

This bit is reserved in Master SPI Mode (MSPIM).

### 24.12.4. USART Control and Status Register 0 C

**Name:** UCSR0C
**Offset:** 0xC2
**Reset:** 0x06
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | UMSEL01 | UMSEL00 | UPM01 | UPM00 | USBS0 | UCSZ01 / UDORD0 | UCSZ00 / UCPHA0 | UCPOL0 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

**Bits 7:6 – UMSEL0n: USART Mode Select 0 n [n = 1:0]**
These bits select the mode of operation of the USART0

**Table 24-8. USART Mode Selection**

| UMSEL0[1:0] | Mode |
|---|---|
| 00 | Asynchronous USART |
| 01 | Synchronous USART |
| 10 | Reserved |
| 11 | Master SPI (MSPIM)(1) |

**Note:**
1. The UDORD0, UCPHA0, and UCPOL0 can be set in the same write operation where the MSPIM is enabled.

**Bits 5:4 – UPM0n: USART Parity Mode 0 n [n = 1:0]**
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 UPM0 setting. If a mismatch is detected, the UPE0 Flag in UCSR0A will be set.

**Table 24-9. USART Mode Selection**

| UPM0[1:0] | ParityMode |
|---|---|
| 00 | Disabled |
| 01 | Reserved |
| 10 | Enabled, Even Parity |
| 11 | Enabled, Odd Parity |

These bits are reserved in Master SPI Mode (MSPIM).

**Bit 3 – USBS0: USART Stop Bit Select 0**
This bit selects the number of stop bits to be inserted by the Transmitter. The Receiver ignores this setting.

**Table 24-10. Stop Bit Settings**

| USBS0 | Stop Bit(s) |
|---|---|
| 0 | 1-bit |
| 1 | 2-bit |

This bit is reserved in Master SPI Mode (MSPIM).

**Bit 2 – UCSZ01 / UDORD0: USART Character Size / Data Order**
**UCSZ0[1:0]: USART Modes:** The UCSZ0[1:0] bits combined with the UCSZ02 bit in UCSR0B sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use.

**Table 24-11. Character Size Settings**

| UCSZ0[2:0] | Character Size |
|---|---|
| 000 | 5-bit |
| 001 | 6-bit |
| 010 | 7-bit |
| 011 | 8-bit |
| 100 | Reserved |
| 101 | Reserved |
| 110 | Reserved |
| 111 | 9-bit |

**UDPRD0: Master SPI Mode:** 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. Refer to the *USART in SPI Mode - Frame Formats* for details.

**Bit 1 – UCSZ00 / UCPHA0: USART Character Size / Clock Phase**
**UCSZ00: USART Modes:** Refer to UCSZ01.

**UCPHA0: Master SPI Mode:** The UCPHA0 bit setting determine if data is sampled on the leasing edge (first) or tailing (last) edge of XCK0. Refer to the *SPI Data Modes and Timing* for details.

**Bit 0 – UCPOL0: Clock Polarity 0**
**USART0 Modes:** This bit is used for synchronous mode only. Write this bit to zero when asynchronous mode is used. The UCPOL0 bit sets the relationship between data output change and data input sample, and the synchronous clock (XCK0).

**Table 24-12. USART Clock Polarity Settings**

| UCPOL0 | Transmitted Data Changed (Output of TxD0 Pin) | Received Data Sampled (Input on RxD0 Pin) |
|---|---|---|
| 0 | Rising XCK0 Edge | Falling XCK0 Edge |
| 1 | Falling XCK0 Edge | Rising XCK0 Edge |

**Master SPI Mode:** The UCPOL0 bit sets the polarity of the XCK0 clock. The combination of the UCPOL0 and UCPHA0 bit settings determine the timing of the data transfer. Refer to the *SPI Data Modes and Timing* for details.

### 24.12.5. USART Baud Rate 0 Register Low

**Name:** UBRR0L
**Offset:** 0xC4
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | UBRR0[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – UBRR0[7:0]: USART Baud Rate 0**
This is a 12-bit register which contains the USART baud rate. The UBRR0H contains the four most significant bits and the UBRR0L 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 UBRR0L will trigger an immediate update of the baud rate prescaler.

### 24.12.6. USART Baud Rate 0 Register High

**Name:** UBRR0H
**Offset:** 0xC5
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | UBRR0[3:0] | | | |
| Access | | | | | R/W | R/W | R/W | R/W |
| Reset | | | | | 0 | 0 | 0 | 0 |

**Bits 3:0 – UBRR0[3:0]: USART Baud Rate 0 n [n = 11:8]**
Refer to UBRR0L.

# 25. USARTSPI - USART in SPI Mode

## 25.1. 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 ($f_{XCKmax} = f_{CK}/2$)
- Flexible Interrupt Generation

## 25.2. Overview

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) can be set to a master SPI compliant mode of operation.

Setting both UMSELn[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.

## 25.3. 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. The Data Direction Register for the XCKn pin (DDR_XCKn) must therefore be set to one (i.e. as output) for the USART in MSPIM to operate correctly. Preferably the DDR_XCKn should be set up before the USART in MSPIM is enabled (i.e. TXENn and RXENn bit set to one).

The internal clock generation used in MSPIM mode is identical to the USART synchronous master mode. The table below contains the equations for calculating the baud rate or UBRRn setting for Synchronous Master Mode.

**Table 25-1. Equations for Calculating Baud Rate Register Setting**

| Operating Mode | Equation for Calculating Baud Rate[1] | Equation for Calculating UBRRn Value |
|---|---|---|
| Synchronous Master mode | $\mathrm{BAUD} = \frac{f_{\mathrm{OSC}}}{2(\mathbf{UBRR}n + 1)}$ | $\mathbf{UBRR}n = \frac{f_{\mathrm{OSC}}}{2\mathrm{BAUD}} - 1$ |

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps)

| | |
|---|---|
| BAUD | Baud rate (in bits per second, bps) |
| $f_{OSC}$ | System Oscillator clock frequency |
| UBRRn | Contents of the UBRRnH and UBRRnL Registers, (0-4095) |

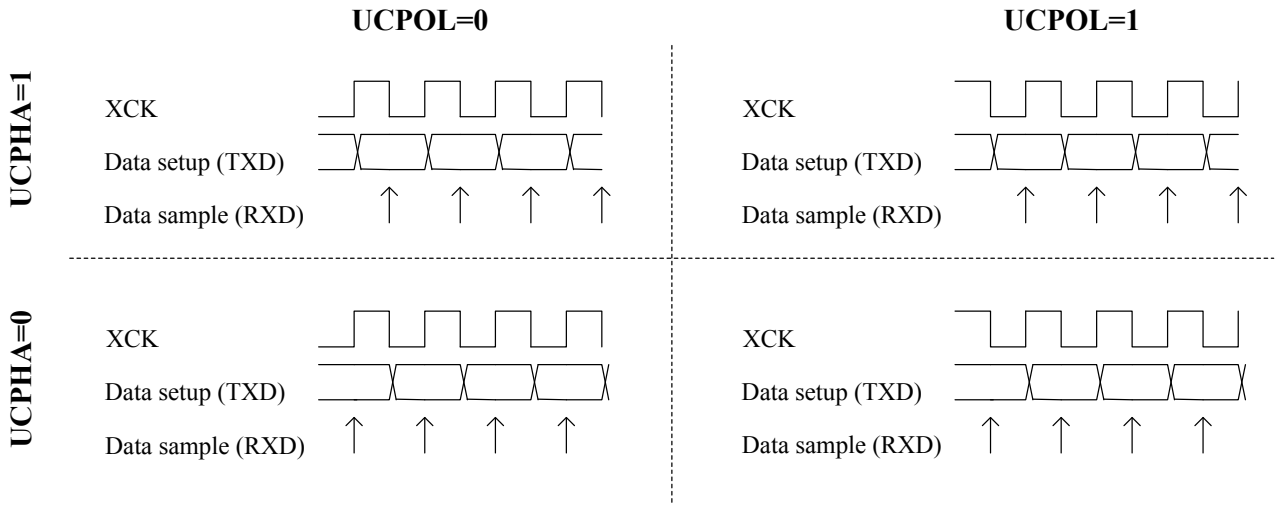
## 25.4. SPI Data Modes and Timing

There are four combinations of XCKn (SCK) phase and polarity with respect to serial data, which are determined by control bits UCPHAn and UCPOLn. The data transfer timing diagrams are shown in the following figure. Data bits are shifted out and latched in on opposite edges of the XCKn signal, ensuring sufficient time for data signals to stabilize. The UCPOLn and UCPHAn functionality is summarized in the following table. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the Receiver and Transmitter.

**Table 25-2. UCPOLn and UCPHAn Functionality**

| UCPOLn | UCPHAn | 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 25-1. UCPHAn and UCPOLn data transfer timing diagrams.**




## 25.5. Frame Formats

A serial frame for the MSPIM is defined to be one character of eight 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 the next data bits, up to a total of eight, are succeeding, 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 UDORDn bit in UCSRnC 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 UDRn. A UART transmit complete interrupt will then signal that the 16-bit value has been shifted out.

### 25.5.1. 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 (by setting DDR_XCKn to one), 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 XCKn output the baud-rate register (UBRRn) must be zero at the time the transmitter is enabled. Contrary to the normal mode USART operation the UBRRn must then be written to the desired value after the transmitter is enabled, but before the first transmission is started. Setting UBRRn to zero before enabling the transmitter is not necessary if the initialization is done immediately after a reset since UBRRn 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 TXCn Flag can be used to check that the Transmitter has completed all transfers, and the RXCn Flag can be used to check that there are no unread data in the receive buffer. Note that the TXCn Flag must be cleared before each transmission (before UDRn 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 the r17:r16 registers.

**Assembly Code Example**

```
clr r18
out UBRRnH,r18
out UBRRnL,r18
; Setting the XCKn port pin as output, enables master mode.
sbi XCKn_DDR, XCKn
; Set MSPI mode of operation and SPI data mode 0.
ldi r18, (1<<UMSELn1)|(1<<UMSELn0)|(0<<UCPHAn)|(0<<UCPOLn)
out UCSRnC,r18
; Enable receiver and transmitter.
ldi r18, (1<<RXENn)|(1<<TXENn)
out UCSRnB,r18
; Set baud rate.
; IMPORTANT: The Baud Rate must be set after the transmitter is enabled!
out UBRRnH, r17
out UBRRnL, r18
ret
```

**C Code Example**

```
{
 UBRRn = 0;
 /* Setting the XCKn port pin as output, enables master mode. */
 XCKn_DDR |= (1<<XCKn);
 /* Set MSPI mode of operation and SPI data mode 0. */
 UCSRnC = (1<<UMSELn1)|(1<<UMSELn0)|(0<<UCPHAn)|(0<<UCPOLn);
 /* Enable receiver and transmitter. */
 UCSRnB = (1<<RXENn)|(1<<TXENn);
 /* Set baud rate. */
```

```
 /* IMPORTANT: The Baud Rate must be set after the transmitter is enabled */
 UBRRn = baud;
}
```

**Related Links**

About Code Examples on page 25

## 25.6. Data Transfer

Using the USART in MSPI mode requires the Transmitter to be enabled, i.e. the TXENn bit in the UCSRnB register is set to one. When the Transmitter is enabled, the normal port operation of the TxDn pin is overridden and given the function as the Transmitter's serial output. Enabling the receiver is optional and is done by setting the RXENn bit in the UCSRnB register to one. When the receiver is enabled, the normal pin operation of the RxDn pin is overridden and given the function as the Receiver's serial input. The XCKn 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 the UDRn I/O location. This is the case for both sending and receiving data since the transmitter controls the transfer clock. The data written to UDRn 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, the UDRn register 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 UDRn 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 (UDREn) Flag and the Receive Complete (RXCn) Flag. 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 UDREn 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 RXCn Flag, before reading the buffer and returning the value.

**Assembly Code Example**

```
USART_MSPIM_Transfer:
 ; Wait for empty transmit buffer
 in r16, UCSRnA
 sbrs r16, UDREn
 rjmp USART_MSPIM_Transfer
 ; Put data (r16) into buffer, sends the data
 out UDRn,r16
 ; Wait for data to be received
USART_MSPIM_Wait_RXCn:
 in r16, UCSRnA
 sbrs r16, RXCn
 rjmp USART_MSPIM_Wait_RXCn
 ; Get and return received data from buffer
 in r16, UDRn
 ret
```

**C Code Example**

```
{
 /* Wait for empty transmit buffer */
 while ( !( UCSRnA & (1<<UDREn)) );
 /* Put data into buffer, sends the data */
 UDRn = data;
 /* Wait for data to be received */
 while ( !(UCSRnA & (1<<RXCn)) );
 /* Get and return received data from buffer */
 return UDRn;
}
```

**Related Links**

About Code Examples on page 25

### 25.6.1. Transmitter and Receiver Flags and Interrupts

The RXCn, TXCn, and UDREn 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 is always read as zero.

### 25.6.2. 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.

## 25.7. AVR USART MSPIM vs. AVR SPI

The USART in MSPIM mode is fully compatible with the AVR SPI regarding:

- Master mode timing diagram
- The UCPOLn bit functionality is identical to the SPI CPOL bit
- The UCPHAn bit functionality is identical to the SPI CPHA bit
- The UDORDn 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 UBRRn 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 the table below.

**Table 25-3. Comparison of USART in MSPIM mode and SPI pins**

| USART_MSPIM | SPI | Comments |
|---|---|---|
| TxDn | MOSI | Master Out only |
| RxDn | MISO | Master In only |

| USART_MSPIM | SPI | Comments |
|---|---|---|
| XCKn | SCK | (Functionally identical) |
| (N/A) | $\overline{SS}$ | Not supported by USART in MSPIM |

## 25.8. Register Description

Refer to the USART Register Description.

**Related Links**

Register Description on page 250

# 26. TWI - 2-wire Serial Interface

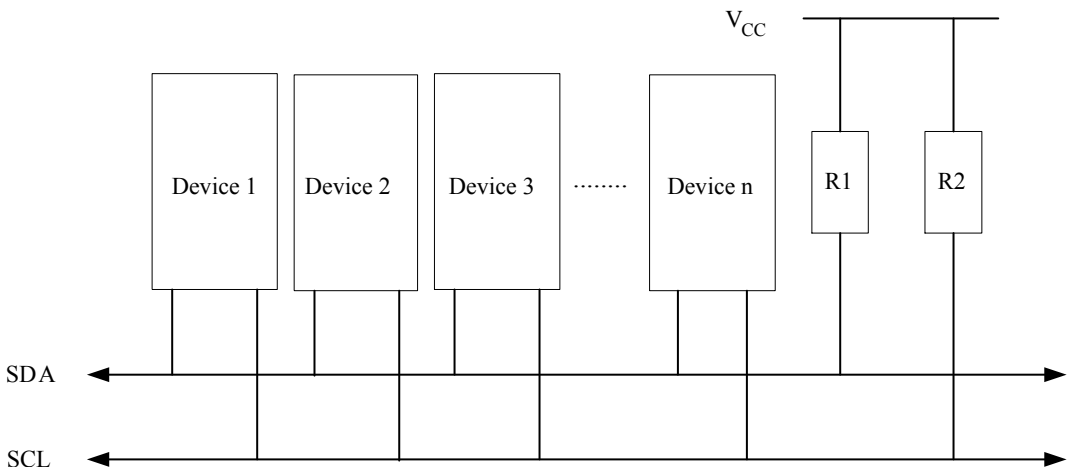
## 26.1. Features

- Simple, yet Powerful and Flexible Communication Interface, only two Bus Lines Needed
- Both Master and Slave Operation Supported
- Device can Operate as Transmitter or Receiver
- 7-bit Address Space Allows up to 128 Different Slave Addresses
- Multi-master Arbitration Support
- Up to 400kHz Data Transfer Speed
- Slew-rate Limited Output Drivers
- Noise Suppression Circuitry Rejects Spikes on Bus Lines
- Fully Programmable Slave Address with General Call Support
- Address Recognition Causes Wake-up When AVR is in Sleep Mode
- Compatible with Philips' $I^2C$ protocol

## 26.2. Two-Wire Serial Interface Bus Definition

The Two-Wire Serial Interface (TWI) is ideally suited for typical microcontroller applications. The TWI protocol allows the systems designer to interconnect up to 128 different devices using only two bi-directional bus lines: one for clock (SCL) and one for data (SDA). The only external hardware needed to implement the bus is a single pull-up resistor for each of the TWI bus lines. All devices connected to the bus have individual addresses, and mechanisms for resolving bus contention are inherent in the TWI protocol.

**Figure 26-1. TWI Bus Interconnection**




### 26.2.1. TWI Terminology

The following definitions are frequently encountered in this section.

**Table 26-1. TWI Terminology**

| Term | Description |
|---|---|
| Master | The device that initiates and terminates a transmission. The Master also generates the SCL clock. |
| Slave | The device addressed by a Master. |
| Transmitter | The device placing data on the bus. |
| Receiver | The device reading data from the bus. |

This device has one instance of TWI. For this reason, the instance index *n* is omitted.

The Power Reduction TWI bit in the Power Reduction Register (PRRn.PRTWI) must be written to '0' to enable the two-wire Serial Interface.

TWI0 is in PRR.

**Related Links**

Power Management and Sleep Modes on page 65

### 26.2.2. Electrical Interconnection

As depicted in the TWI Bus Definition, both bus lines are connected to the positive supply voltage through pull-up resistors. The bus drivers of all TWI-compliant devices are open-drain or open-collector. This implements a wired-AND function which is essential to the operation of the interface. A low level on a TWI bus line is generated when one or more TWI devices output a zero. A high level is output when all TWI devices tri-state their outputs, allowing the pull-up resistors to pull the line high. Note that all AVR devices connected to the TWI bus must be powered in order to allow any bus operation.

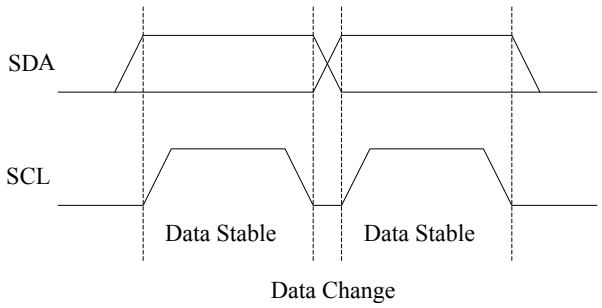
The number of devices that can be connected to the bus is only limited by the bus capacitance limit of 400pF and the 7-bit slave address space. Two different sets of specifications are presented there, one relevant for bus speeds below 100kHz, and one valid for bus speeds up to 400kHz.

## 26.3. Data Transfer and Frame Format

### 26.3.1. Transferring Bits

Each data bit transferred on the TWI bus is accompanied by a pulse on the clock line. The level of the data line must be stable when the clock line is high. The only exception to this rule is for generating start and stop conditions.

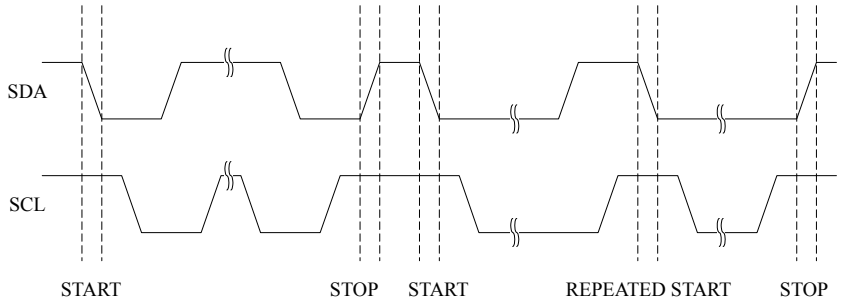
**Figure 26-2. Data Validity**



### 26.3.2. START and STOP Conditions

The Master initiates and terminates a data transmission. The transmission is initiated when the Master issues a START condition on the bus, and it is terminated when the Master issues a STOP condition. Between a START and a STOP condition, the bus is considered busy, and no other master should try to seize control of the bus. A special case occurs when a new START condition is issued between a START and STOP condition. This is referred to as a REPEATED START condition, and is used when the Master wishes to initiate a new transfer without relinquishing control of the bus. After a REPEATED START, the bus is considered busy until the next STOP. This is identical to the START behavior, and therefore START is used to describe both START and REPEATED START for the remainder of this datasheet, unless otherwise noted. As depicted below, START and STOP conditions are signalled by changing the level of the SDA line when the SCL line is high.

**Figure 26-3. START, REPEATED START and STOP conditions**




### 26.3.3. Address Packet Format

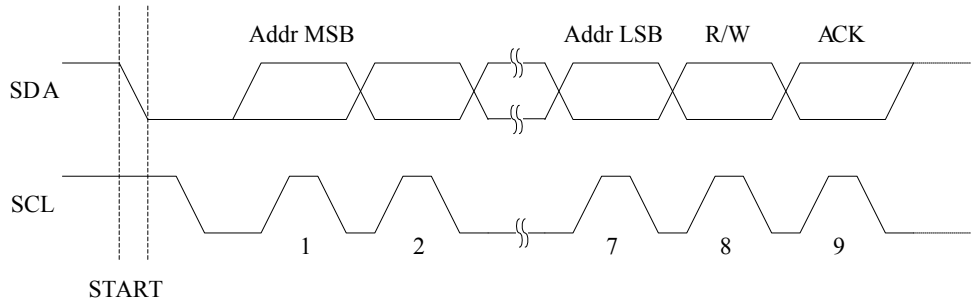
All address packets transmitted on the TWI bus are 9 bits long, consisting of 7 address bits, one READ/WRITE control bit and an acknowledge bit. If the READ/WRITE bit is set, a read operation is to be performed, otherwise a write operation should be performed. When a Slave recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle. If the addressed Slave is busy, or for some other reason can not service the Master's request, the SDA line should be left high in the ACK clock cycle. The Master can then transmit a STOP condition, or a REPEATED START condition to initiate a new transmission. An address packet consisting of a slave address and a READ or a WRITE bit is called SLA+R or SLA+W, respectively.

The MSB of the address byte is transmitted first. Slave addresses can freely be allocated by the designer, but the address '0000 000' is reserved for a general call.

When a general call is issued, all slaves should respond by pulling the SDA line low in the ACK cycle. A general call is used when a Master wishes to transmit the same message to several slaves in the system. When the general call address followed by a Write bit is transmitted on the bus, all slaves set up to acknowledge the general call will pull the SDA line low in the ACK cycle. The following data packets will then be received by all the slaves that acknowledged the general call. Note that transmitting the general call address followed by a Read bit is meaningless, as this would cause contention if several slaves started transmitting different data.

All addresses of the format '1111 xxx' should be reserved for future purposes.

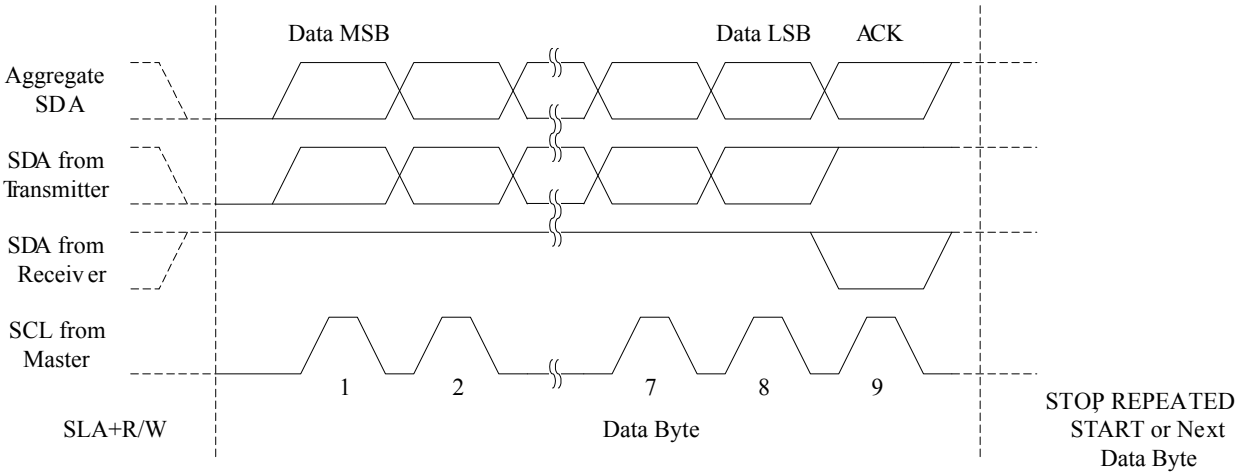
**Figure 26-4. Address Packet Format**




### 26.3.4. Data Packet Format

All data packets transmitted on the TWI bus are nine bits long, consisting of one data byte and an acknowledge bit. During a data transfer, the Master generates the clock and the START and STOP conditions, while the Receiver is responsible for acknowledging the reception. An Acknowledge (ACK) is signalled by the Receiver pulling the SDA line low during the ninth SCL cycle. If the Receiver leaves the SDA line high, a NACK is signalled. When the Receiver has received the last byte, or for some reason cannot receive any more bytes, it should inform the Transmitter by sending a NACK after the final byte. The MSB of the data byte is transmitted first.

**Figure 26-5. Data Packet Format**

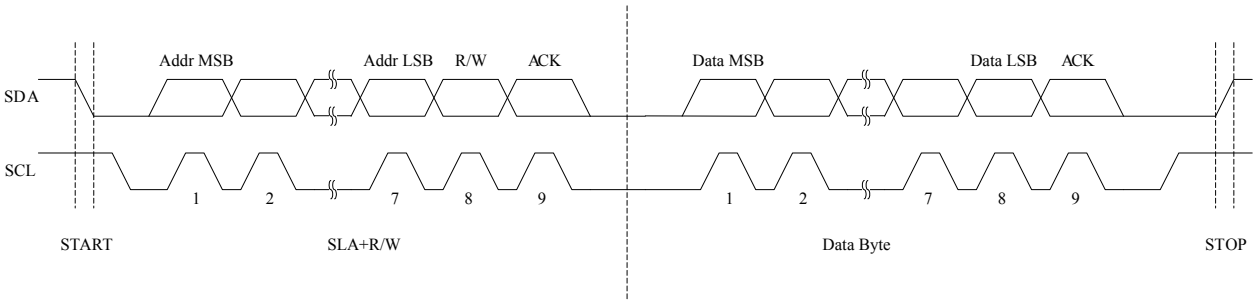



### 26.3.5. Combining Address and Data Packets into a Transmission

A transmission basically consists of a START condition, a SLA+R/W, one or more data packets and a STOP condition. An empty message, consisting of a START followed by a STOP condition, is illegal. Note that the "Wired-ANDing" of the SCL line can be used to implement handshaking between the Master and the Slave. The Slave can extend the SCL low period by pulling the SCL line low. This is useful if the clock speed set up by the Master is too fast for the Slave, or the Slave needs extra time for processing between the data transmissions. The Slave extending the SCL low period will not affect the SCL high period, which is determined by the Master. As a consequence, the Slave can reduce the TWI data transfer speed by prolonging the SCL duty cycle.

The following figure depicts a typical data transmission. Note that several data bytes can be transmitted between the SLA+R/W and the STOP condition, depending on the software protocol implemented by the application software.

**Figure 26-6. Typical Data Transmission**



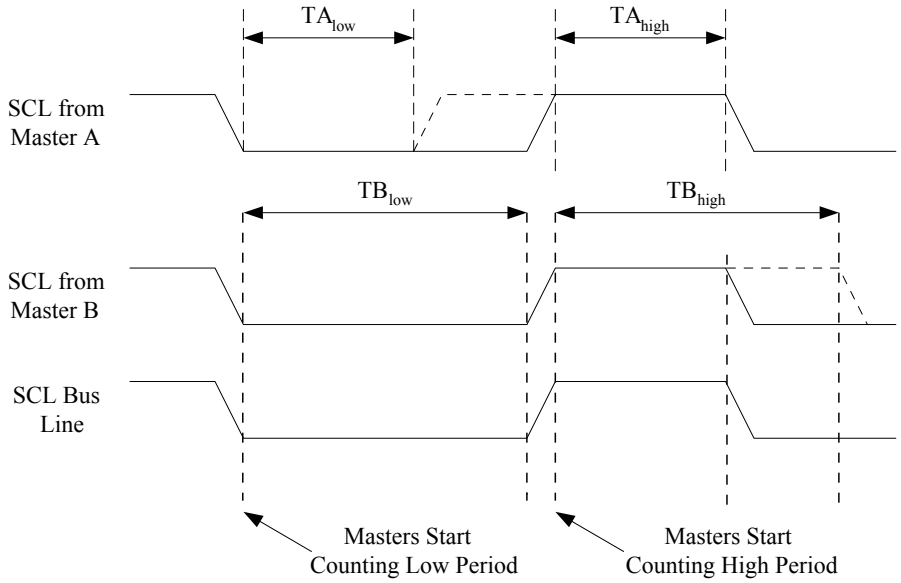

## 26.4. Multi-master Bus Systems, Arbitration and Synchronization

The TWI protocol allows bus systems with several masters. Special concerns have been taken in order to ensure that transmissions will proceed as normal, even if two or more masters initiate a transmission at the same time. Two problems arise in multi-master systems:

- An algorithm must be implemented allowing only one of the masters to complete the transmission. All other masters should cease transmission when they discover that they have lost the selection process. This selection process is called arbitration. When a contending master discovers that it has lost the arbitration process, it should immediately switch to Slave mode to check whether it is being addressed by the winning master. The fact that multiple masters have started transmission at the same time should not be detectable to the slaves, i.e. the data being transferred on the bus must not be corrupted.
- Different masters may use different SCL frequencies. A scheme must be devised to synchronize the serial clocks from all masters, in order to let the transmission proceed in a lockstep fashion. This will facilitate the arbitration process.

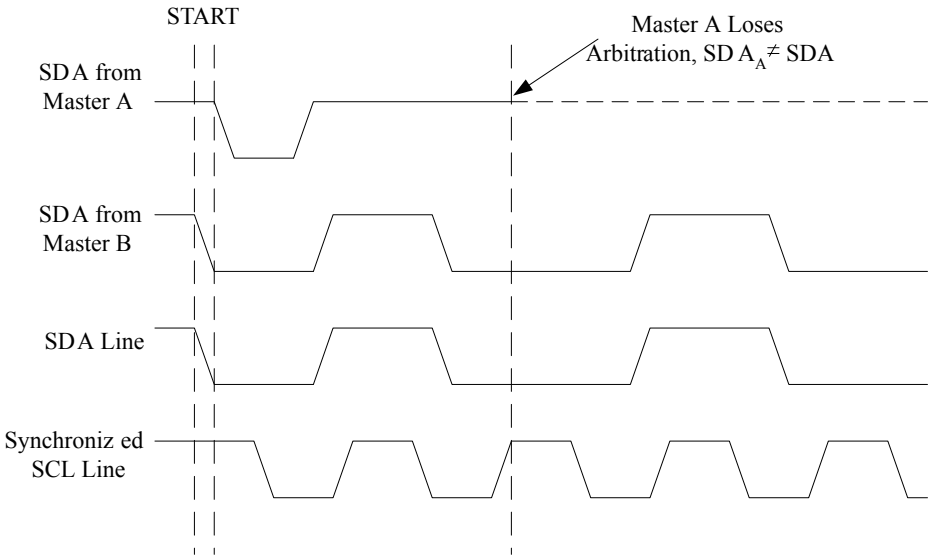
The wired-ANDing of the bus lines is used to solve both these problems. The serial clocks from all masters will be wired-ANDed, yielding a combined clock with a high period equal to the one from the Master with the shortest high period. The low period of the combined clock is equal to the low period of the Master with the longest low period. Note that all masters listen to the SCL line, effectively starting to count their SCL high and low time-out periods when the combined SCL line goes high or low, respectively.

**Figure 26-7. SCL Synchronization Between Multiple Masters**




Arbitration is carried out by all masters continuously monitoring the SDA line after outputting data. If the value read from the SDA line does not match the value the Master had output, it has lost the arbitration. Note that a Master can only lose arbitration when it outputs a high SDA value while another Master outputs a low value. The losing Master should immediately go to Slave mode, checking if it is being addressed by the winning Master. The SDA line should be left high, but losing masters are allowed to generate a clock signal until the end of the current data or address packet. Arbitration will continue until only one Master remains, and this may take many bits. If several masters are trying to address the same Slave, arbitration will continue into the data packet.

**Figure 26-8. Arbitration Between Two Masters**




Note that arbitration is not allowed between:

- A REPEATED START condition and a data bit
- A STOP condition and a data bit
- A REPEATED START and a STOP condition

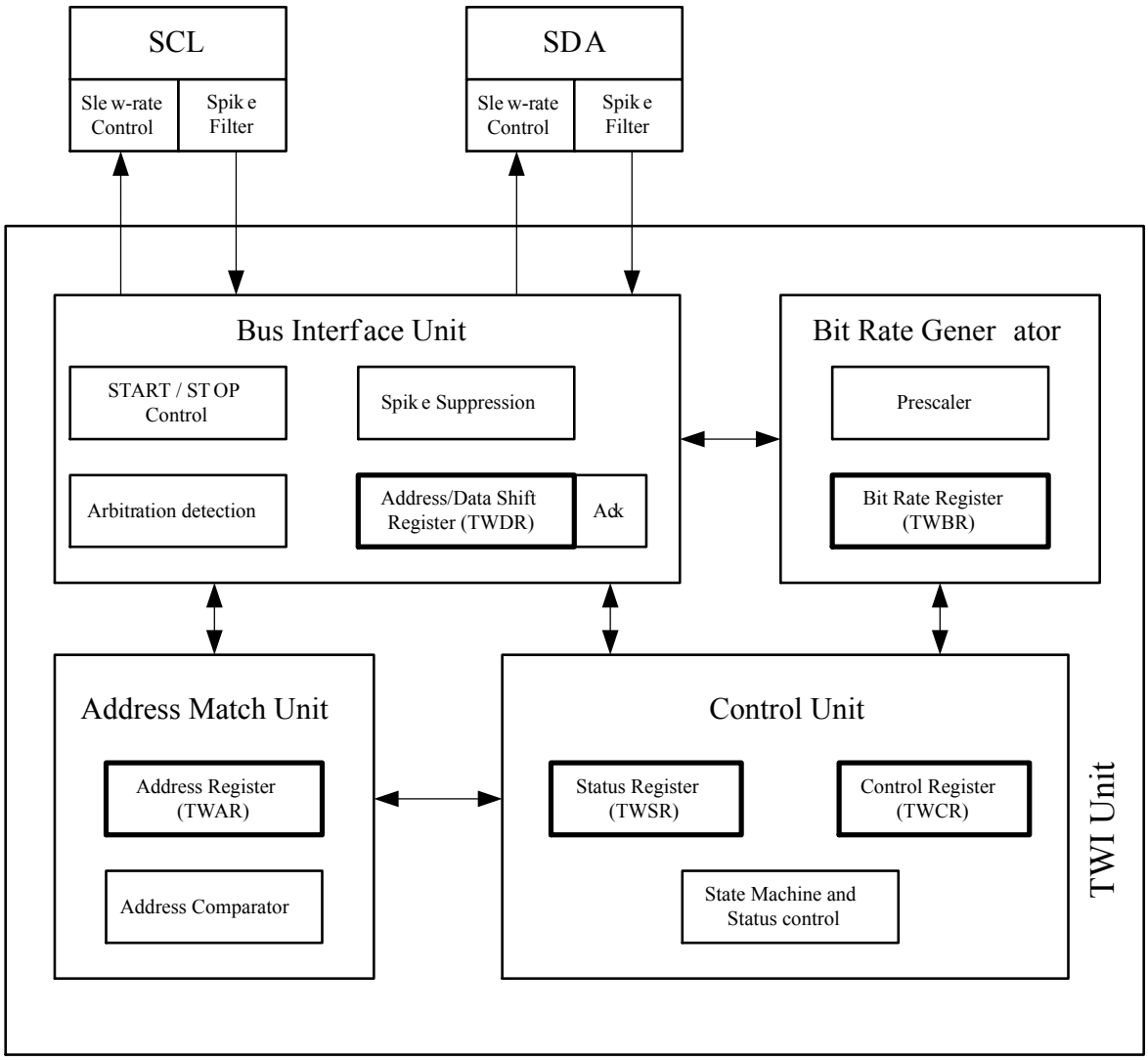
It is the user software's responsibility to ensure that these illegal arbitration conditions never occur. This implies that in multi-master systems, all data transfers must use the same composition of SLA+R/W and

data packets. In other words; All transmissions must contain the same number of data packets, otherwise the result of the arbitration is undefined.

## 26.5. Overview of the TWI Module

The TWI module is comprised of several submodules, as shown in the following figure. The registers drawn in a thick line are accessible through the AVR data bus.

**Figure 26-9. Overview of the TWI Module**




### 26.5.1. SCL and SDA Pins

These pins interface the AVR TWI with the rest of the MCU system. The output drivers contain a slew-rate limiter in order to conform to the TWI specification. The input stages contain a spike suppression unit removing spikes shorter than 50ns. Note that the internal pull-ups in the AVR pads can be enabled by setting the PORT bits corresponding to the SCL and SDA pins, as explained in the I/O Port section. The internal pull-ups can in some systems eliminate the need for external ones.

### 26.5.2. Bit Rate Generator Unit

This unit controls the period of SCL when operating in a Master mode. The SCL period is controlled by settings in the TWI Bit Rate Register (TWBRn) and the Prescaler bits in the TWI Status Register

(TWSRn). Slave operation does not depend on Bit Rate or Prescaler settings, but the CPU clock frequency in the Slave must be at least 16 times higher than the SCL frequency. Note that slaves may prolong the SCL low period, thereby reducing the average TWI bus clock period.

The SCL frequency is generated according to the following equation:

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot (\text{PrescalerValue})}$$

- TWBR = Value of the TWI Bit Rate Register TWBRn
- PrescalerValue = Value of the prescaler, see description of the TWI Prescaler bits in the TWSR Status Register description (TWSRn.TWPS[1:0])

**Note:** Pull-up resistor values should be selected according to the SCL frequency and the capacitive bus line load. See the *Two-Wire Serial Interface Characteristics* for a suitable value of the pull-up resistor.

**Related Links**

Two-wire Serial Interface Characteristics on page 391

### 26.5.3. Bus Interface Unit

This unit contains the Data and Address Shift Register (TWDRn), a START/STOP Controller and Arbitration detection hardware. The TWDRn contains the address or data bytes to be transmitted, or the address or data bytes received. In addition to the 8-bit TWDRn, the Bus Interface Unit also contains a register containing the (N)ACK bit to be transmitted or received. This (N)ACK Register is not directly accessible by the application software. However, when receiving, it can be set or cleared by manipulating the TWI Control Register (TWCRn). When in Transmitter mode, the value of the received (N)ACK bit can be determined by the value in the TWSRn.

The START/STOP Controller is responsible for generation and detection of START, REPEATED START, and STOP conditions. The START/STOP controller is able to detect START and STOP conditions even when the AVR MCU is in one of the sleep modes, enabling the MCU to wake up if addressed by a Master.

If the TWI has initiated a transmission as Master, the Arbitration Detection hardware continuously monitors the transmission trying to determine if arbitration is in process. If the TWI has lost an arbitration, the Control Unit is informed. Correct action can then be taken and appropriate status codes generated.

### 26.5.4. Address Match Unit

The Address Match unit checks if received address bytes match the seven-bit address in the TWI Address Register (TWARn). If the TWI General Call Recognition Enable bit (TWARn.TWGCE) is written to '1', all incoming address bits will also be compared against the General Call address. Upon an address match, the Control Unit is informed, allowing correct action to be taken. The TWI may or may not acknowledge its address, depending on settings in the TWI Control Register (TWCRn). The Address Match unit is able to compare addresses even when the AVR MCU is in sleep mode, enabling the MCU to wake up if addressed by a Master.

### 26.5.5. Control Unit

The Control unit monitors the TWI bus and generates responses corresponding to settings in the TWI Control Register (TWCRn). When an event requiring the attention of the application occurs on the TWI bus, the TWI Interrupt Flag (TWINT) is asserted. In the next clock cycle, the TWI Status Register (TWSRn) is updated with a status code identifying the event. The TWSRn only contains relevant status information when the TWI Interrupt Flag is asserted. At all other times, the TWSRn contains a special status code indicating that no relevant status information is available. As long as the TWINT Flag is set, the SCL line is held low. This allows the application software to complete its tasks before allowing the TWI transmission to continue.

The TWINT Flag is set in the following situations:

- After the TWI has transmitted a START/REPEATED START condition
- After the TWI has transmitted SLA+R/W
- After the TWI has transmitted an address byte
- After the TWI has lost arbitration
- After the TWI has been addressed by own slave address or general call
- After the TWI has received a data byte
- After a STOP or REPEATED START has been received while still addressed as a Slave
- When a bus error has occurred due to an illegal START or STOP condition

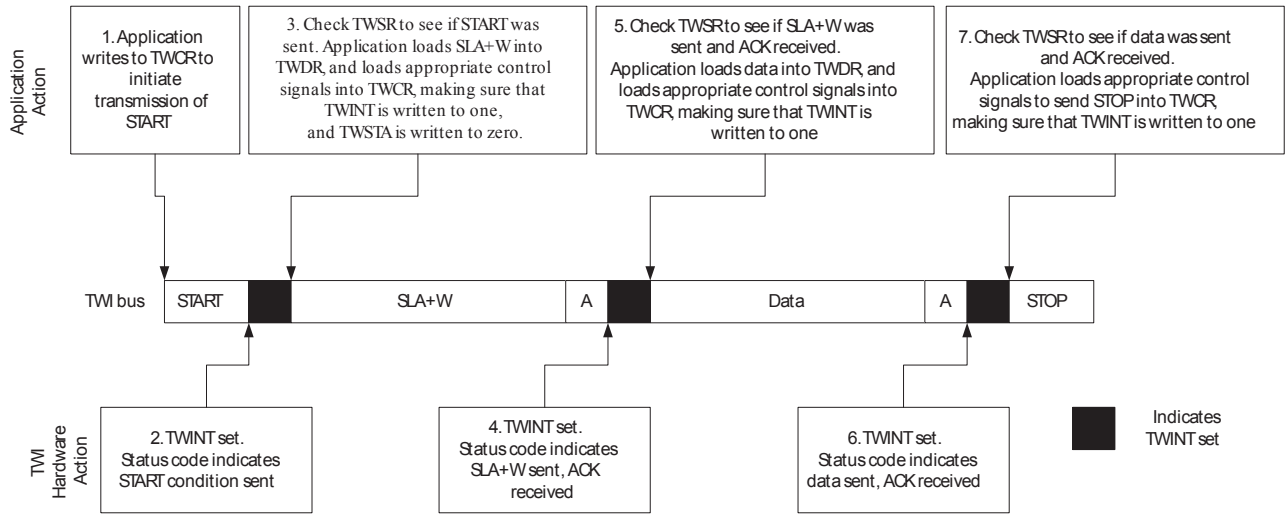
## 26.6. Using the TWI

The AVR TWI is byte-oriented and interrupt based. Interrupts are issued after all bus events, like reception of a byte or transmission of a START condition. Because the TWI is interrupt-based, the application software is free to carry on other operations during a TWI byte transfer. Note that the TWI Interrupt Enable (TWIE) bit in TWCRn together with the Global Interrupt Enable bit in SREG allow the application to decide whether or not assertion of the TWINT Flag should generate an interrupt request. If the TWIE bit is cleared, the application must poll the TWINT Flag in order to detect actions on the TWI bus.

When the TWINT Flag is asserted, the TWI has finished an operation and awaits application response. In this case, the TWI Status Register (TWSRn) contains a value indicating the current state of the TWI bus. The application software can then decide how the TWI should behave in the next TWI bus cycle by manipulating the TWCRn and TWDRn Registers.

The following figure illustrates a simple example of how the application can interface to the TWI hardware. In this example, a Master wishes to transmit a single data byte to a Slave. A more detailed explanation follows later in this section. Simple code examples are presented in the table below.

**Figure 26-10. Interfacing the Application to the TWI in a Typical Transmission**




1. The first step in a TWI transmission is to transmit a START condition. This is done by writing a specific value into TWCRn, instructing the TWI n hardware to transmit a START condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI n will not start any operation as long as the

TWINT bit in TWCRn is set. Immediately after the application has cleared TWINT, the TWI n will initiate transmission of the START condition.

2. When the START condition has been transmitted, the TWINT Flag in TWCRn is set, and TWSRn is updated with a status code indicating that the START condition has successfully been sent.
3. The application software should now examine the value of TWSRn, to make sure that the START condition was successfully transmitted. If TWSRn indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load SLA+W into TWDR. Remember that TWDRn is used both for address and data. After TWDRn has been loaded with the desired SLA+W, a specific value must be written to TWCRn, instructing the TWI n hardware to transmit the SLA+W present in TWDRn. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCRn is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the address packet.
4. When the address packet has been transmitted, the TWINT Flag in TWCRn is set, and TWSRn is updated with a status code indicating that the address packet has successfully been sent. The status code will also reflect whether a Slave acknowledged the packet or not.
5. The application software should now examine the value of TWSRn, to make sure that the address packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSRn indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load a data packet into TWDRn. Subsequently, a specific value must be written to TWCRn, instructing the TWI n hardware to transmit the data packet present in TWDRn. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI n will not start any operation as long as the TWINT bit in TWCRn is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the data packet.
6. When the data packet has been transmitted, the TWINT Flag in TWCRn is set, and TWSRn is updated with a status code indicating that the data packet has successfully been sent. The status code will also reflect whether a Slave acknowledged the packet or not.
7. The application software should now examine the value of TWSRn, to make sure that the data packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must write a specific value to TWCRn, instructing the TWI n hardware to transmit a STOP condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI n will not start any operation as long as the TWINT bit in TWCRn is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the STOP condition. Note that TWINT is *not* set after a STOP condition has been sent.

Even though this example is simple, it shows the principles involved in all TWI transmissions. These can be summarized as follows:

- When the TWI has finished an operation and expects application response, the TWINT Flag is set. The SCL line is pulled low until TWINT is cleared.
- When the TWINT Flag is set, the user must update all TWI n Registers with the value relevant for the next TWI n bus cycle. As an example, TWDRn must be loaded with the value to be transmitted in the next bus cycle.
- After all TWI n Register updates and other pending application software tasks have been completed, TWCRn is written. When writing TWCRn, the TWINT bit should be set. Writing a one to

TWINT clears the flag. The TWI n will then commence executing whatever operation was specified by the TWCRn setting.

The following table lists assembly and C implementation examples for TWI0. Note that the code below assumes that several definitions have been made, e.g. by using include-files.

**Table 26-2. Assembly and C Code Example**

| | Assembly Code Example | C Example | Comments |
|---|---|---|---|
| 1 | `ldi r16, (1<<TWINT)\|`<br>`(1<<TWSTA)\|(1<<TWEN)`<br>`out TWCR0, r16` | `TWCR0 = (1<<TWINT)\|`<br>`(1<<TWSTA)\|(1<<TWEN)` | Send START condition |
| 2 | `wait1:`<br>`in r16,TWCR0`<br>`sbrs r16,TWINT`<br>`rjmp wait1` | `while (!(TWCR0 &`<br>`(1<<TWINT)));` | Wait for TWINT Flag set. This indicates that the START condition has been transmitted. |
| 3 | `in r16,TWSR0`<br>`andi r16, 0xF8`<br>`cpi r16, START`<br>`brne ERROR` | `if ((TWSR0 & 0xF8) !=`<br>`START)`<br>`ERROR();` | Check value of TWI Status Register. Mask prescaler bits. If status different from START go to ERROR. |
| | `ldi r16, SLA_W`<br>`out TWDR0, r16`<br>`ldi r16, (1<<TWINT) \|`<br>`(1<<TWEN)`<br>`out TWCR0, r16` | `TWDR0 = SLA_W;`<br>`TWCR0 = (1<<TWINT) \|`<br>`(1<<TWEN);` | Load SLA_W into TWDR Register. Clear TWINT bit in TWCR to start transmission of address. |
| 4 | `wait2:`<br>`in r16,TWCR0`<br>`sbrs r16,TWINT`<br>`rjmp wait2` | `while (!(TWCR0 &`<br>`(1<<TWINT)));` | Wait for TWINT Flag set. This indicates that the SLA+W has been transmitted, and ACK/NACK has been received. |
| 5 | `in r16,TWSR0`<br>`andi r16, 0xF8`<br>`cpi r16, MT_SLA_ACK`<br>`brne ERROR` | `if ((TWSR0 & 0xF8) !=`<br>`MT_SLA_ACK) ERROR();` | Check value of TWI Status Register. Mask prescaler bits. If status different from MT_SLA_ACK go to ERROR. |
| | `ldi r16, DATA`<br>`out TWDR0, r16`<br>`ldi r16, (1<<TWINT) \|`<br>`(1<<TWEN)`<br>`out TWCR, r16` | `TWDR0 = DATA;`<br>`TWCR0 = (1<<TWINT) \|`<br>`(1<<TWEN);` | Load DATA into TWDR Register. Clear TWINT bit in TWCR to start transmission of data. |
| 6 | `wait3:`<br>`in r16,TWCR0`<br>`sbrs r16,TWINT`<br>`rjmp wait3` | `while (!(TWCR0 &`<br>`(1<<TWINT)));` | Wait for TWINT Flag set. This indicates that the DATA has been transmitted, and ACK/NACK has been received. |

| | Assembly Code Example | C Example | Comments |
|---|---|---|---|
| 7 | `in r16,TWSR0`<br>`andi r16, 0xF8`<br>`cpi r16, MT_DATA_ACK`<br>`brne ERROR` | `if ((TWSR0 & 0xF8) != MT_DATA_ACK) ERROR();` | Check value of TWI Status Register. Mask prescaler bits. If status different from MT_DATA_ACK go to ERROR. |
| | `ldi r16, (1<<TWINT)\|`<br>`(1<<TWEN)\| (1<<TWSTO)`<br>`out TWCR0, r16` | `TWCR0 = (1<<TWINT)\|`<br>`(1<<TWEN)\|(1<<TWSTO);` | Transmit STOP condition. |

## 26.7. Transmission Modes

The TWI can operate in one of four major modes:

- Master Transmitter (MT)
- Master Receiver (MR)
- Slave Transmitter (ST)
- Slave Receiver (SR)

Several of these modes can be used in the same application. As an example, the TWI can use MT mode to write data into a TWI EEPROM, MR mode to read the data back from the EEPROM. If other masters are present in the system, some of these might transmit data to the TWI, and then SR mode would be used. It is the application software that decides which modes are legal.

The following sections describe each of these modes. Possible status codes are described along with figures detailing data transmission in each of the modes. These figures use the following abbreviations:

| | |
|---|---|
| S | START condition |
| Rs | REPEATED START condition |
| R | Read bit (high level at SDA) |
| W | Write bit (low level at SDA) |
| A | Acknowledge bit (low level at SDA) |
| $\overline{A}$ | Not acknowledge bit (high level at SDA) |
| Data | 8-bit data byte |
| P | STOP condition |
| SLA | Slave Address |

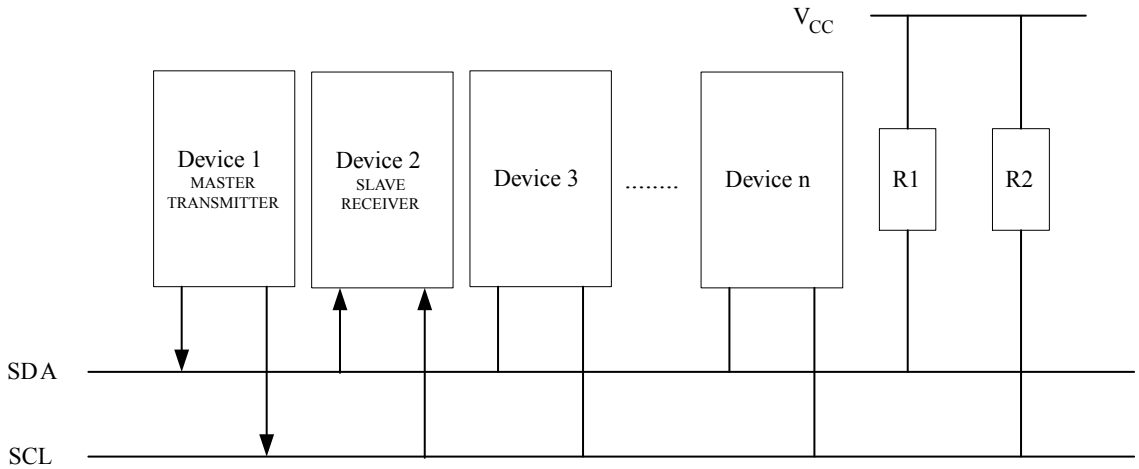
**Circles** are used to indicate that the TWINT Flag is set. The numbers in the circles show the status code held in TWSRn, with the prescaler bits masked to zero. At these points, actions must be taken by the application to continue or complete the TWI transfer. The TWI transfer is suspended until the TWINT Flag is cleared by software.

When the TWINT Flag is set, the status code in TWSRn is used to determine the appropriate software action. For each status code, the required software action and details of the following serial transfer are given below in the Status Code table for each mode. Note that the prescaler bits are masked to zero in these tables.

### 26.7.1. Master Transmitter Mode

In the Master Transmitter (MT) mode, a number of data bytes are transmitted to a Slave Receiver, see figure below. In order to enter a Master mode, a START condition must be transmitted. The format of the following address packet determines whether MT or Master Receiver (MR) mode is to be entered: If SLA +W is transmitted, MT mode is entered, if SLA+R is transmitted, MR mode is entered. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

**Figure 26-11. Data Transfer in Master Transmitter Mode**




A START condition is sent by writing a value to the TWI Control Register n (TWCRn) of the type TWCRn=1x10x10x:

- The TWI Enable bit (TWCRn.TWEN) must be written to '1' to enable the 2-wire Serial Interface
- The TWI Start Condition bit (TWCRn.TWSTA) must be written to '1' to transmit a START condition
- The TWI Interrupt Flag (TWCRn.TWINT) must be written to '1' to clear the flag.

The TWI n will then test the 2-wire Serial Bus and generate a START condition as soon as the bus becomes free. After a START condition has been transmitted, the TWINT Flag is set by hardware, and the status code in TWSRn will be 0x08 (see Status Code table below). In order to enter MT mode, SLA +W must be transmitted. This is done by writing SLA+W to the TWI Data Register (TWDRn). Thereafter, the TWCRn.TWINT Flag should be cleared (by writing a '1' to it) to continue the transfer. This is accomplished by writing a value to TWRC of the type TWCR=1x00x10x.

When SLA+W have been transmitted and an acknowledgment bit has been received, TWINT is set again and a number of status codes in TWSR are possible. Possible status codes in Master mode are 0x18, 0x20, or 0x38. The appropriate action to be taken for each of these status codes is detailed in the Status Code table below.

When SLA+W has been successfully transmitted, a data packet should be transmitted. This is done by writing the data byte to TWDR. TWDR must only be written when TWINT is high. If not, the access will be discarded, and the Write Collision bit (TWWC) will be set in the TWCRn Register. After updating TWDRn, the TWINT bit should be cleared (by writing '1' to it) to continue the transfer. This is accomplished by writing again a value to TWCRn of the type TWCRn=1x00x10x.

This scheme is repeated until the last byte has been sent and the transfer is ended, either by generating a STOP condition or a by a repeated START condition. A repeated START condition is accomplished by writing a regular START value TWCRn=1x10x10x. A STOP condition is generated by writing a value of the type TWCRn=1x01x10x.

After a repeated START condition (status code 0x10), the 2-wire Serial Interface can access the same Slave again, or a new Slave without transmitting a STOP condition. Repeated START enables the Master

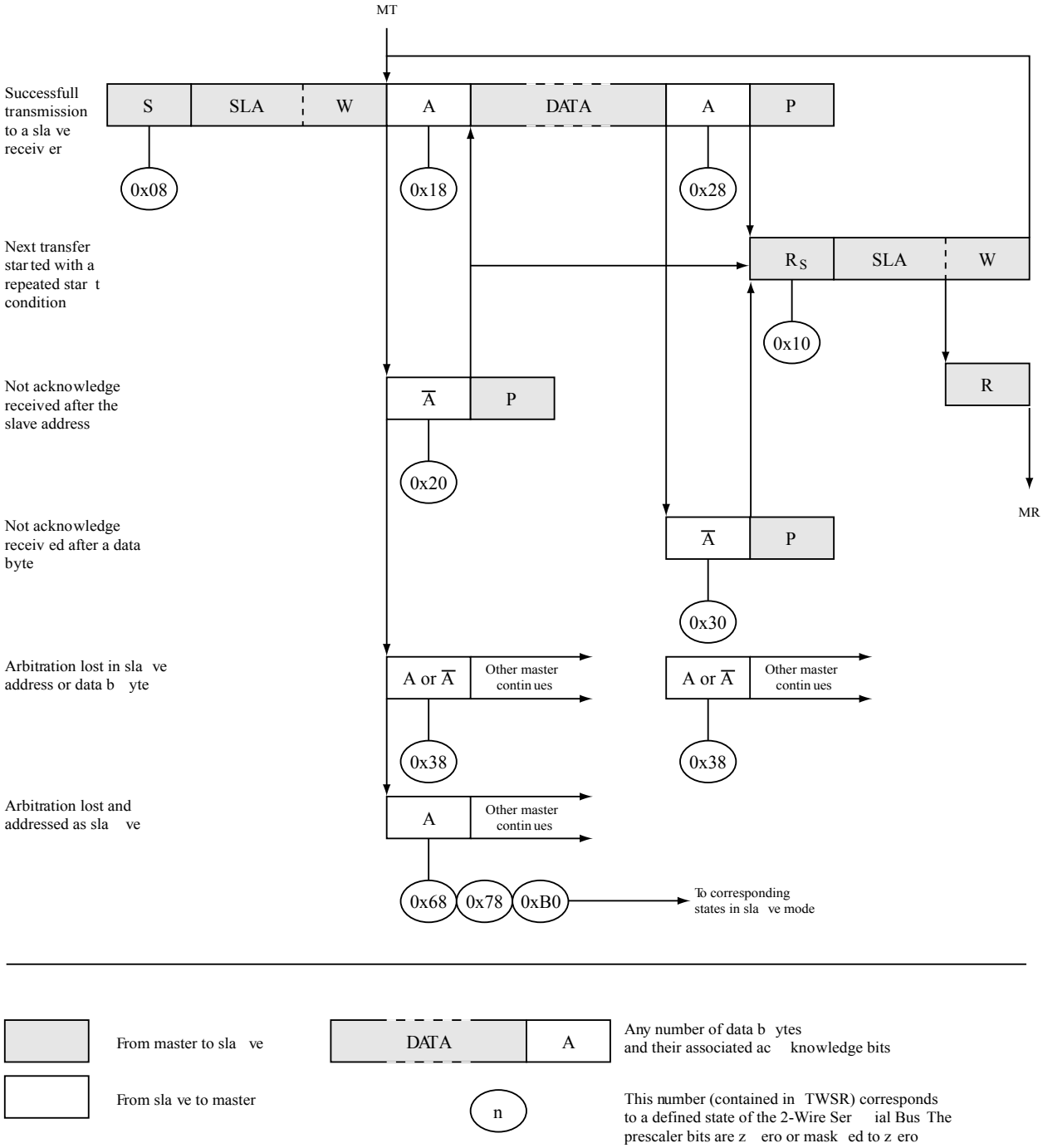
to switch between Slaves, Master Transmitter mode and Master Receiver mode without losing control of the bus.

**Table 26-3. Status Codes for Master Transmitter Mode**

| Status Code (TWSR) Prescaler Bits are 0 | Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware | Application Software Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWDR | To TWCRn | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0x08 | A START condition has been transmitted | Load SLA+W | 0 | 0 | 1 | X | SLA+W will be transmitted; ACK or NOT ACK will be received |
| 0x10 | A repeated START condition has been transmitted | Load SLA+W or | 0 | 0 | 1 | X | SLA+W will be transmitted; ACK or NOT ACK will be received |
| | | Load SLA+R | 0 | 0 | 1 | X | SLA+R will be transmitted; Logic will switch to Master Receiver mode |
| 0x18 | SLA+W has been transmitted; ACK has been received | Load data byte or | 0 | 0 | 1 | X | Data byte will be transmitted and ACK or NOT ACK will be received |
| | | No TWDR action or | 1 | 0 | 1 | X | Repeated START will be transmitted |
| | | No TWDR action or | 0 | 1 | 1 | X | STOP condition will be transmitted and TWSTO Flag will be reset |
| | | No TWDR action | 1 | 1 | 1 | X | STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset |
| 0x20 | SLA+W has been transmitted; NOT ACK has been received | Load data byte or | 0 | 0 | 1 | X | Data byte will be transmitted and ACK or NOT ACK will be received |
| | | No TWDR action or | 1 | 0 | 1 | X | Repeated START will be transmitted |
| | | No TWDR action or | 0 | 1 | 1 | X | STOP condition will be transmitted and TWSTO Flag will be reset |
| | | No TWDR action | 1 | 1 | 1 | X | STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset |

| Status Code (TWSR) Prescaler Bits are 0 | Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware | Application Software Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWDR | To TWCRn | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0x28 | Data byte has been transmitted; ACK has been received | Load data byte or | 0 | 0 | 1 | X | Data byte will be transmitted and ACK or NOT ACK will be received |
| | | No TWDR action or | 1 | 0 | 1 | X | Repeated START will be transmitted |
| | | No TWDR action or | 1 | 0 | 1 | X | STOP condition will be transmitted and TWSTO Flag will be reset |
| | | No TWDR action | 1 | 1 | 1 | X | STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset |
| 0x30 | Data byte has been transmitted; NOT ACK has been received | Load data byte or | 0 | 0 | 1 | X | Data byte will be transmitted and ACK or NOT ACK will be received |
| | | No TWDR action or | 1 | 0 | 1 | X | Repeated START will be transmitted |
| | | No TWDR action or | 0 | 1 | 1 | X | STOP condition will be transmitted and TWSTO Flag will be reset |
| | | No TWDR action | 1 | 1 | 1 | X | STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset |
| 0x38 | Arbitration lost in SLA+W or data bytes | No TWDR action or | 0 | 0 | 1 | X | 2-wire Serial Bus will be released and not addressed Slave mode entered |
| | | No TWDR action | 1 | 0 | 1 | X | A START condition will be transmitted when the bus becomes free |

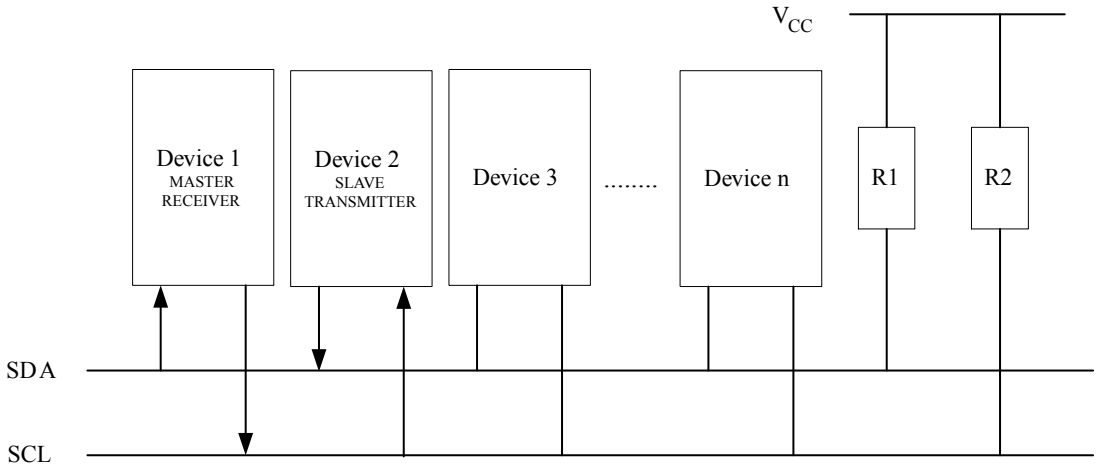
**Figure 26-12. Formats and States in the Master Transmitter Mode**




### 26.7.2. Master Receiver Mode

In the Master Receiver (MR) mode, a number of data bytes are received from a Slave Transmitter (see next figure). In order to enter a Master mode, a START condition must be transmitted. The format of the following address packet determines whether Master Transmitter (MT) or MR mode is to be entered. If SLA+W is transmitted, MT mode is entered, if SLA+R is transmitted, MR mode is entered. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

**Figure 26-13. Data Transfer in Master Receiver Mode**




A START condition is sent by writing to the TWI Control register (TWCRn) a value of the type TWCRn=1x10x10x:

- TWCRn.TWEN must be written to '1' to enable the 2-wire Serial Interface
- TWCRn.TWSTA must be written to '1' to transmit a START condition
- TWCRn.TWINT must be cleared by writing a '1' to it.

The TWI will then test the 2-wire Serial Bus and generate a START condition as soon as the bus becomes free. After a START condition has been transmitted, the TWINT Flag is set by hardware, and the status code in TWSRn will be 0x08 (see Status Code table below). In order to enter MR mode, SLA +R must be transmitted. This is done by writing SLA+R to TWDR. Thereafter, the TWINT flag should be cleared (by writing '1' to it) to continue the transfer. This is accomplished by writing the a value to TWCRn of the type TWCRn=1x00x10x.

When SLA+R have been transmitted and an acknowledgment bit has been received, TWINT is set again and a number of status codes in TWSRn are possible. Possible status codes in Master mode are 0x38, 0x40, or 0x48. The appropriate action to be taken for each of these status codes is detailed in the table below. Received data can be read from the TWDR Register when the TWINT Flag is set high by hardware. This scheme is repeated until the last byte has been received. After the last byte has been received, the MR should inform the ST by sending a NACK after the last received data byte. The transfer is ended by generating a STOP condition or a repeated START condition. A repeated START condition is sent by writing to the TWI Control register (TWCRn) a value of the type TWCRn=1x10x10x again. A STOP condition is generated by writing TWCRn=1xx01x10x:

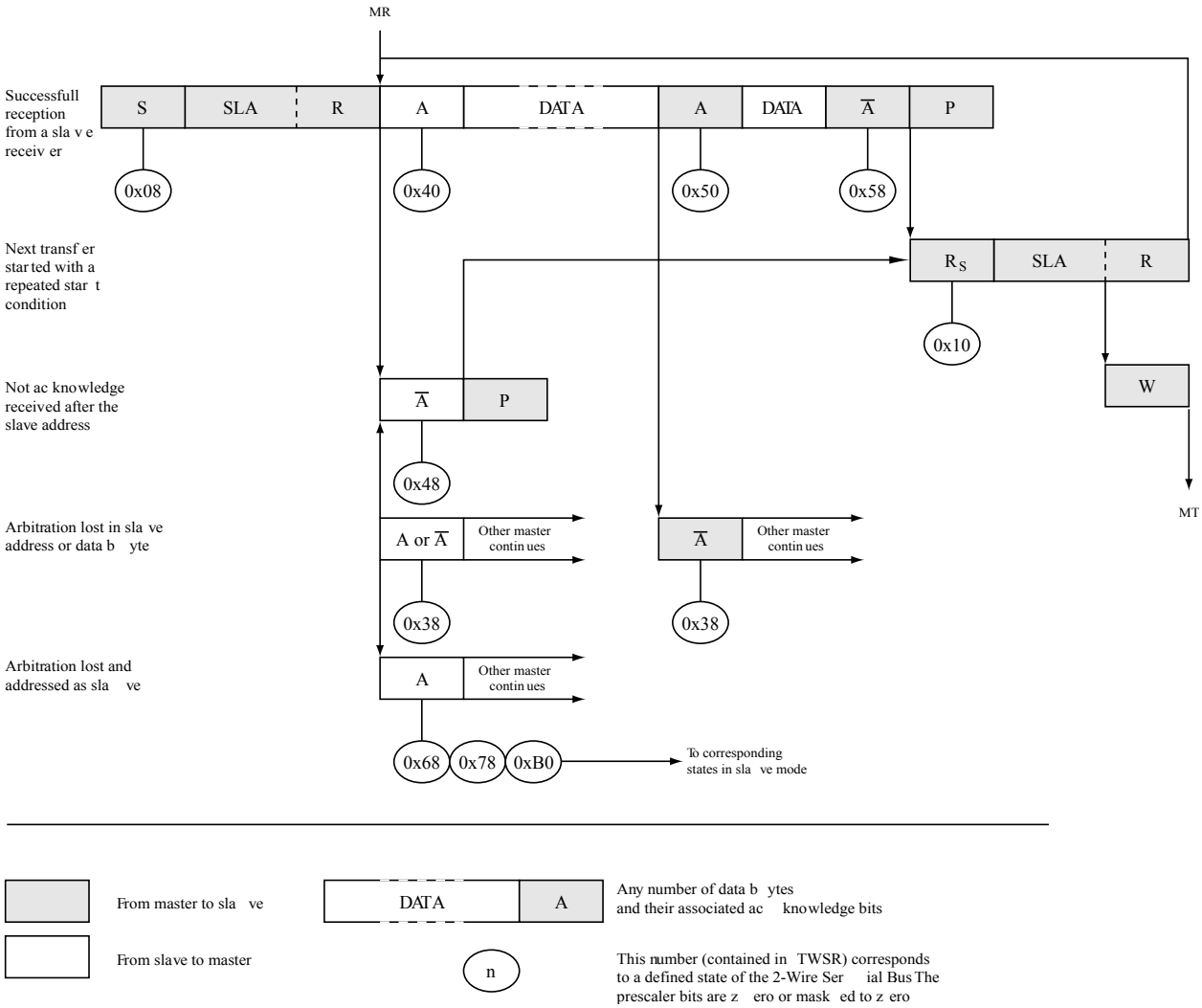
After a repeated START condition (status code 0x10) the 2-wire Serial Interface can access the same Slave again, or a new Slave without transmitting a STOP condition. Repeated START enables the Master to switch between Slaves, Master Transmitter mode and Master Receiver mode without losing control over the bus.

**Table 26-4. Status codes for Master Receiver Mode**

| Status Code (TWSRn) Prescaler Bits are 0 | Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware | Application Software Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWD | To TWCRn | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0x08 | A START condition has been transmitted | Load SLA+R | 0 | 0 | 1 | X | SLA+R will be transmitted ACK or NOT ACK will be received |
| 0x10 | A repeated START condition has been transmitted | Load SLA+R | 0 | 0 | 1 | X | SLA+R will be transmitted ACK or NOT ACK will be received |
| | | Load SLA+W | 0 | 0 | 1 | X | SLA+W will be transmitted<br>Logic will switch to Master Transmitter mode |
| 0x38 | Arbitration lost in SLA+R or NOT ACK bit | No TWDR action | 0 | 0 | 1 | X | 2-wire Serial Bus will be released and not addressed Slave mode will be entered |
| | | | 1 | 0 | 1 | X | A START condition will be transmitted when the bus becomes free |
| 0x40 | SLA+R has been transmitted; ACK has been received | No TWDR action | 0 | 0 | 1 | 0 | Data byte will be received and NOT ACK will be returned |
| | | | 0 | 0 | 1 | 1 | Data byte will be received and ACK will be returned |
| 0x48 | SLA+R has been transmitted; NOT ACK has been received | | 1 | 0 | 1 | X | Repeated START will be transmitted |
| | | | 0 | 1 | 1 | X | STOP condition will be transmitted and TWSTO Flag will be reset |
| | | | 1 | 1 | 1 | X | STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset |
| 0x50 | Data byte has been received; ACK has been returned | Read data byte | 0 | 0 | 1 | 0 | Data byte will be received and NOT ACK will be returned |
| | | | 0 | 0 | 1 | 1 | Data byte will be received and ACK will be returned |

| Status Code (TWSRn) Prescaler Bits are 0 | Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware | Application Software Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWD | To TWCRn | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0x58 | Data byte has been received; NOT ACK has been returned | Read data byte | 1 | 0 | 1 | X | Repeated START will be transmitted |
| | | | 0 | 1 | 1 | X | STOP condition will be transmitted and TWSTO Flag will be reset |
| | | | 1 | 1 | 1 | X | STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset |

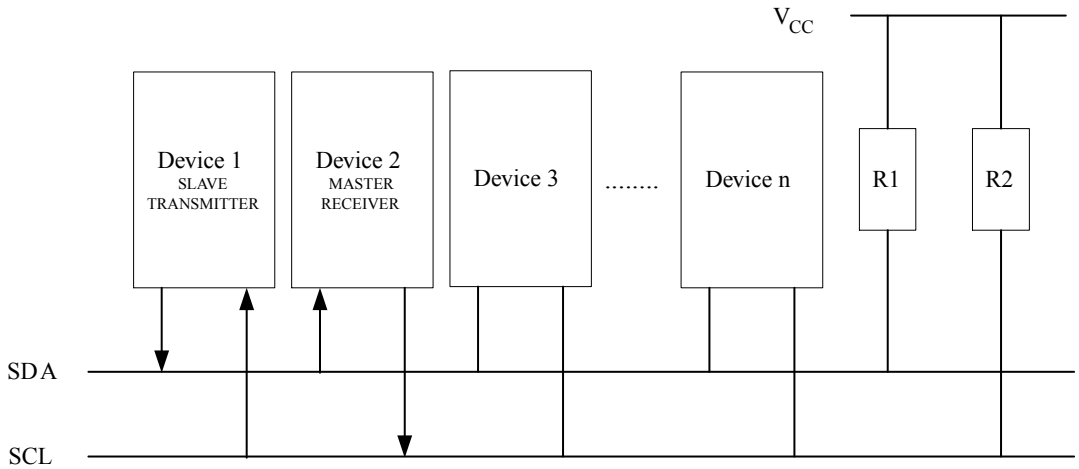
**Figure 26-14. Formats and States in the Master Receiver Mode**



### 26.7.3. Slave Transmitter Mode

In the Slave Transmitter (ST) mode, a number of data bytes are transmitted to a Master Receiver, as in the figure below. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

**Figure 26-15. Data Transfer in Slave Transmitter Mode**




To initiate the SR mode, the TWI (Slave) Address Register (TWARn) and the TWI Control Register (TWCRn) must be initialized as follows:

The upper seven bits of TWARn are the address to which the 2-wire Serial Interface will respond when addressed by a Master (TWARn.TWA[6:0]). If the LSB of TWARn is written to TWARn.TWGCI=1, the TWI will respond to the general call address (0x00), otherwise it will ignore the general call address.

TWCRn must hold a value of the type TWCRn=0100010x - TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgment of the device's own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWARn and TWCRn have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is "1" (read), the TWI will operate in ST mode, otherwise SR mode is entered. After its own slave address and the write bit have been received, the TWINT Flag is set and a valid status code can be read from TWSRb. The status code is used to determine the appropriate sofTWARne action. The appropriate action to be taken for each status code is detailed in the table below. The ST mode may also be entered if arbitration is lost while the TWI is in the Master mode (see state 0xB0).

If the TWCRn.TWEA bit is written to zero during a transfer, the TWI will transmit the last byte of the transfer. State 0xC0 or state 0xC8 will be entered, depending on whether the Master Receiver transmits a NACK or ACK after the final byte. The TWI is switched to the not addressed Slave mode, and will ignore the Master if it continues the transfer. Thus the Master Receiver receives all '1' as serial data. State 0xC8 is entered if the Master demands additional data bytes (by transmitting ACK), even though the Slave has transmitted the last byte (TWEA zero and expecting NACK from the Master).

While TWCRn.TWEA is zero, the TWI does not respond to its own slave address. However, the 2-wire Serial Bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the 2-wire Serial Bus.

In all sleep modes other than Idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the 2-wire Serial Bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock will low during the wake up and until the TWINT Flag is cleared (by writing '1' to it). Further data transmission will be carried out as normal, with the AVR clocks running as normal. Observe that if the

AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

**Note:** The 2-wire Serial Interface Data Register (TWDRn) does not reflect the last byte present on the bus when waking up from these Sleep modes.

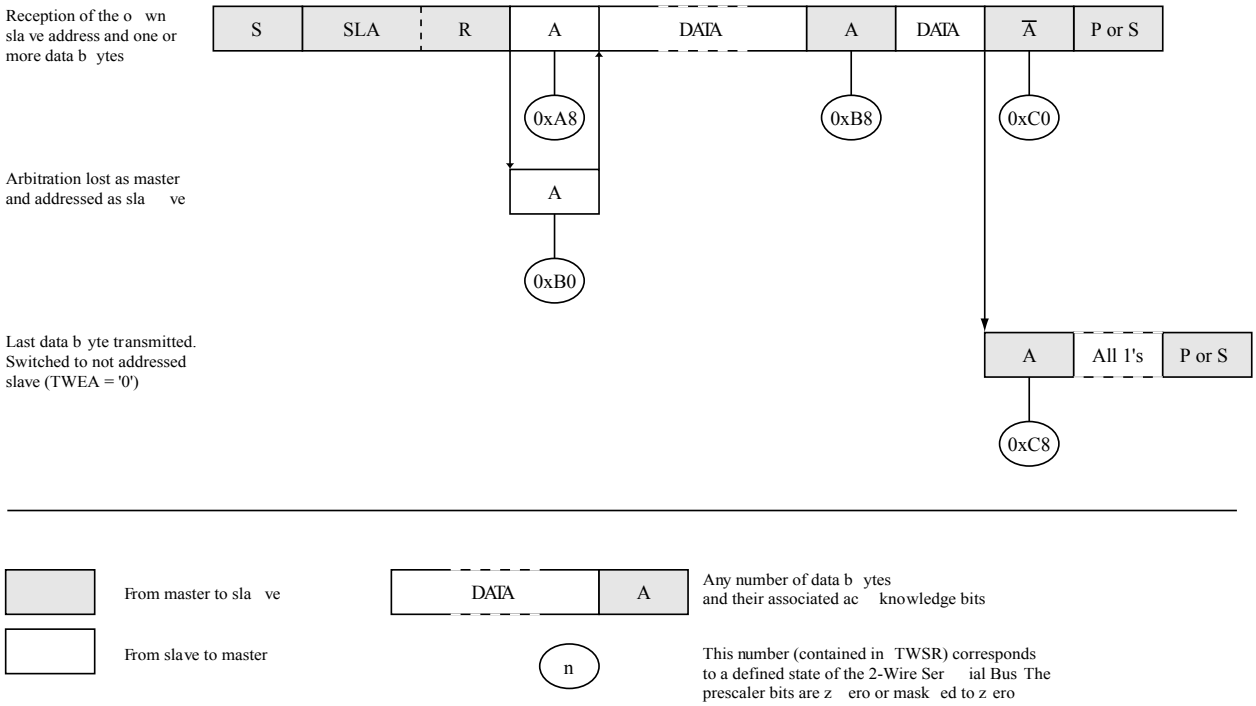
**Table 26-5. Status Codes for Slave Transmitter Mode**

| Status Code (TWSRb) Prescaler Bits are 0 | Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware | Application SofTWARne Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWDRn | To TWCRn | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0xA8 | Own SLA+R has been received; ACK has been returned | Load data byte | X | 0 | 1 | 0 | Last data byte will be transmitted and NOT ACK should be received |
| | | | X | 0 | 1 | 1 | Data byte will be transmitted and ACK should be received |
| 0xB0 | Arbitration lost in SLA+R/W as Master; own SLA+R has been received; ACK has been returned | Load data byte | X | 0 | 1 | 0 | Last data byte will be transmitted and NOT ACK should be received |
| | | | X | 0 | 1 | 1 | Data byte will be transmitted and ACK should be received |
| 0xB8 | Data byte in TWDRn has been transmitted; ACK has been received | Load data byte | X | 0 | 1 | 0 | Last data byte will be transmitted and NOT ACK should be received |
| | | | X | 0 | 1 | 1 | Data byte will be transmitted and ACK should be received |

| Status Code (TWSRb) Prescaler Bits are 0 | Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware | Application SofTWARne Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWDRn | To TWCRn | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0xC0 | Data byte in TWDRn has been transmitted;<br>NOT ACK has been received | No TWDRn action | 0 | 0 | 1 | 0 | Switched to the not addressed Slave mode;<br>no recognition of own SLA or GCA |
| | | | 0 | 0 | 1 | 1 | Switched to the not addressed Slave mode;<br>own SLA will be recognized;<br>GCA will be recognized if TWGCE = “1” |
| | | | 1 | 0 | 1 | 0 | Switched to the not addressed Slave mode;<br>no recognition of own SLA or GCA;<br>a START condition will be transmitted when the bus becomes free |
| | | | 1 | 0 | 1 | 1 | Switched to the not addressed Slave mode;<br>own SLA will be recognized;<br>GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free |

| Status Code (TWSRb) Prescaler Bits are 0 | Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware | Application SofTWARne Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWDRn | To TWCRn | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0xC8 | Last data byte in TWDRn has been transmitted (TWEA = “0”);<br>ACK has been received | No TWDRn action | 0 | 0 | 1 | 0 | Switched to the not addressed Slave mode;<br>no recognition of own SLA or GCA |
| | | | 0 | 0 | 1 | 1 | Switched to the not addressed Slave mode;<br>own SLA will be recognized;<br>GCA will be recognized if TWGCE = “1” |
| | | | 1 | 0 | 1 | 0 | Switched to the not addressed Slave mode;<br>no recognition of own SLA or GCA;<br>a START condition will be transmitted when the bus<br>becomes free |
| | | | 1 | 0 | 1 | 1 | Switched to the not addressed Slave mode;<br>own SLA will be recognized;<br>GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free |

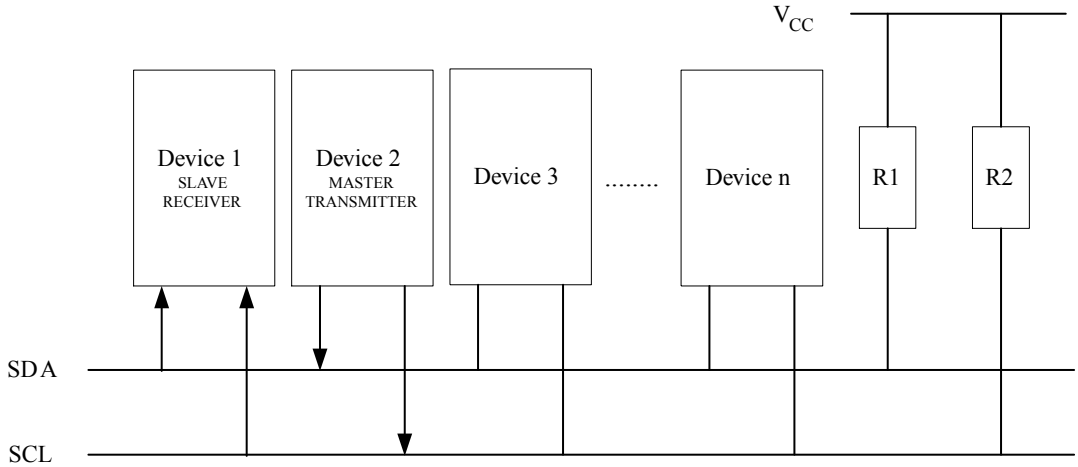
**Figure 26-16. Formats and States in the Slave Transmitter Mode**




### 26.7.4. Slave Receiver Mode

In the Slave Receiver (SR) mode, a number of data bytes are received from a Master Transmitter (see figure below). All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

**Figure 26-17. Data transfer in Slave Receiver mode**




To initiate the SR mode, the TWI (Slave) Address Register n (TWARn) and the TWI Control Register n (TWCRn) must be initialized as follows:

The upper seven bits of TWARn are the address to which the 2-wire Serial Interface will respond when addressed by a Master (TWARn.TWA[6:0]). If the LSB of TWARn is written to TWARn.TWGCI=1, the TWI n will respond to the general call address (0x00), otherwise it will ignore the general call address.

TWCRn must hold a value of the type TWCRn=0100010x - TWCRn.TWEN must be written to '1' to enable the TWI. TWCRn.TWEA bit must be written to '1' to enable the acknowledgment of the device's own slave address or the general call address. TWCRn.TWSTA and TWSTO must be written to zero.

When TWARn and TWCRn have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address, if enabled) followed by the data direction bit. If the direction bit is '0' (write), the TWI will operate in SR mode, otherwise ST mode is entered. After its own slave address and the write bit have been received, the TWINT Flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action, as detailed in the table below. The SR mode may also be entered if arbitration is lost while the TWI is in the Master mode (see states 0x68 and 0x78).

If the TWCRn.TWEA bit is reset during a transfer, the TWI will return a "Not Acknowledge" ('1') to SDA after the next received data byte. This can be used to indicate that the Slave is not able to receive any more bytes. While TWEA is zero, the TWI does not acknowledge its own slave address. However, the 2-wire Serial Bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the 2-wire Serial Bus.

In all sleep modes other than Idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the 2-wire Serial Bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock low during the wake up and until the TWINT Flag is cleared (by writing '1' to it). Further data reception will be carried out as normal, with the AVR clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

**Note:** The 2-wire Serial Interface Data Register (TWDRn) does not reflect the last byte present on the bus when waking up from these Sleep modes.

**Table 26-6. Status Codes for Slave Receiver Mode**

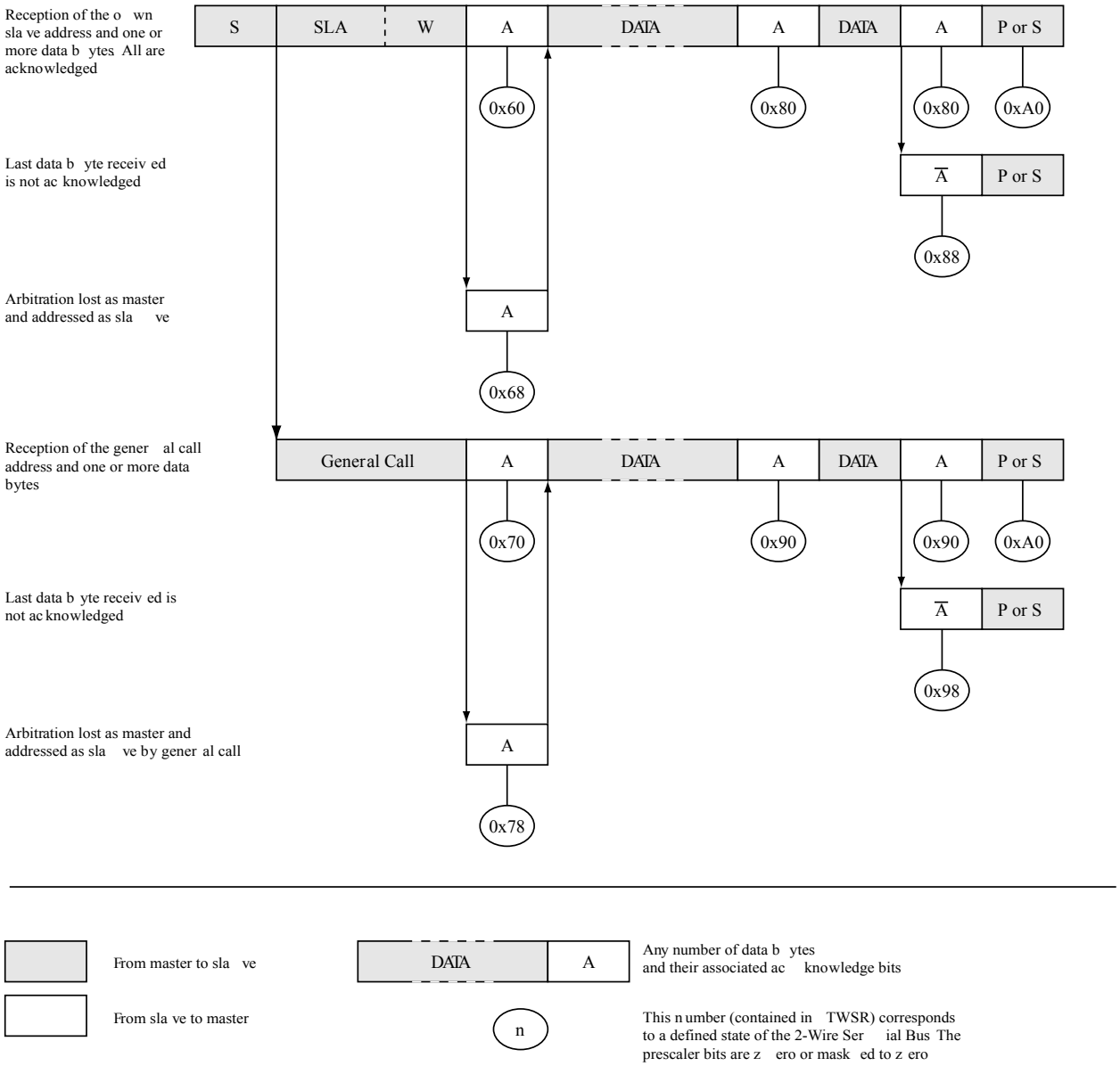
| Status Code (TWSR) Prescaler Bits are 0 | Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware | Application SofTWARne Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWDRn | To TWCRn | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0x60 | Own SLA+W has been received; ACK has been returned | No TWDRn action | X | 0 | 1 | 0 | Data byte will be received and NOT ACK will be returned |
| | | | X | 0 | 1 | 1 | Data byte will be received and ACK will be returned |
| 0x68 | Arbitration lost in SLA+R/W as Master; own SLA+W has been received; ACK has been returned | No TWDRn action | X | 0 | 1 | 0 | Data byte will be received and NOT ACK will be returned |
| | | | X | 0 | 1 | 1 | Data byte will be received and ACK will be returned |
| 0x70 | General call address has been received; ACK has been returned | No TWDRn action | X | 0 | 1 | 0 | Data byte will be received and NOT ACK will be returned |
| | | | X | 0 | 1 | 1 | Data byte will be received and ACK will be returned |
| 0x78 | Arbitration lost in SLA+R/W as Master; General call address has been received; ACK has been returned | No TWDRn action | X | 0 | 1 | 0 | Data byte will be received and NOT ACK will be returned |
| | | | X | 0 | 1 | 1 | Data byte will be received and ACK will be returned |

| Status Code (TWSR) Prescaler Bits are 0 | Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware | Application SofTWARne Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWDRn | To TWCRn | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0x80 | Previously addressed with own SLA+W; data has been received; ACK has been returned | Read data byte | X | 0 | 1 | 0 | Data byte will be received and NOT ACK will be returned |
| | | | X | 0 | 1 | 1 | Data byte will be received and ACK will be returned |
| 0x88 | Previously addressed with own SLA+W; data has been received; NOT ACK has been returned | Read data byte | 0 | 0 | 1 | 0 | Switched to the not addressed Slave mode; no recognition of own SLA or GCA |
| | | | 0 | 0 | 1 | 1 | Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1” |
| | | | 1 | 0 | 1 | 0 | Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free |
| | | | 1 | 0 | 1 | 1 | Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free |
| 0x90 | Previously addressed with general call; data has been received; ACK has been returned | Read data byte | X | 0 | 1 | 0 | Data byte will be received and NOT ACK will be returned |
| | | | X | 0 | 1 | 1 | Data byte will be received and ACK will be returned |

| Status Code (TWSR) Prescaler Bits are 0 | Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware | Application SofTWARne Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWDRn | To TWCRn | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0x98 | Previously addressed with general call;<br>data has been received;<br>NOT ACK has been returned | Read data byte | 0 | 0 | 1 | 0 | Switched to the not addressed Slave mode;<br>no recognition of own SLA or GCA |
| | | | 0 | 0 | 1 | 1 | Switched to the not addressed Slave mode;<br>own SLA will be recognized;<br>GCA will be recognized if TWGCE = “1” |
| | | | 1 | 0 | 1 | 0 | Switched to the not addressed Slave mode;<br>no recognition of own SLA or GCA;<br>a START condition will be transmitted when the bus becomes free |
| | | | 1 | 0 | 1 | 1 | Switched to the not addressed Slave mode;<br>own SLA will be recognized;<br>GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free |

| Status Code (TWSR) Prescaler Bits are 0 | Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware | Application SofTWARne Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWDRn | To TWCRn | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0xA0 | A STOP condition or repeated START condition has been received while still addressed as Slave | No action | 0 | 0 | 1 | 0 | Switched to the not addressed Slave mode; no recognition of own SLA or GCA |
| | | | 0 | 0 | 1 | 1 | Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1” |
| | | | 1 | 0 | 1 | 0 | Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free |
| | | | 1 | 0 | 1 | 1 | Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free |

**Figure 26-18. Formats and States in the Slave Receiver Mode**




### 26.7.5. Miscellaneous States

There are two status codes that do not correspond to a defined TWI state, see the table in this section.

Status 0xF8 indicates that no relevant information is available because the TWINT Flag is not set. This occurs between other states, and when the TWI is not involved in a serial transfer.

Status 0x00 indicates that a bus error has occurred during a 2-wire Serial Bus transfer. A bus error occurs when a START or STOP condition occurs at an illegal position in the format frame. Examples of such illegal positions are during the serial transfer of an address byte, a data byte, or an acknowledge bit. When a bus error occurs, TWINT is set. To recover from a bus error, the TWSTO Flag must set and TWINT must be cleared by writing a logic one to it. This causes the TWI to enter the not addressed Slave mode and to clear the TWSTO Flag (no other bits in TWCRn are affected). The SDA and SCL lines are released, and no STOP condition is transmitted.

**Table 26-7. Miscellaneous States**

| Status Code (TWSR) Prescaler Bits are 0 | Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware | Application Software Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWDRn | To TWCRn | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0xF8 | No relevant state information available; TWINT = “0” | No TWDRn action | No TWCRn action | | | | Wait or proceed current transfer |
| 0x00 | Bus error due to an illegal START or STOP condition | No TWDRn action | 0 | 1 | 1 | X | Only the internal hardware is affected, no STOP condition is sent on the bus. In all cases, the bus is released and TWSTO is cleared. |

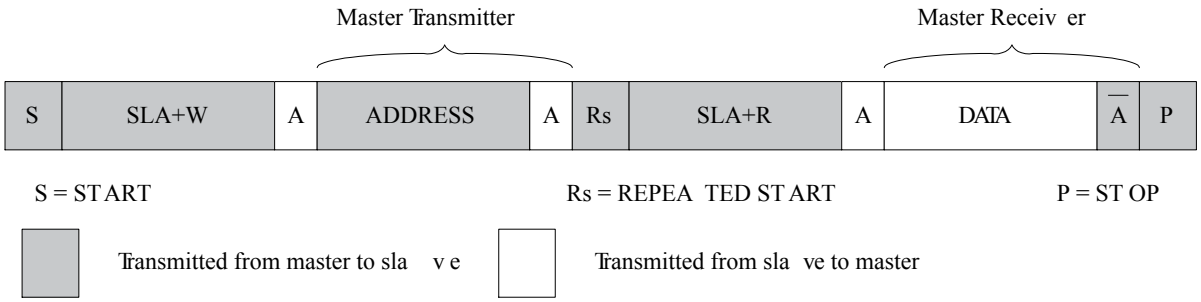
### 26.7.6. Combining Several TWI Modes

In some cases, several TWI modes must be combined in order to complete the desired action. Consider for example reading data from a serial EEPROM. Typically, such a transfer involves the following steps:

1. The transfer must be initiated.
2. The EEPROM must be instructed what location should be read.
3. The reading must be performed.
4. The transfer must be finished.

Note that data is transmitted both from Master to Slave and vice versa. The Master must instruct the Slave what location it wants to read, requiring the use of the MT mode. Subsequently, data must be read from the Slave, implying the use of the MR mode. Thus, the transfer direction must be changed. The Master must keep control of the bus during all these steps, and the steps should be carried out as an atomical operation. If this principle is violated in a multi master system, another Master can alter the data pointer in the EEPROM between steps 2 and 3, and the Master will read the wrong data location. Such a change in transfer direction is accomplished by transmitting a REPEATED START between the transmission of the address byte and reception of the data. After a REPEATED START, the Master keeps ownership of the bus. The flow in this transfer is depicted in the following figure:

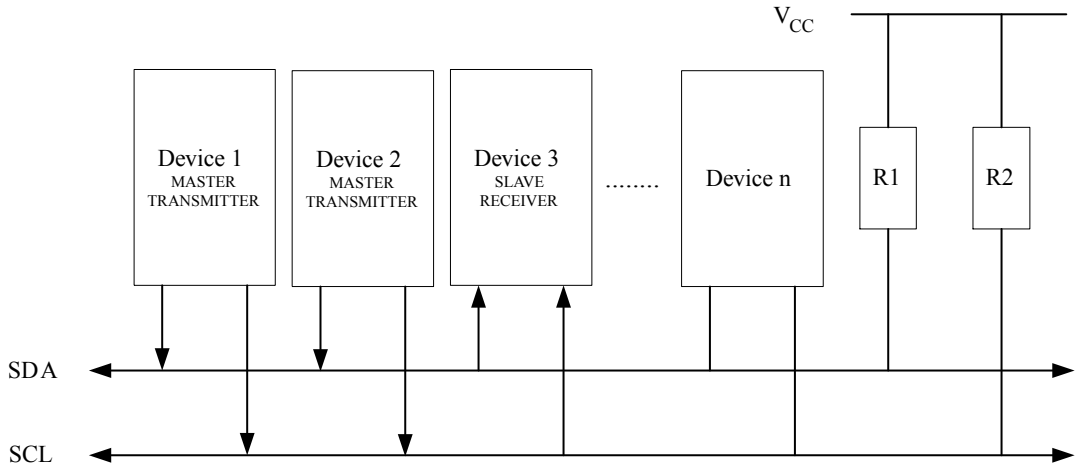
**Figure 26-19. Combining Several TWI Modes to Access a Serial EEPROM**




## 26.8. Multi-master Systems and Arbitration

If multiple masters are connected to the same bus, transmissions may be initiated simultaneously by one or more of them. The TWI standard ensures that such situations are handled in such a way that one of the masters will be allowed to proceed with the transfer, and that no data will be lost in the process. An example of an arbitration situation is depicted below, where two masters are trying to transmit data to a Slave Receiver.

**Figure 26-20. An Arbitration Example**

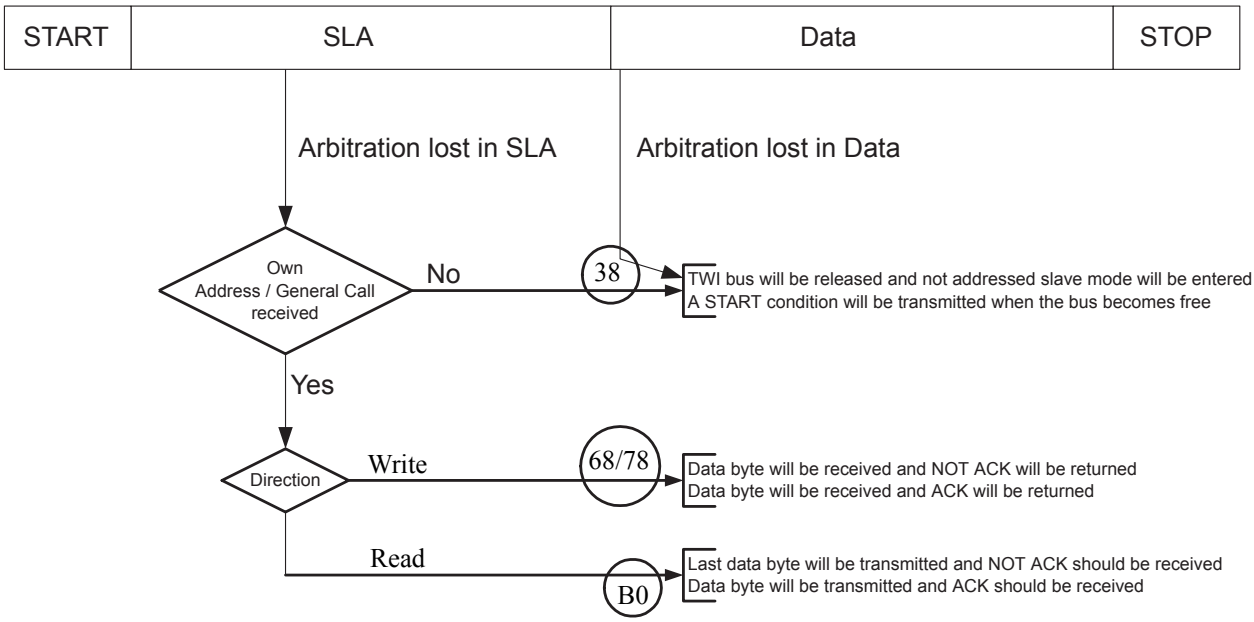



Several different scenarios may arise during arbitration, as described below:

- Two or more masters are performing identical communication with the same Slave. In this case, neither the Slave nor any of the masters will know about the bus contention.
- Two or more masters are accessing the same Slave with different data or direction bit. In this case, arbitration will occur, either in the READ/WRITE bit or in the data bits. The masters trying to output a '1' on SDA while another Master outputs a zero will lose the arbitration. Losing masters will switch to not addressed Slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.
- Two or more masters are accessing different slaves. In this case, arbitration will occur in the SLA bits. Masters trying to output a '1' on SDA while another Master outputs a zero will lose the arbitration. Masters losing arbitration in SLA will switch to Slave mode to check if they are being addressed by the winning Master. If addressed, they will switch to SR or ST mode, depending on the value of the READ/WRITE bit. If they are not being addressed, they will switch to not addressed Slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.

This is summarized in the next figure. Possible status values are given in circles.

**Figure 26-21. Possible Status Codes Caused by Arbitration**




## 26.9. Register Description

### 26.9.1. TWI Bit Rate Register

**Name:** TWBR
**Offset:** 0xB8
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TWBR7 | TWBR6 | TWBR5 | TWBR4 | TWBR3 | TWBR2 | TWBR1 | TWBR0 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – TWBRn: TWI Bit Rate Register [n = 7:0]**
TWBR selects the division factor for the bit rate generator. The bit rate generator is a frequency divider which generates the SCL clock frequency in the Master modes.

### 26.9.2. TWI Status Register

**Name:** TWSR
**Offset:** 0xB9
**Reset:** 0xF8
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TWS4 | TWS3 | TWS2 | TWS1 | TWS0 | | TWPS1 | TWPS0 |
| Access | R | R | R | R | R | | R/W | R/W |
| Reset | 1 | 1 | 1 | 1 | 1 | | 0 | 0 |

**Bits 3, 4, 5, 6, 7 – TWSn: TWI Status Bit**
The TWS[7:3] reflect the status of the TWI logic and the 2-wire Serial Bus. The different status codes are described later in this section. Note that the value read from TWSR contains both the 5-bit status value and the 2-bit prescaler value. The application designer should mask the prescaler bits to zero when checking the Status bits. This makes status checking independent of prescaler setting. This approach is used in this datasheet, unless otherwise noted.

**Bits 0, 1 – TWPSn: TWI Prescaler**
These bits can be read and written, and control the bit rate prescaler.

**Table 26-8. TWI Bit Rate Prescaler**

| TWS[1:0] | Prescaler Value |
|---|---|
| 00 | 1 |
| 01 | 4 |
| 10 | 16 |
| 11 | 64 |

To calculate bit rates, refer to Bit Rate Generator Unit. The value of TWPS1...0 is used in the equation.

### 26.9.3. TWI (Slave) Address Register

The TWAR should be loaded with the 7-bit Slave address (in the seven most significant bits of TWAR) to which the TWI will respond when programmed as a Slave Transmitter or Receiver, and not needed in the Master modes. In multi master systems, TWAR must be set in masters which can be addressed as Slaves by other Masters.

The LSB of TWAR is used to enable recognition of the general call address (0x00). There is an associated address comparator that looks for the slave address (or general call address if enabled) in the received serial address. If a match is found, an interrupt request is generated.

**Name:** TWAR
**Offset:** 0xBA
**Reset:** 0xFE
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TWA6 | TWA5 | TWA4 | TWA3 | TWA2 | TWA1 | TWA0 | TWGCE |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

**Bits 1, 2, 3, 4, 5, 6, 7 – TWAn: TWI (Slave) Address**
These seven bits constitute the slave address of the TWI unit.

**Bit 0 – TWGCE: TWI General Call Recognition Enable Bit**
If set, this bit enables the recognition of a General Call given over the 2-wire Serial Bus.

### 26.9.4. TWI Data Register

In Transmit mode, TWDR contains the next byte to be transmitted. In Receive mode, the TWDR contains the last byte received. It is writable while the TWI is not in the process of shifting a byte. This occurs when the TWI Interrupt Flag (TWINT) is set by hardware. Note that the Data Register cannot be initialized by the user before the first interrupt occurs. The data in TWDR remains stable as long as TWINT is set. While data is shifted out, data on the bus is simultaneously shifted in. TWDR always contains the last byte present on the bus, except after a wake up from a sleep mode by the TWI interrupt. In this case, the contents of TWDR is undefined. In the case of a lost bus arbitration, no data is lost in the transition from Master to Slave. Handling of the ACK bit is controlled automatically by the TWI logic, the CPU cannot access the ACK bit directly.

**Name:** TWDR
**Offset:** 0xBB
**Reset:** 0xFF
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TWD7 | TWD6 | TWD5 | TWD4 | TWD3 | TWD2 | TWD1 | TWD0 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Bits 0, 1, 2, 3, 4, 5, 6, 7 – TWDn: TWI Data**

These eight bits constitute the next data byte to be transmitted, or the latest data byte received on the 2-wire Serial Bus.

### 26.9.5. TWI Control Register

The TWCR is used to control the operation of the TWI. It is used to enable the TWI, to initiate a Master access by applying a START condition to the bus, to generate a Receiver acknowledge, to generate a stop condition, and to control halting of the bus while the data to be written to the bus are written to the TWDR. It also indicates a write collision if data is attempted written to TWDR while the register is inaccessible.

**Name:** TWCR
**Offset:** 0xBC
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TWINT | TWEA | TWSTA | TWSTO | TWWC | TWEN | | TWIE |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | | 0 |

**Bit 7 – TWINT: TWI Interrupt Flag**

This bit is set by hardware when the TWI has finished its current job and expects application software response. If the I-bit in SREG and TWIE in TWCR are set, the MCU will jump to the TWI Interrupt Vector. While the TWINT Flag is set, the SCL low period is stretched. The TWINT Flag must be cleared by software by writing a logic one to it.

Note that this flag is not automatically cleared by hardware when executing the interrupt routine. Also note that clearing this flag starts the operation of the TWI, so all accesses to the TWI Address Register (TWAR), TWI Status Register (TWSR), and TWI Data Register (TWDR) must be complete before clearing this flag.

**Bit 6 – TWEA: TWI Enable Acknowledge**

The TWEA bit controls the generation of the acknowledge pulse. If the TWEA bit is written to one, the ACK pulse is generated on the TWI bus if the following conditions are met:

1. The device's own slave address has been received.
2. A general call has been received, while the TWGCE bit in the TWAR is set.
3. A data byte has been received in Master Receiver or Slave Receiver mode.

By writing the TWEA bit to zero, the device can be virtually disconnected from the 2-wire Serial Bus temporarily. Address recognition can then be resumed by writing the TWEA bit to one again.

**Bit 5 – TWSTA: TWI START Condition**

The application writes the TWSTA bit to one when it desires to become a Master on the 2-wire Serial Bus. The TWI hardware checks if the bus is available, and generates a START condition on the bus if it is free. However, if the bus is not free, the TWI waits until a STOP condition is detected, and then generates a new START condition to claim the bus Master status. TWSTA must be cleared by software when the START condition has been transmitted.

**Bit 4 – TWSTO: TWI STOP Condition**

Writing the TWSTO bit to one in Master mode will generate a STOP condition on the 2-wire Serial Bus. When the STOP condition is executed on the bus, the TWSTO bit is cleared automatically. In Slave mode, setting the TWSTO bit can be used to recover from an error condition. This will not generate a

STOP condition, but the TWI returns to a well-defined unaddressed Slave mode and releases the SCL and SDA lines to a high impedance state.

**Bit 3 – TWWC: TWI Write Collision Flag**
The TWWC bit is set when attempting to write to the TWI Data Register – TWDR when TWINT is low. This flag is cleared by writing the TWDR Register when TWINT is high.

**Bit 2 – TWEN: TWI Enable**
The TWEN bit enables TWI operation and activates the TWI interface. When TWEN is written to one, the TWI takes control over the I/O pins connected to the SCL and SDA pins, enabling the slew-rate limiters and spike filters. If this bit is written to zero, the TWI is switched off and all TWI transmissions are terminated, regardless of any ongoing operation.

**Bit 0 – TWIE: TWI Interrupt Enable**
When this bit is written to one, and the I-bit in SREG is set, the TWI interrupt request will be activated for as long as the TWINT Flag is high.

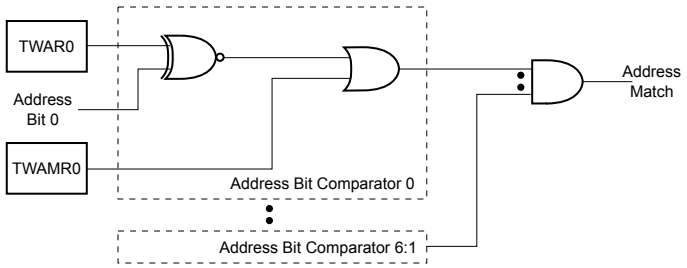
### 26.9.6. TWI (Slave) Address Mask Register

**Name:** TWAMR
**Offset:** 0xBD
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TWAM6 | TWAM5 | TWAM4 | TWAM3 | TWAM2 | TWAM1 | TWAM0 | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Bits 1, 2, 3, 4, 5, 6, 7 – TWAMn: TWI (Slave) Address**
The TWAMR can be loaded with a 7-bit Slave Address mask. Each of the bits in TWAMR can mask (disable) the corresponding address bits in the TWI Address Register (TWAR). If the mask bit is set to one then the address match logic ignores the compare between the incoming address bit and the corresponding bit in TWAR.

**Figure 26-22. TWI Address Match Logic**



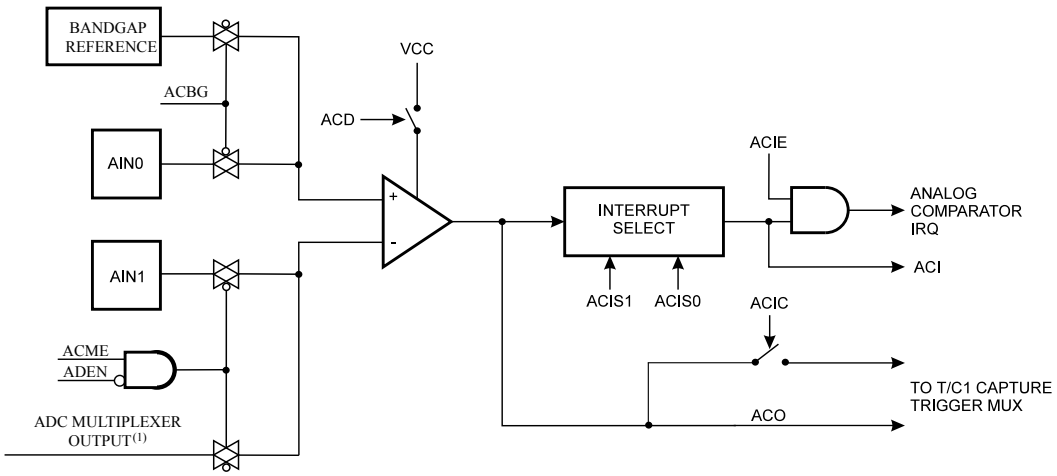
# 27. AC - Analog Comparator

## 27.1. Overview

The Analog Comparator compares the input values on the positive pin AIN0 and negative pin AIN1. When the voltage on the positive pin AIN0 is higher than the voltage on the negative pin AIN1, the Analog Comparator output, ACO, is set. The comparator's output can be set to trigger the Timer/Counter1 Input Capture function. In addition, 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 below.

The Power Reduction ADC bit in the Power Reduction Register (PRR.PRADC) must be written to '0' in order to be able to use the ADC input MUX.

**Figure 27-1. Analog Comparator Block Diagram**




**Note:** Refer to the *Pin Configuration* and the I/O Ports description for Analog Comparator pin placement

**Related Links**

I/O-Ports on page 104

PRR on page 73

Power Management and Sleep Modes on page 65

Minimizing Power Consumption on page 68

## 27.2. Analog Comparator Multiplexed Input

It is possible to select any of the ADC[7:0] pins to replace the negative input to the Analog Comparator. The ADC multiplexer is used to select this input, and consequently, the ADC must be switched off to utilize this feature. If the Analog Comparator Multiplexer Enable bit in the ADC Control and Status Register B (ADCSRB.ACME) is '1' and the ADC is switched off (ADCSRA.ADEN=0), the three least significant Analog Channel Selection bits in the ADC Multiplexer Selection register (ADMUX.MUX[2:0]) select the input pin to replace the negative input to the Analog Comparator, as shown in the table below. When ADCSRB.ACME=0 or ADCSRA.ADEN=1, AIN1 is applied to the negative input of the Analog Comparator.

Table 27-1. Analog Comparator Multiplexed Input

| ACME | ADEN | MUX[2:0] | Analog Comparator Negative Input |
|---|---|---|---|
| 0 | x | xxx | AIN1 |
| 1 | 1 | xxx | AIN1 |
| 1 | 0 | 000 | ADC0 |
| 1 | 0 | 001 | ADC1 |
| 1 | 0 | 010 | ADC2 |
| 1 | 0 | 011 | ADC3 |
| 1 | 0 | 100 | ADC4 |
| 1 | 0 | 101 | ADC5 |
| 1 | 0 | 110 | ADC6 |
| 1 | 0 | 111 | ADC7 |

## 27.3. Register Description

### 27.3.1. ADC Control and Status Register B

**Name:** ADCSRB
**Offset:** 0x7B
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | ACME | | | | ADTS2 | ADTS1 | ADTS0 |
| Access | | R/W | | | | R/W | R/W | R/W |
| Reset | | 0 | | | | 0 | 0 | 0 |

**Bit 6 – ACME: Analog Comparator Multiplexer Enable**
When this bit is written logic one and the ADC is switched off (ADEN in ADCSRA is zero), the ADC multiplexer selects the negative input to the Analog Comparator. When this bit is written logic zero, AIN1 is applied to the negative input of the Analog Comparator. For a detailed description of this bit, see *Analog Comparator Multiplexed Input*..

**Bits 2:0 – ADTSn: ADC Auto Trigger Source [n = 2:0]**
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 27-2. ADC Auto Trigger Source Selection**

| ADTS[2:0] | Trigger Source |
|---|---|
| 000 | Free Running mode |
| 001 | Analog Comparator |
| 010 | External Interrupt Request 0 |
| 011 | Timer/Counter0 Compare Match A |
| 100 | Timer/Counter0 Overflow |
| 101 | Timer/Counter1 Compare Match B |
| 110 | Timer/Counter1 Overflow |
| 111 | Timer/Counter1 Capture Event |

### 27.3.2. Analog Comparator Control and Status Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** ACSR
**Offset:** 0x50
**Reset:** N/A
**Property:** When addressing as I/O Register: address offset is 0x30

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ACD | ACBG | ACO | ACI | ACIE | ACIC | ACIS1 | ACIS0 |
| Access | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 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 the ACD bit, the Analog Comparator Interrupt must be disabled by clearing the ACIE bit in ACSR. Otherwise an interrupt can occur when the bit is changed.

**Bit 6 – ACBG: Analog Comparator Bandgap Select**
When this bit is set, a fixed bandgap reference voltage replaces the positive input to the Analog Comparator. When this bit is cleared, AIN0 is applied to the positive input of the Analog Comparator. When the bandgap reference is used as input to the Analog Comparator, it will take a certain time for the voltage to stabilize. If not stabilized, the first conversion may give a wrong value.

**Bit 5 – ACO: Analog Comparator Output**
The output of the Analog Comparator is synchronized and then directly connected to ACO. 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 the ACIE bit is written logic one and the I-bit in the Status Register is set, the Analog Comparator interrupt is activated. When written logic zero, the interrupt is disabled.

**Bit 2 – ACIC: Analog Comparator Input Capture Enable**
When written logic one, this bit enables the input capture function in Timer/Counter1 to be triggered by the Analog Comparator. The comparator output is in this case 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. When written logic zero, no connection between the Analog Comparator and the input capture function exists. To make the comparator trigger the Timer/Counter1 Input Capture interrupt, the ICIE1 bit in the Timer Interrupt Mask Register (TIMSK1) must be set.

**Bits 1:0 – ACISn: Analog Comparator Interrupt Mode Select [n = 1:0]**

These bits determine which comparator events that trigger the Analog Comparator interrupt.

**Table 27-3. ACIS[1:0] 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 the ACIS1/ACIS0 bits, the Analog Comparator Interrupt must be disabled by clearing its Interrupt Enable bit in the ACSR Register. Otherwise an interrupt can occur when the bits are changed.

### 27.3.3. Digital Input Disable Register 1

**Name:** DIDR1
**Offset:** 0x7F
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | AIN1D | AIN0D |
| Access | | | | | | | R/W | R/W |
| Reset | | | | | | | 0 | 0 |

**Bit 1 – AIN1D: AIN1 Digital Input Disable**

**Bit 0 – AIN0D: AIN0 Digital Input Disable**
When this bit is written logic one, the digital input buffer on the AIN1/0 pin is disabled. The corresponding PIN Register bit will always read as zero when this bit is set. When an analog signal is applied to the AIN1/0 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.

# 28. ADC - Analog to Digital Converter

## 28.1. Features

- 10-bit Resolution
- 0.5 LSB Integral Non-Linearity
- ±2 LSB Absolute Accuracy
- 13 - 260µs Conversion Time
- Up to 76.9kSPS (Up to 15kSPS at Maximum Resolution)
- Six Multiplexed Single Ended Input Channels
- Two Additional Multiplexed Single Ended Input Channels (TQFP and VFQFN Package only)
- Temperature Sensor Input Channel
- Optional Left Adjustment for ADC Result Readout
- 0 - $V_{CC}$ ADC Input Voltage Range
- Selectable 1.1V ADC Reference Voltage
- Free Running or Single Conversion Mode
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceler

## 28.2. Overview

The device features a 10-bit successive approximation ADC. The ADC is connected to an 8-channel Analog Multiplexer which allows eight single-ended voltage inputs constructed from the pins of Port A. The single-ended voltage inputs refer to 0V (GND).

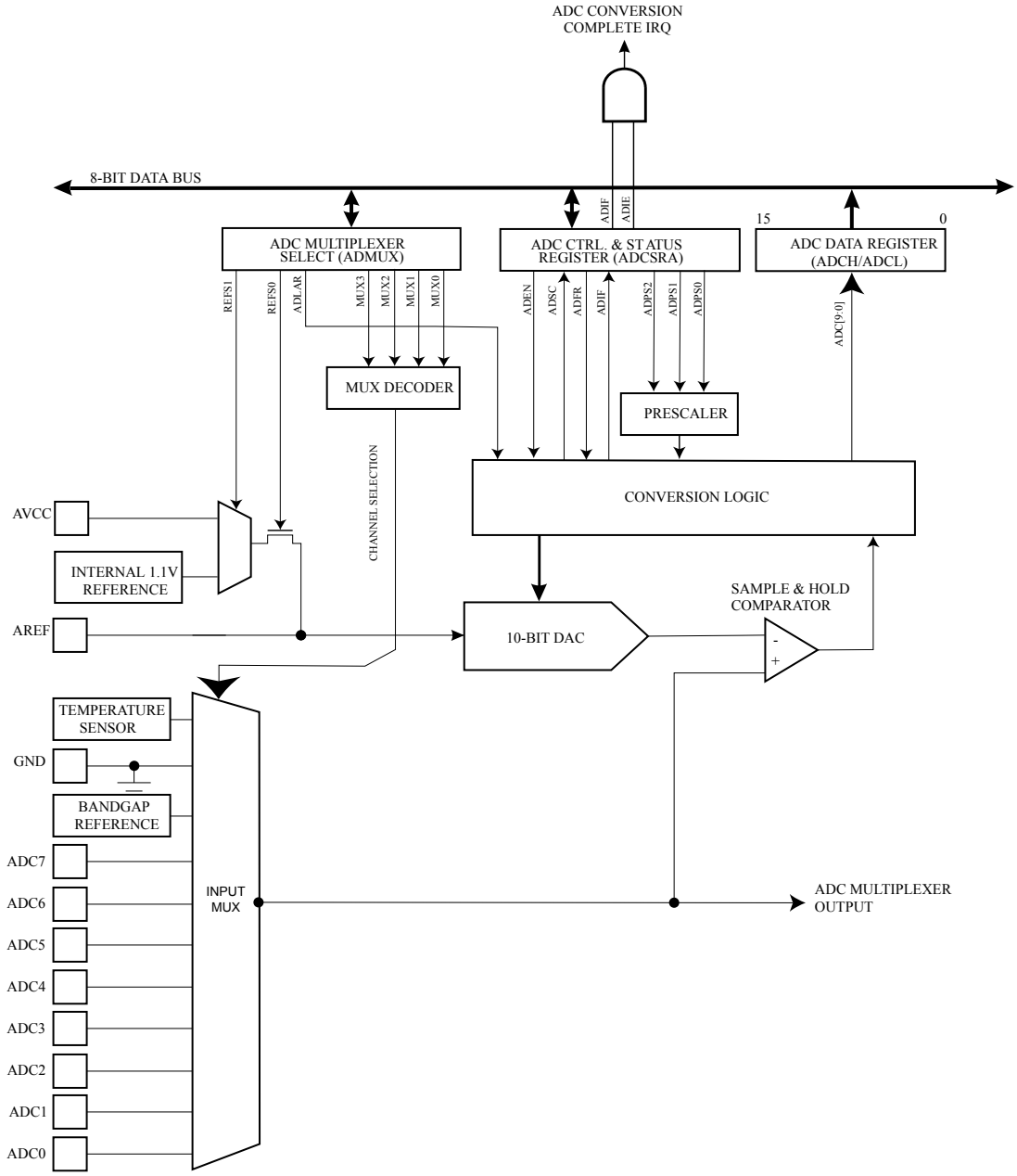
The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A block diagram of the ADC is shown below.

The ADC has a separate analog supply voltage pin, $AV_{CC}$. $AV_{CC}$ must not differ more than ±0.3V from $V_{CC}$. See section ADC Noise Canceler on how to connect this pin.

The Power Reduction ADC bit in the Power Reduction Register (PRR.PRADC) must be written to '0' in order to be enable the ADC.

The ADC converts an analog input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. Optionally, $AV_{CC}$ or an internal 1.1V reference voltage may be connected to the AREF pin by writing to the REFSn bits in the ADMUX Register. The internal voltage reference must be decoupled by an external capacitor at the AREF pin to improve noise immunity.

**Figure 28-1. Analog to Digital Converter Block Schematic Operation**




The analog input channel is selected by writing to the MUX bits in the ADC Multiplexer Selection register ADMUX.MUX[3:0]. Any of the ADC input pins, as well as GND and a fixed bandgap voltage reference, can be selected as single ended inputs to the ADC. The ADC is enabled by writing a '1' to the ADC Enable bit in the ADC Control and Status Register A (ADCSRA.ADEN). Voltage reference and input channel selections will not take 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 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 ADC Left Adjust Result bit ADMUX.ADLAR.

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, 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 second conversion completes before ADCH is read, neither register is updated and the result from the second 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.

**Related Links**

Power Management and Sleep Modes on page 65
Power Reduction Register on page 67

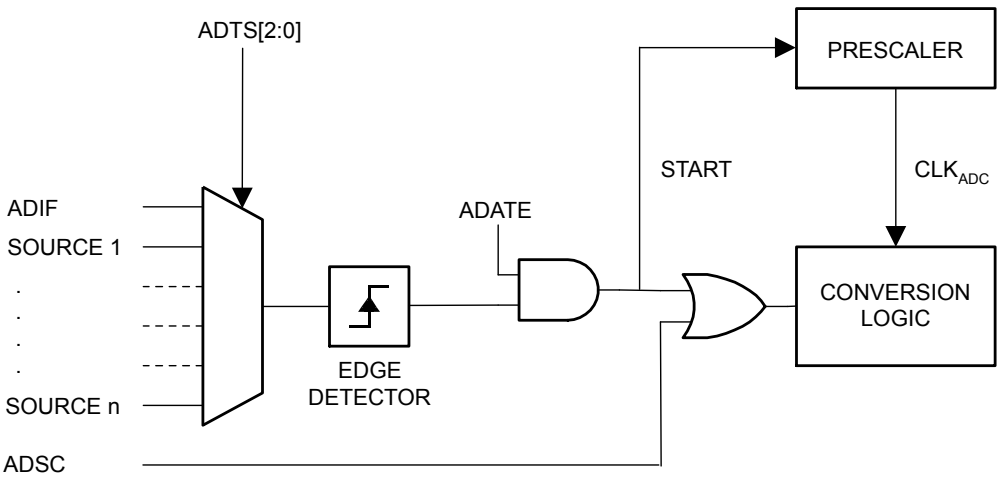
## 28.3. Starting a Conversion

A single conversion is started by writing a '0' to the Power Reduction ADC bit in the Power Reduction Register (PRR.PRADC), and writing a '1' to the ADC Start Conversion bit in the ADC Control and Status Register A (ADCSRA.ADSC). ADCS will stay 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 (ADCSRA.ADATE). The trigger source is selected by setting the ADC Trigger Select bits in the ADC Control and Status Register B (ADCSRB.ADTS). See the description of the ADCSRB.ADTS for a list of available 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 the AVR Status REgister (SREG.I) 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 28-2. ADC Auto Trigger Logic**



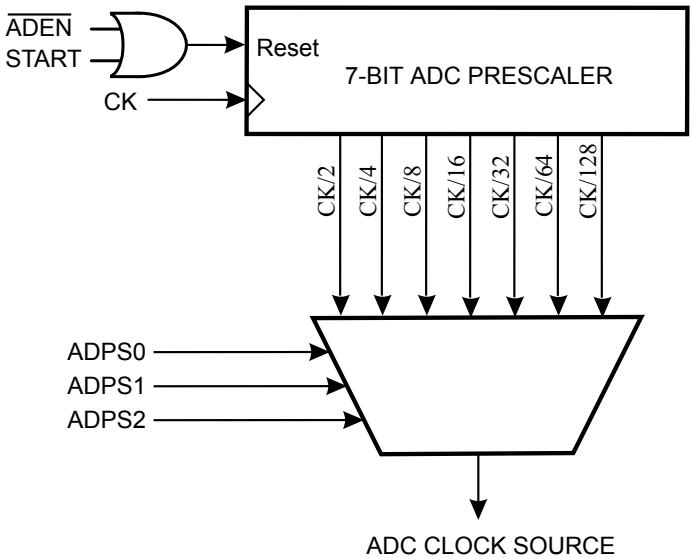

Using the ADC Interrupt Flag as a trigger source makes the ADC start a new conversion as soon as the ongoing conversion has finished. The ADC then operates in Free Running mode, constantly sampling and updating the ADC Data Register. The first conversion must be started by writing a '1' to

ADCSRA.ADSC. 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 ADCSRA.ADSC to '1'. ADSC can also be used to determine if a conversion is in progress. The ADSC bit will be read as '1' during a conversion, independently of how the conversion was started.

## 28.4. Prescaling and Conversion Timing

**Figure 28-3. ADC Prescaler**




By default, the successive approximation circuitry requires an input clock frequency between 50kHz and 200kHz to get maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200kHz to get a higher sample rate.

The ADC module contains a prescaler, which generates an acceptable ADC clock frequency from any CPU frequency above 100kHz. The prescaling is selected by the ADC Prescaler Select bits in the ADC Control and Status Register A (ADCSRA.ADPS). The prescaler starts counting from the moment the ADC is switched on by writing the ADC Enable bit ADCSRA.ADEN to '1'. The prescaler keeps running for as long as ADEN=1, and is continuously reset when ADEN=0.

When initiating a single ended conversion by writing a '1' to the ADC Start Conversion bit (ADCSRA.ADSC), the conversion starts at the following rising edge of the ADC clock cycle.

A normal conversion takes 13 ADC clock cycles. The first conversion after the ADC is switched on (i.e., ADCSRA.ADEN is written to '1') takes 25 ADC clock cycles in order to initialize the analog circuitry.

When the bandgap reference voltage is used as input to the ADC, it will take a certain time for the voltage to stabilize. If not stabilized, the first value read after the first conversion may be wrong.

The actual sample-and-hold takes place 1.5 ADC clock cycles after the start of a normal conversion and 13.5 ADC clock cycles after the start of an first conversion. When a conversion is complete, the result is written to the ADC Data Registers (ADCL and ADCH), and the ADC Interrupt Flag (ADCSRA.ADIF) is set. In Single Conversion mode, ADCSRA.ADSC is cleared simultaneously. The software may then set ADCSRA.ADSC again, and a new conversion will be initiated on the first rising ADC clock edge.

When Auto Triggering is used, the prescaler is reset when the trigger event occurs. This assures a fixed delay from the trigger event to the start of conversion. In this mode, the sample-and-hold takes place two

ADC clock cycles after the rising edge on the trigger source signal. Three additional CPU clock cycles are used for synchronization logic.

In Free Running mode, a new conversion will be started immediately after the conversion completes, while ADCRSA.ADSC remains high. See also the ADC Conversion Time table below.

**Figure 28-4. ADC Timing Diagram, First Conversion (Single Conversion Mode)**

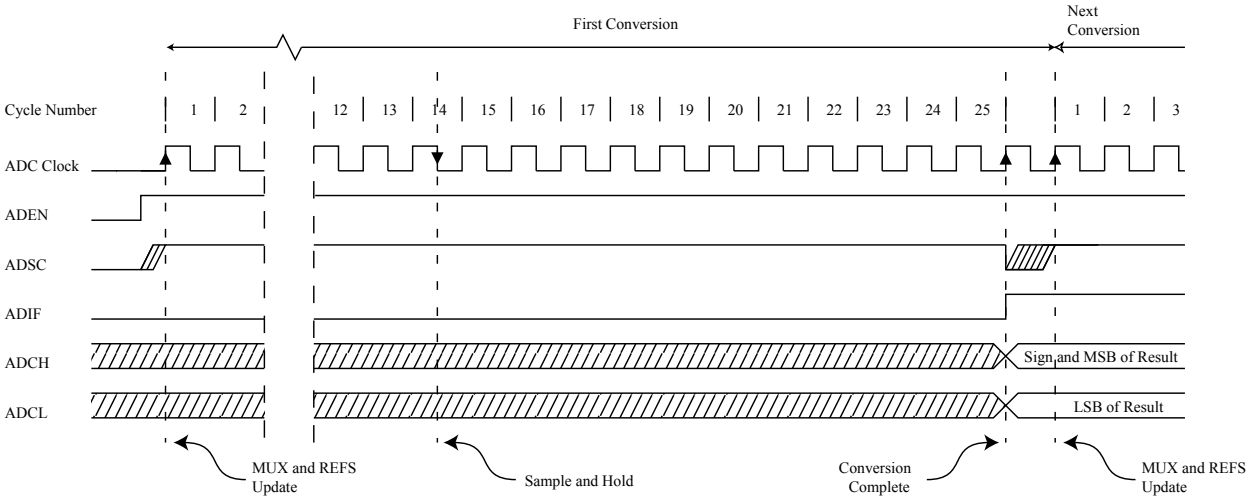



**Figure 28-5. ADC Timing Diagram, Single Conversion**

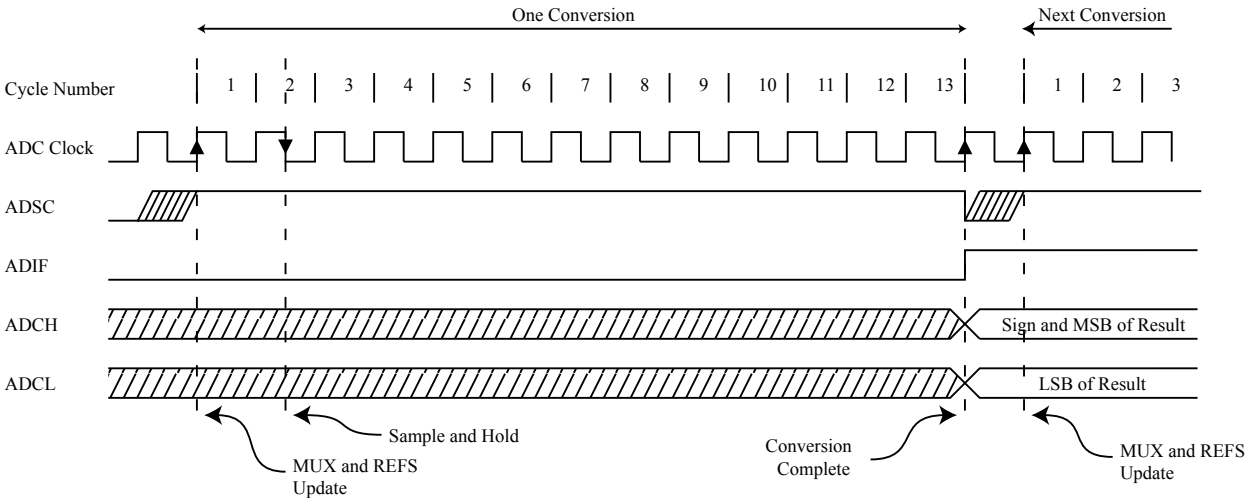


**Figure 28-6. ADC Timing Diagram, Auto Triggered Conversion**

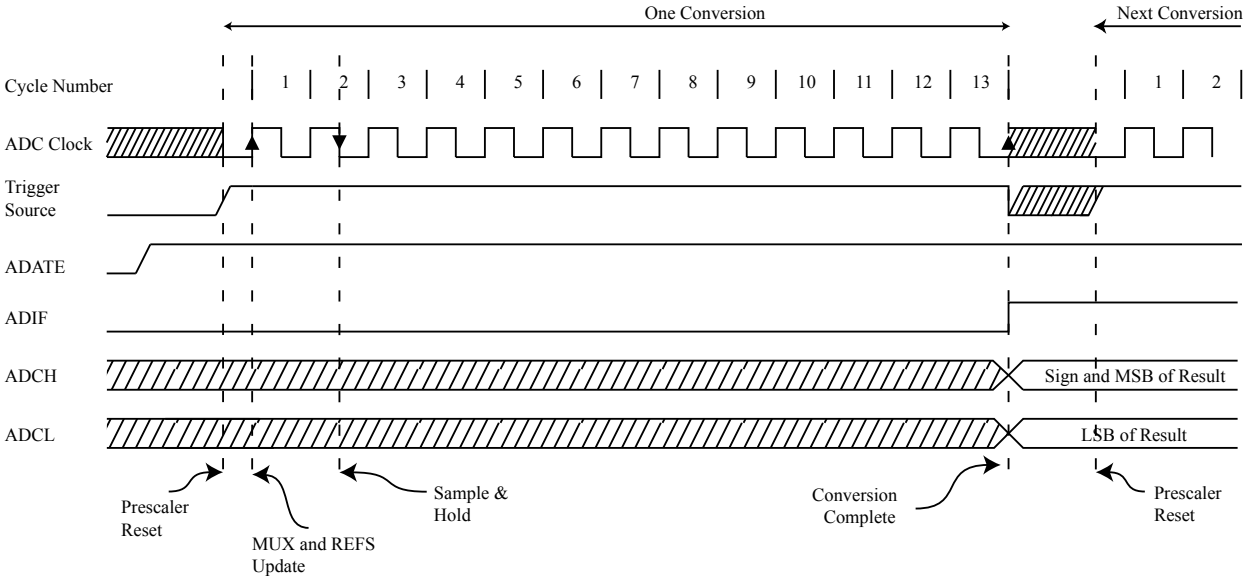



**Figure 28-7. ADC Timing Diagram, Free Running Conversion**

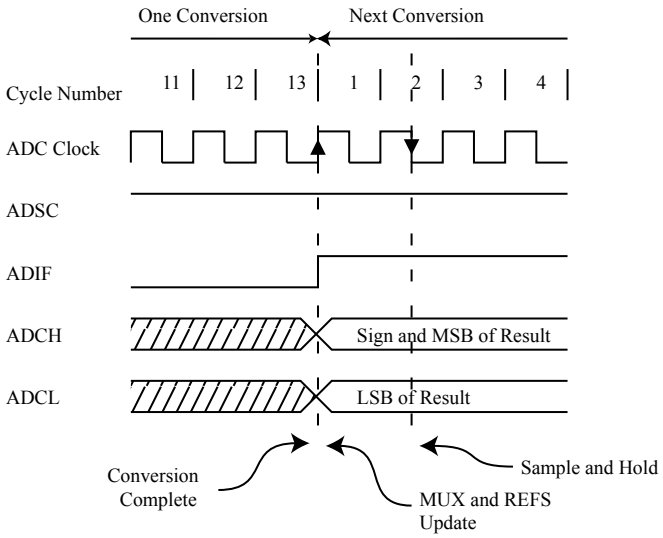



**Table 28-1. ADC Conversion Time**

| Condition | Sample & Hold (Cycles from Start of Conversion) | Conversion Time (Cycles) |
|---|---|---|
| First conversion | 13.5 | 25 |
| Normal conversions, single ended | 1.5 | 13 |
| Auto Triggered conversions | 2 | 13.5 |

## 28.5. Changing Channel or Reference Selection

The Analog Channel Selection bits (MUX) and the Reference Selection bits (REFS) bits in the ADC Multiplexer Selection Register (ADMUX.MUX[3:0] and ADMUX.REFS[1:0]) 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 (indicated by ADCSRA.ADIF 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 to ADMUX until one ADC clock cycle after the ADC Start Conversion bit (ADCRSA.ADSC) was written.

If Auto Triggering is used, the exact time of the triggering event can be indeterministic. Special care must be taken when updating the ADMUX Register, in order to control which conversion will be affected by the new settings.

If both the ADC Auto Trigger Enable and ADC Enable bits (ADCRSA.ADATE, ADCRSA.ADEN) are written to '1', an interrupt event can occur at any time. If the ADMUX Register is changed in this period, the user cannot tell if the next conversion is based on the old or the new settings. ADMUX can be safely updated in the following ways:

1. When ADATE or ADEN is cleared.
    1.1. During conversion, minimum one ADC clock cycle after the trigger event.
    1.2. After a conversion, before the Interrupt Flag used as trigger source is cleared.

When updating ADMUX in one of these conditions, the new settings will affect the next ADC conversion.

### 28.5.1. 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.

The user is advised not to write new channel or reference selection values during Free Running mode.

### 28.5.2. ADC Voltage Reference

The reference voltage for the ADC ($V_{REF}$) 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 $AV_{CC}$, internal 1.1V reference, or external AREF pin.

$AV_{CC}$ is connected to the ADC through a passive switch. The internal 1.1V reference is generated from the internal bandgap reference ($V_{BG}$) through an internal amplifier. In either case, the external AREF pin is directly connected to the ADC, and the reference voltage can be made more immune to noise by connecting a capacitor between the AREF pin and ground. $V_{REF}$ can also be measured at the AREF pin with a high impedance voltmeter. Note that $V_{REF}$ is a high impedance source, and only a capacitive load should be connected in a system.

If the user has a fixed voltage source connected to the AREF pin, the user may not use the other reference voltage options in the application, as they will be shorted to the external voltage. If no external voltage is applied to the AREF pin, the user may switch between $AV_{CC}$ and 1.1V as reference selection. The first ADC conversion result after switching reference voltage source may be inaccurate, and the user is advised to discard this result.

## 28.6. ADC Noise Canceler

The ADC features a noise canceler that enables conversion during sleep mode to reduce 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:

1. 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.
2. Enter ADC Noise Reduction mode (or Idle mode). The ADC will start a conversion once the CPU has been halted.
3. 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:** The ADC will not be automatically turned off when entering other sleep modes than Idle mode and ADC Noise Reduction mode. The user is advised to write zero to ADCRSA.ADEN before entering such sleep modes to avoid excessive power consumption.

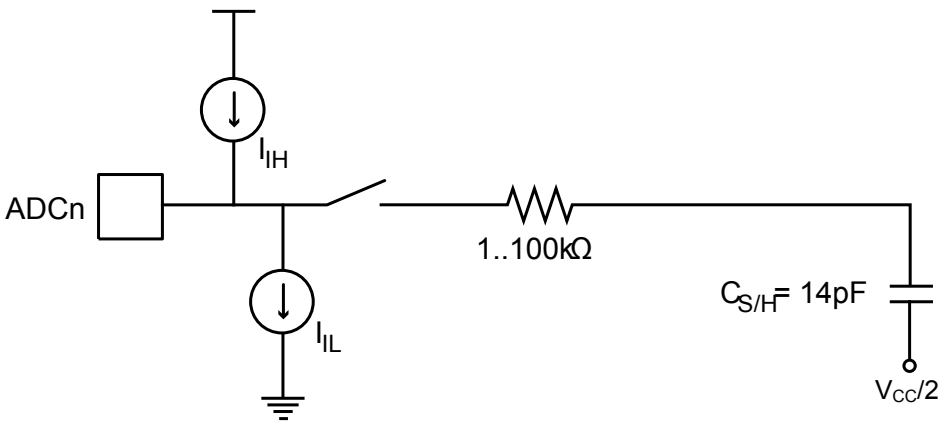
### 28.6.1. Analog Input Circuitry

The analog input circuitry for single ended channels is illustrated below. 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 10 kΩ 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, with 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.

Signal components higher than the Nyquist frequency ($f_{ADC}/2$) should not be present for either kind of channels, to avoid distortion from unpredictable signal convolution. The user is advised to remove high frequency components with a low-pass filter before applying the signals as inputs to the ADC.

**Figure 28-8. Analog Input Circuitry**



### 28.6.2. Analog Noise Canceling Techniques

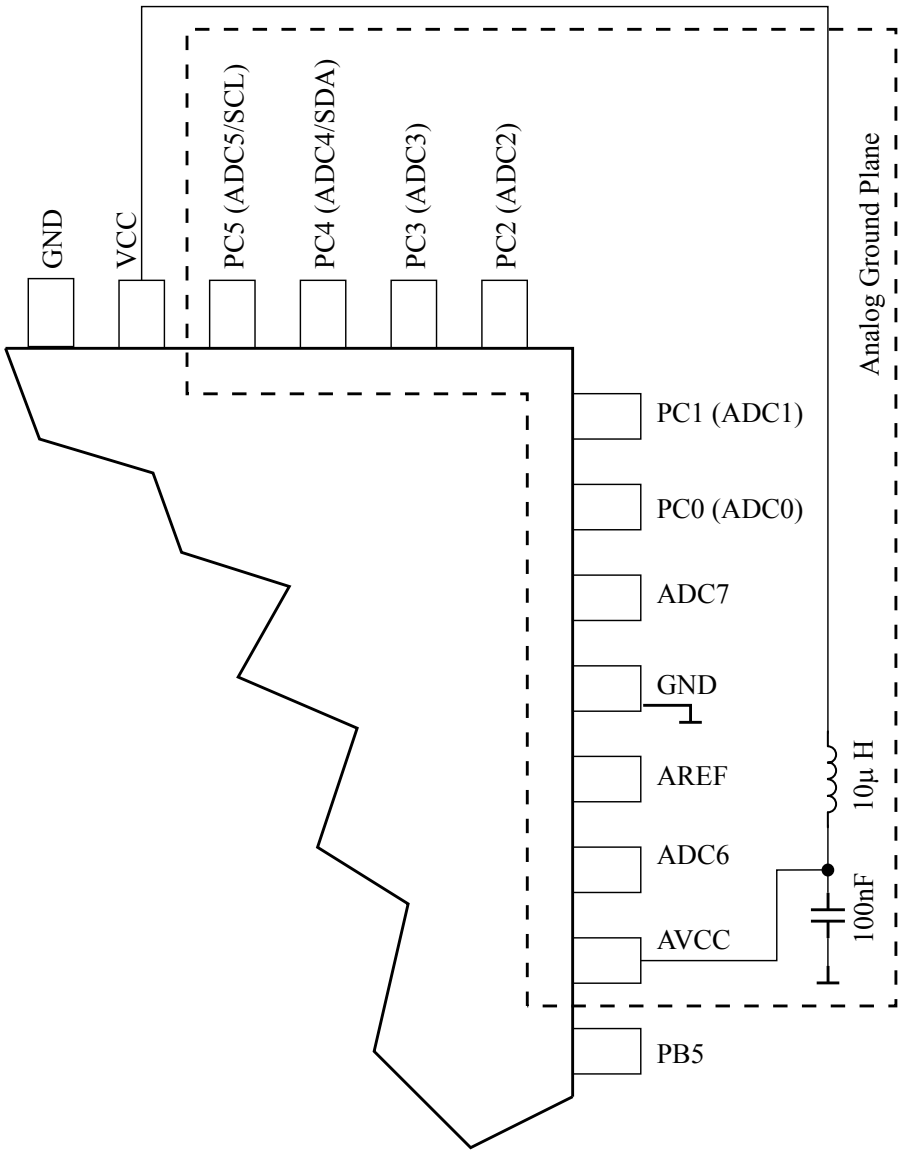
Digital circuitry inside and outside the device generates EMI which might affect the accuracy of analog measurements. If conversion accuracy is critical, the noise level can be reduced by applying the following techniques:

1. Keep analog signal paths as short as possible. Make sure analog tracks run over the analog ground plane, and keep them well away from high-speed switching digital tracks.
   1.1. The $AV_{CC}$ pin on the device should be connected to the digital $V_{CC}$ supply voltage via an LC network as shown in the figure below.

   1.2. Use the ADC noise canceler function to reduce induced noise from the CPU.

   1.3. If any ADC [3:0] port pins are used as digital outputs, it is essential that these do not switch while a conversion is in progress. However, using the 2-wire Interface (ADC4 and ADC5) will only affect the conversion on ADC4 and ADC5 and not the other ADC channels.

**Figure 28-9. ADC Power Connections**



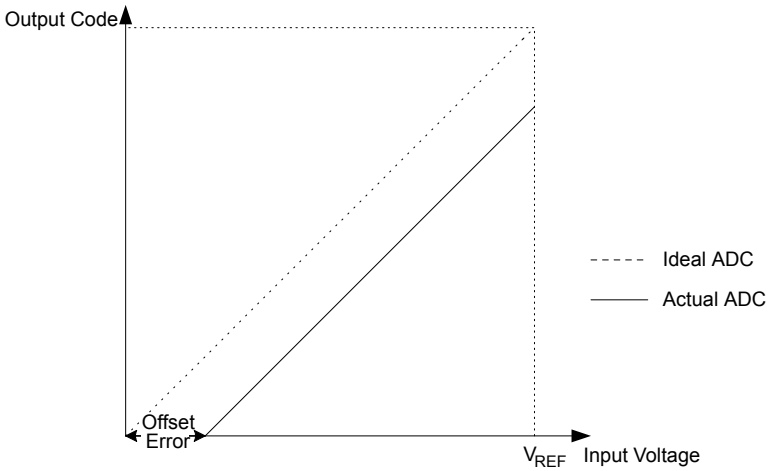
### 28.6.3. ADC Accuracy Definitions

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:

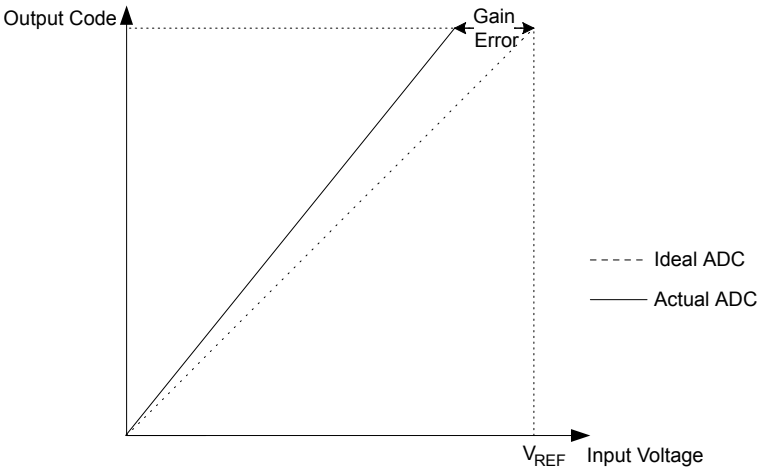
- Offset: The deviation of the first transition (0x000 to 0x001) compared to the ideal transition (at 0.5 LSB). Ideal value: 0 LSB.

**Figure 28-10. Offset Error**



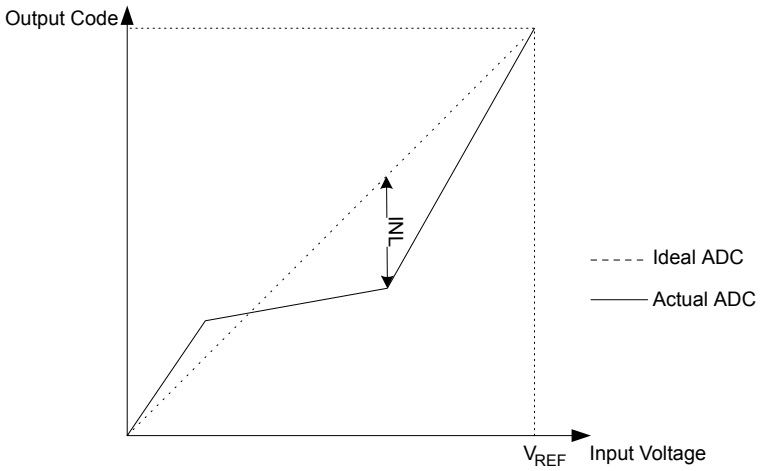

- Gain error: After adjusting for offset, the gain error is found as the deviation of the last transition (0x3FE to 0x3FF) compared to the ideal transition (at 1.5 LSB below maximum). Ideal value: 0 LSB.

**Figure 28-11. Gain Error**



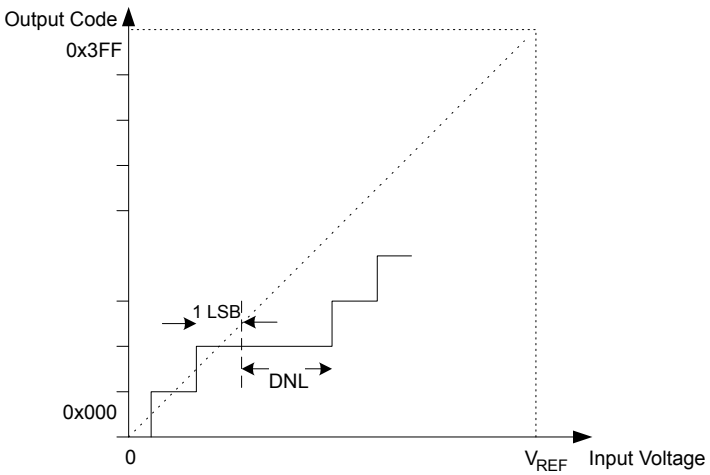

- Integral Non-linearity (INL): After adjusting for offset and gain error, the INL is the maximum deviation of an actual transition compared to an ideal transition for any code. Ideal value: 0 LSB.

**Figure 28-12. 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 28-13. 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.

## 28.7. ADC Conversion Result

After the conversion is complete (ADCSRA.ADIF is set), the conversion result can be found in the ADC Result Registers (ADCL, ADCH).

For single ended conversion, the result is

$$\mathrm{ADC} = \frac{V_{\mathrm{IN}} \cdot 1024}{V_{\mathrm{REF}}}$$

where $V_{IN}$ is the voltage on the selected input pin, and $V_{REF}$ the selected voltage reference (see also descriptions of ADMUX.REFSn and ADMUX.MUX). 0x000 represents analog ground, and 0x3FF represents the selected reference voltage minus one LSB.

*Example:*

ADMUX = 0xED (ADC3 - ADC2, 10× gain, 2.56V reference, left adjusted result)

Voltage on ADC3 is 300 mV, voltage on ADC2 is 500 mV.

ADCR = 512 × 10 × (300 - 500) / 2560 = -400 = 0x270

ADCL will thus read 0x00, and ADCH will read 0x9C.

Writing zero to ADLAR right adjusts the result: ADCL = 0x70, ADCH = 0x02.

## 28.8. Temperature Measurement

The temperature measurement is based on an on-chip temperature sensor that is coupled to a single ended Temperature sensor channel. Selecting the Temperature sensor channel by writing ADMUX.MUX[3:0] to '1000' enables the temperature sensor. The internal 1.1V voltage reference must also be selected for the ADC voltage 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 the following table. The voltage sensitivity is approximately 1mV/°C, the accuracy of the temperature measurement is ±10°C.

**Table 28-2. Temperature vs. Sensor Output Voltage (Typical Case)**

| Temperature | -45°C | +25°C | +85°C |
|---|---|---|---|
| Voltage | 242mV | 314mV | 380mV |

The values described in the table above are typical values. However, due to process variations 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 software calibration requires that a calibration value is measured and stored in a register or EEPROM for each chip, as a part of the production test. The software calibration can be done utilizing the formula:

$T = \{ [(ADCH << 8) | ADCL] - T_{OS} \} / k$

where ADCH and ADCL are the ADC data registers, $k$ is a fixed coefficient and $T_{OS}$ is the temperature sensor offset value determined and stored into EEPROM as a part of the production test.

## 28.9. Register Description

### 28.9.1. ADC Multiplexer Selection Register

**Name:** ADMUX
**Offset:** 0x7C
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | REFS1 | REFS0 | ADLAR | | MUX3 | MUX2 | MUX1 | MUX0 |
| Access | R/W | R/W | R/W | | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | | 0 | 0 | 0 | 0 |

**Bits 7:6 – REFSn: Reference Selection [n = 1:0]**
These bits select the voltage reference for the ADC. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

**Table 28-3. ADC Voltage Reference Selection**

| REFS[1:0] | Voltage Reference Selection |
|---|---|
| 00 | AREF, Internal $V_{ref}$ turned off |
| 01 | $AV_{CC}$ with external capacitor at AREF pin |
| 10 | Reserved |
| 11 | Internal 1.1V Voltage Reference with external capacitor at AREF pin |

**Bit 5 – 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 complete description of this bit, see ADCL and ADCH.

**Bits 3:0 – MUXn: Analog Channel Selection [n = 3:0]**
The value of these bits selects which analog inputs are connected to the ADC. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set).

**Table 28-4. Input Channel Selection**

| MUX[3:0] | Single Ended Input |
|---|---|
| 0000 | ADC0 |
| 0001 | ADC1 |
| 0010 | ADC2 |
| 0011 | ADC3 |
| 0100 | ADC4 |
| 0101 | ADC5 |

| MUX[3:0] | Single Ended Input |
|---|---|
| 0110 | ADC6 |
| 0111 | ADC7 |
| 1000 | Temperature sensor |
| 1001 | Reserved |
| 1010 | Reserved |
| 1011 | Reserved |
| 1100 | Reserved |
| 1101 | Reserved |
| 1110 | 1.1V ($V_{BG}$) |
| 1111 | 0V (GND) |

### 28.9.2. ADC Control and Status Register A

**Name:** ADCSRA
**Offset:** 0x7A
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ADEN | ADSC | ADATE | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 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 25 ADC clock cycles instead of the normal 13. 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. This also applies if the SBI and CBI instructions are used.

**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 – ADPSn: ADC Prescaler Select [n = 2:0]**
These bits determine the division factor between the system clock frequency and the input clock to the ADC.

**Table 28-5. Input Channel Selection**

| ADPS[2:0] | Division Factor |
|---|---|
| 000 | 2 |
| 001 | 2 |

| ADPS[2:0] | Division Factor |
|---|---|
| 010 | 4 |
| 011 | 8 |
| 100 | 16 |
| 101 | 32 |
| 110 | 64 |
| 111 | 128 |

### 28.9.3. ADC Data Register Low (ADLAR=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 MUXn bits in ADMUX 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.

**Name:** ADCL
**Offset:** 0x78
**Reset:** 0x00
**Property:** ADLAR = 0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ADC7 | ADC6 | ADC5 | ADC4 | ADC3 | ADC2 | ADC1 | ADC0 |
| Access | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – ADCn: ADC Conversion Result [n = 7:0]**
These bits represent the result from the conversion. Refer to ADC Conversion Result for details.

### 28.9.4. ADC Data Register High (ADLAR=0)

**Name:** ADCH
**Offset:** 0x79
**Reset:** 0x00
**Property:** ADLAR = 0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | ADC9 | ADC8 |
| Access | | | | | | | R | R |
| Reset | | | | | | | 0 | 0 |

**Bit 1 – ADC9: ADC Conversion Result**
Refer to ADCL.

**Bit 0 – ADC8: ADC Conversion Result**

### 28.9.5. ADC Data Register Low (ADLAR=1)

**Name:** ADCL
**Offset:** 0x78
**Reset:** 0x00
**Property:** ADLAR = 1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ADC1 | ADC0 | | | | | | |
| Access | R | R | | | | | | |
| Reset | 0 | 0 | | | | | | |

**Bit 7 – ADC1: ADC Conversion Result**
Refer to ADCL.

**Bit 6 – ADC0: ADC Conversion Result**

### 28.9.6. ADC Data Register High (ADLAR=1)

**Name:** ADCH
**Offset:** 0x79
**Reset:** 0x00
**Property:** ADLAR = 1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ADC9 | ADC8 | ADC7 | ADC6 | ADC5 | ADC4 | ADC3 | ADC2 |
| Access | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bit 7 – ADC9: ADC Conversion Result 9**
Refer to ADCL.

**Bit 6 – ADC8: ADC Conversion Result 8**
Refer to ADCL.

**Bit 5 – ADC7: ADC Conversion Result 7**
Refer to ADCL.

**Bit 4 – ADC6: ADC Conversion Result 6**
Refer to ADCL.

**Bit 3 – ADC5: ADC Conversion Result 5**
Refer to ADCL.

**Bit 2 – ADC4: ADC Conversion Result 4**
Refer to ADCL.

**Bit 1 – ADC3: ADC Conversion Result 3**
Refer to ADCL.

**Bit 0 – ADC2: ADC Conversion Result 2**
Refer to ADCL.

### 28.9.7. ADC Control and Status Register B

**Name:** ADCSRB
**Offset:** 0x7B
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | ACME | | | | ADTS2 | ADTS1 | ADTS0 |
| Access | | R/W | | | | R/W | R/W | R/W |
| Reset | | 0 | | | | 0 | 0 | 0 |

**Bit 6 – ACME: Analog Comparator Multiplexer Enable**
When this bit is written logic one and the ADC is switched off (ADEN in ADCSRA is zero), the ADC multiplexer selects the negative input to the Analog Comparator. When this bit is written logic zero, AIN1 is applied to the negative input of the Analog Comparator. For a detailed description of this bit, see *Analog Comparator Multiplexed Input*..

**Bits 2:0 – ADTSn: ADC Auto Trigger Source [n = 2:0]**
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 28-6. ADC Auto Trigger Source Selection**

| ADTS[2:0] | Trigger Source |
|---|---|
| 000 | Free Running mode |
| 001 | Analog Comparator |
| 010 | External Interrupt Request 0 |
| 011 | Timer/Counter0 Compare Match A |
| 100 | Timer/Counter0 Overflow |
| 101 | Timer/Counter1 Compare Match B |
| 110 | Timer/Counter1 Overflow |
| 111 | Timer/Counter1 Capture Event |

### 28.9.8. Digital Input Disable Register 0

When the respective bits are written to logic one, the digital input buffer on the corresponding ADC pin is disabled. The corresponding PIN Register bit will always read as zero when this bit is set. When an analog signal is applied to the ADC7...0 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.

**Name:** DIDR0
**Offset:** 0x7E
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ADC7D | ADC6D | ADC5D | ADC4D | ADC3D | ADC2D | ADC1D | ADC0D |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 0, 1, 2, 3, 4, 5, 6, 7 – ADC0D, ADC1D, ADC2D, ADC3D, ADC4D, ADC5D, ADC6D, ADC7D: ADC Digital Input Disable**

# 29. DBG - debugWIRE On-chip Debug System

## 29.1. 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

## 29.2. Overview

The debugWIRE On-chip debug system uses a wire with bi-directional interface to control the program flow and execute AVR instructions in the CPU and to program the different non-volatile memories.

## 29.3. Physical Interface

When the debugWIRE Enable (DWEN) bit is programmed to '0' and Lock bits are unprogrammed ('1'), 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 29-1. The debugWIRE Setup**

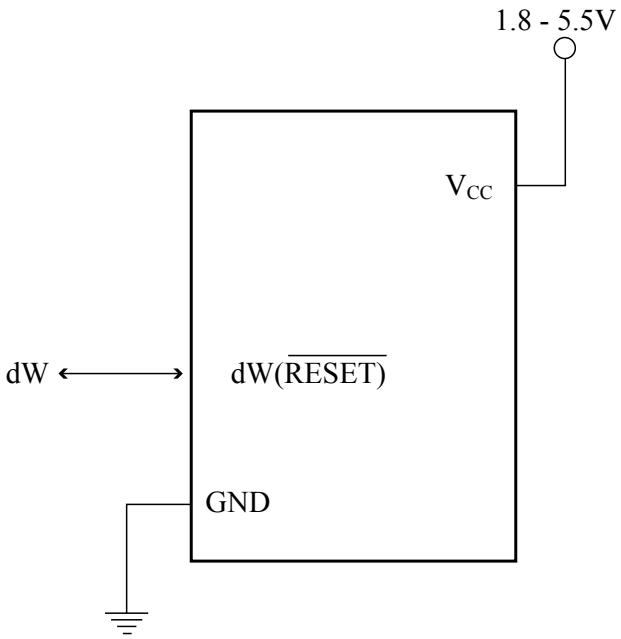


Figure. The debugWIRE Setup shows the schematic of a target MCU, with debugWIRE enabled, and the emulator connector. The system clock is not affected by debugWIRE and will always be the clock source selected by the CKSEL Fuses.

When designing a system where debugWIRE will be used, the following observations must be made for correct operation:

- Pull-up resistors on the dW/(RESET) line must not be smaller than 10kΩ. The pull-up resistor is not required for debugWIRE functionality
- Connecting the RESET pin directly to $V_{CC}$ will not work.
- Capacitors connected to the RESET pin must be disconnected when using debugWire.
- All external reset sources must be disconnected.

## 29.4. Software Break Points

debugWIRE supports Break Points function in Program Memory by the AVR Break instruction. Setting a break point in Atmel 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 when a Break Point is changed. This is automatically handled by Atmel Studio through the debugWIRE interface. The use of Break Points will therefore reduce the Flash Data retention. Devices used for debugging purposes should not be shipped to end customers.

## 29.5. 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.

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.

## 29.6. Register Description

The following section describes the registers used with the debugWire.

### 29.6.1. debugWire Data Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** DWDR
**Offset:** 0x51
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x31

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DWDR[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – DWDR[7:0]: debugWire Data**

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.

# 30. Self-Programming the Flash

## 30.1. Overview

In ATmega48A, there is no Read-While-Write support and no separate Boot Loader Section. The SPM instruction can be executed from the entire Flash.

The device provides a Self-Programming mechanism for downloading and uploading program code by the MCU itself. The Self-Programming can use any available data interface and associated protocol to read code and write (program) that code into the Program Memory.

The 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 with 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:

**Alternative 1, fill the buffer before a Page Erase**

- Fill temporary page buffer
- Perform a Page Erase
- Perform a Page Write

**Alternative 2, fill the buffer after Page Erase**

- Perform a Page Erase
- Fill temporary page buffer
- Perform a Page Write

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. 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. 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.

### 30.1.1. Performing Page Erase by Store Program Memory (SPM)

To execute Page Erase, set up the address in the Z-pointer (R30 and R31), write “0x00000011” to Store Program Memory Control and Status Register (SPMCSR) and execute Store Program Memory (SPM) within four clock cycles after writing SPMCSR. The data in R1 and R0 is ignored. The page address must be written to PCPAGE ([Z12:Z6]) in the Z-register. Other bits in the Z-pointer will be ignored during this operation.

- The CPU is halted during the Page Erase operation
  **Note:** If an interrupt occurs in the time sequence the four cycle access cannot be guaranteed. In order to ensure atomic operation you should disable interrupts before writing to SPMCSR.

### 30.1.2. Filling the Temporary Buffer (Page Loading)

To write an instruction word, set up the address in the Z-pointer (R30 and R31) and data in R1:R0, write “0x00000001” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The content of PCWORD ([Z5:Z1]) 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.

### 30.1.3. Performing a Page Write

To execute Page Write, set up the address in the Z-pointer(R30 and R31), write “0x00000101” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The data in R1 and R0 is ignored. The page address must be written to PCPAGE ([Z12:Z6]). Other bits in the Z-pointer must be written to zero during this operation.

- The CPU is halted during the Page Write operation

## 30.2. Addressing the Flash During Self-Programming

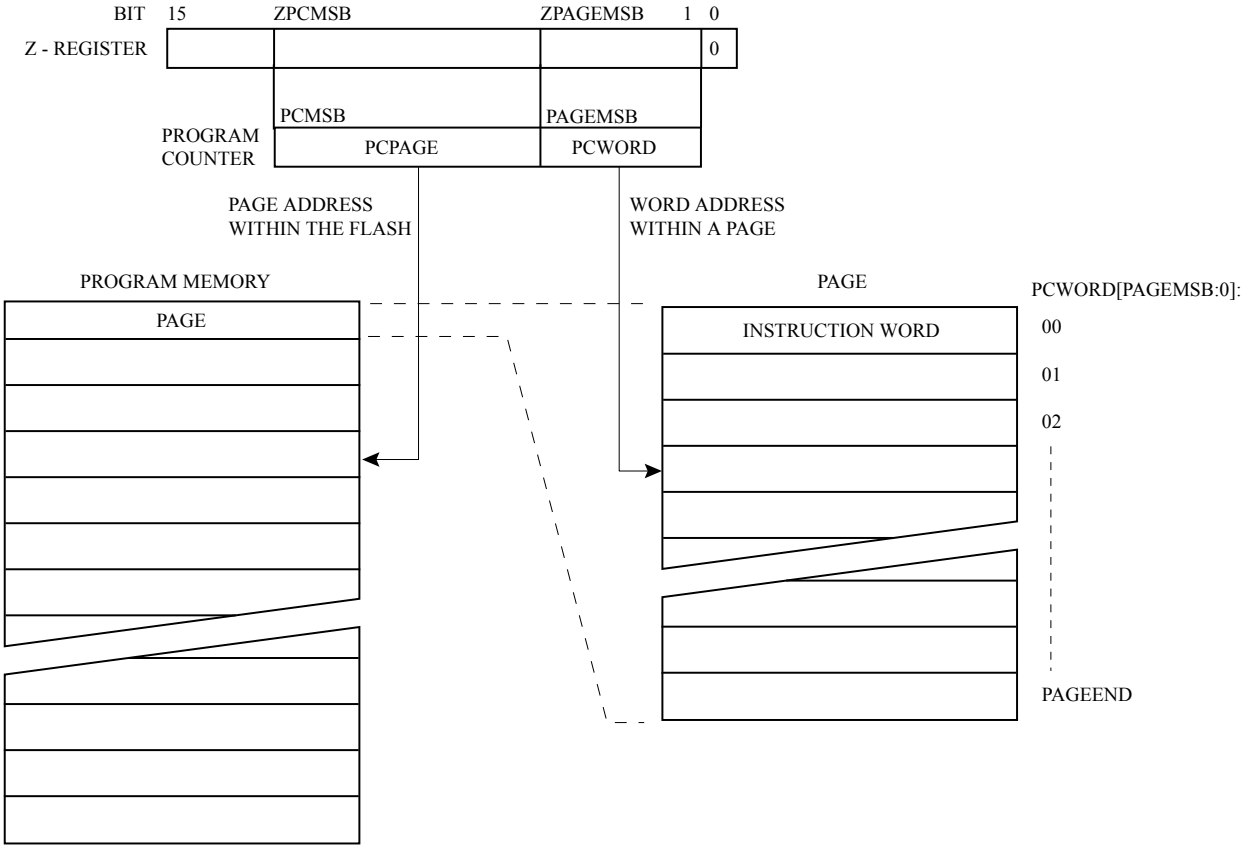
The Z-pointer (R30 and R31) is used to address the SPM commands.

| 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 (Please refer to *Page Size* section in Memory Programming chapter), 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 the following figure. Note that the Page Erase and Page Write operations are addressed independently. Therefore it is of major importance that the software addresses the same page in both the Page Erase and Page Write operation.

The Load Program Memory (LPM) instruction uses the Z-pointer to store the address. Since this instruction addresses the Flash byte-by-byte, also the LSB (bit Z0) of the Z-pointer is used.

**Figure 30-1. Addressing the Flash During SPM**




**Note:** The different variables used in this figure are listed in *Page Size* section in Memory Programming chapter.

**Related Links**

Page Size on page 368

### 30.2.1. EEPROM Write Prevents Writing to SPMCSR

Note that an EEPROM write operation will block all software programming to Flash. Reading the Fuses and Lock bits from software will also be prevented during the EEPROM write operation. It is recommended that the user checks the status of the EEPE bit in the EEPROM Control Register (EECR.EEPE) and verifies that the bit is cleared before writing to the SPMCSR Register.

### 30.2.2. Reading the Fuse and Lock Bits from Software

It is possible to read both the Fuse and Lock bits (LB) from software. To read the Lock bits, load the Z-pointer with 0x0001 and set the BLBSET and SPMEN bits in SPMCSR (SPMCSR.BLBSET and SPMCSR.SPMEN). When an LPM instruction is executed within three CPU cycles after the BLBSET and SPMEN bits are set in SPMCSR (SPMCSR.BLBSET and SPMCSR.SPMEN), the value of the Lock bits will be loaded in the destination register. The SPMCSR.BLBSET and SPMCSR.SPMEN will auto-clear upon completion of reading the Lock bits or if no LPM instruction is executed within three CPU cycles or no SPM instruction is executed within four CPU cycles. When SPMCSR.BLBSET and SPMCSR.SPMEN are cleared, LPM will work as described in the Instruction set Manual.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Rd | – | – | – | – | – | – | LB2 | LB1 |

The algorithm for reading the Fuse Low byte (FLB) is similar to the one described above for reading the Lock bits. To read the Fuse Low byte, load the Z-pointer with 0x0000 and set the BLBSET and SPMEN

bits in SPMCSR (SPMCSR.BLBSET and SPMCSR.SPMEN). When an LPM instruction is executed within three cycles after the SPMCSR.BLBSET and SPMCSR.SPMEN are set, the value of the Fuse Low byte (FLB) will be loaded in the destination register as shown below. Refer to Table. Fuse Low Byte in *Fuse Bits* of Memory Programming chapter for a detailed description and mapping of the Fuse Low byte.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Rd | FLB7 | FLB6 | FLB5 | FLB4 | FLB3 | FLB2 | FLB1 | FLB0 |

Similarly, when reading the Fuse High byte (FHB), load 0x0003 in the Z-pointer. When an LPM instruction is executed within three cycles after the SPMCSR.BLBSET and SPMCSR.SPMEN are set, the value of the Fuse High byte (FHB) will be loaded in the destination register as shown below. Refer to Table. Fuse High Byte for this device in *Fuse Bits* of Memory Programming chapter for detailed description and mapping of the Fuse High byte.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Rd | FHB7 | FHB6 | FHB5 | FHB4 | FHB3 | FHB2 | FHB1 | FHB0 |

When reading the Extended Fuse byte (EFB), load 0x0002 in the Z-pointer. When an LPM instruction is executed within three cycles after the SPMCSR.BLBSET and SPMCSR.SPMEN are set, the value of the Extended Fuse byte (EFB) will be loaded in the destination register as shown below. Refer to Table. Extended Fuse Byte for ATmega88A/ATmega168A in *Fuse Bits* of Memory Programming chapter for detailed description and mapping of the Extended Fuse byte.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Rd | EFB7 | EFB6 | EFB5 | EFB4 | EFB3 | EFB2 | EFB1 | EFB0 |

Fuse and Lock bits that are programmed, will be read as zero. Fuse and Lock bits that are unprogrammed, will be read as one.

**Related Links**

Fuse Bits on page 365
Latching of Fuses on page 367

### 30.2.3. 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.

### 30.2.4. Programming Time for Flash when Using SPM

The calibrated RC Oscillator is used to time Flash accesses. The following table shows the typical programming time for Flash accesses from the CPU.

**Table 30-1. SPM Programming Time**

| Symbol | Min. Programming Time | Max Programming Time |
|---|---|---|
| Flash write (Page Erase, Page Write, and write Lock bits by SPM) | 3.2ms | 3.4ms |

**Note:** Minimum and maximum programming time is per individual operation.

### 30.2.5. Simple Assembly Code Example for a Boot Loader

Note that the Store Program Memory Control and Status Register (RWWSB) bit in SPMCSR will always be read as zero in Atmel ATmega48A. Nevertheless, it is recommended to check this bit as shown in the code example, to ensure compatibility with devices supporting Read-While-Write.

```
          ;-the routine writes one page of data from RAM to Flash
          ; the first data location in RAM is pointed to by the Y pointer
          ; the first data location in Flash is pointed to by the Z-pointer
          ;-error handling is not included
          ;-the routine must be placed inside the Boot space
          ; (at least the Do_spm sub routine). Only code inside NRWW section can
          ; be read during Self-Programming (Page Erase and Page Write).
          ;-registers used: r0, r1, temp1 (r16), temp2 (r17), looplo (r24),
          ; loophi (r25), spmcrval (r20)
          ; storing and restoring of registers is not included in the routine
          ; register usage can be optimized at the expense of code size
          ;-It is assumed that either the interrupt table is moved to the Boot
          ; loader section or that the interrupts are disabled.
.equ PAGESIZEB = PAGESIZE*2 ;PAGESIZEB is page size in BYTES, not words
.org SMALLBOOTSTART
Write_page:
          ; Page Erase
          ldi spmcrval, (1<<PGERS) | (1<<SPMEN)
          rcall Do_spm

          ; re-enable the RWW section
          ldi spmcrval, (1<<RWWSRE) | (1<<SPMEN)
          rcall Do_spm

          ; transfer data from RAM to Flash page buffer
          ldi looplo, low(PAGESIZEB)  ;init loop variable
          ldi loophi, high(PAGESIZEB) ;not required for PAGESIZEB<=256

Wrloop:
          ld r0, Y+
```

```
          ld r1, Y+
          ldi spmcrval, (1<<SPMEN)
          rcall Do_spm
          adiw ZH:ZL, 2
          sbiw loophi:looplo, 2 ;use subi for PAGESIZEB<=256
          brne Wrloop

         ; execute Page Write
         subi ZL, low(PAGESIZEB)  ;restore pointer
         sbci ZH, high(PAGESIZEB) ;not required for PAGESIZEB<=256
         ldi spmcrval, (1<<PGWRT) | (1<<SPMEN)
 rcall Do_spm

         ; re-enable the RWW section
 ldi spmcrval, (1<<RWWSRE) | (1<<SPMEN)
         rcall Do_spm

         ; read back and check, optional
         ldi looplo, low(PAGESIZEB)  ;init loop variable
         ldi loophi, high(PAGESIZEB) ;not required for PAGESIZEB<=256
         subi YL, low(PAGESIZEB)     ;restore pointer
         sbci YH, high(PAGESIZEB)
Rdloop:
         lpm r0, Z+
         ld r1, Y+
         cpse r0, r1
         rjmp Error
         sbiw loophi:looplo, 1 ;use subi for PAGESIZEB<=256
         brne Rdloop

        ; return to RWW section
        ; verify that RWW section is safe to read
Return:
        in temp1, SPMCSR
        sbrs temp1, RWWSB ; If RWWSB is set, the RWW section is not ready yet
        ret
        ; re-enable the RWW section
        ldi spmcrval, (1<<RWWSRE) | (1<<SPMEN)
        rcall Do_spm
        rjmp Return
```

```
Do_spm:
        ; check for previous SPM complete
Wait_spm:
        in temp1, SPMCSR
        sbrc temp1, SPMEN
        rjmp Wait_spm
        ; input: spmcrval determines SPM action
        ; disable interrupts if enabled, store status
        in temp2, SREG
        cli
        ; check that no EEPROM write access is present
Wait_ee:
        sbic EECR, EEPE
        rjmp Wait_ee
        ; SPM timed sequence
        out SPMCSR, spmcrval
        spm
        ; restore SREG (to enable interrupts if originally enabled)
        out SREG, temp2
ret
```

## 30.3. Register Description

### 30.3.1. 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.

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** SPMCSR
**Offset:** 0x57
**Reset:** 0x00
**Property:** When addressing I/O Registers as data space the offset address is 0x37

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | SPMIE | RWWSB | SIGRD | RWWSRE | BLBSET | PGWRT | PGERS | SPMEN |
| Access | R/W | R | R | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bit 7 – SPMIE: SPM Interrupt Enable**
When the SPMIE bit is written to '1', and the I-bit in the Status Register is set ('1'), the SPM ready interrupt will be enabled. The SPM ready Interrupt will be executed as long as the SPMEN bit in the SPMCSR Register is cleared (SPMCSR.SPMEN). The interrupt will not be generated during EEPROM write or SPM.

**Bit 6 – RWWSB: Read-While-Write Section Busy**
This bit is for compatibility with devices supporting Read-While-Write. It will always read as zero.

**Bit 5 – SIGRD: Signature Row Read**
If this bit is written to one at the same time as SPMEN, the next LPM instruction within three clock cycles will read a byte from the signature row into the destination register. Please refer to *Reading the Signature Row from Software*. An SPM instruction within four cycles after SIGRD and SPMEN are set will have no effect. This operation is reserved for future use and should not be used.

**Bit 4 – RWWSRE: Read-While-Write Section Read Enable**
The functionality of this bit in ATmega48A is a subset of the functionality in ATmega88A and ATmega168A. If the RWWSRE bit is written while filling the temporary page buffer, the temporary page buffer will be cleared and the data will be lost.

**Bit 3 – BLBSET: Boot Lock Bit Set**
The functionality of this bit in ATmega48A is a subset of the functionality in ATmega88A and ATmega168A. An LPM instruction within three cycles after BLBSET and SPMEN are set in the SPMCSR Register, will read either the Lock bits or the Fuse bits (depending on Z0 in the Z-pointer) into the destination register. Please refer to Reading the Fuse and Lock Bits from Software

**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 Zpointer. 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**

This bit enables the SPM instruction for the next four clock cycles. If written to one together with either RWWSRE, BLBSET, PGWRT, or PGERS, the following SPM instruction will have a special meaning, see description above. 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.

Writing any other combination than “0x10001”, “0x01001”, “0x00101”, “0x00011” or “0x00001” in the lower five bits will have no effect.

# 31. BTLDR - Boot Loader Support – Read-While-Write Self-Programming

## 31.1. Features

- Read-While-Write Self-Programming
- Flexible Boot Memory Size
- High Security (Separate Boot Lock Bits for a Flexible Protection)
- Separate Fuse to Select Reset Vector
- Optimized Page(1) Size
- Code Efficient Algorithm
- Efficient Read-Modify-Write Support

**Note:** 1. A page is a section in the Flash consisting of several bytes (see Table. No. of Words in a Page and No. of Pages in the Flash in *Page Size*) used during programming. The page organization does not affect normal operation.

## 31.2. Overview

In this device, the Boot Loader Support provides a real Read-While-Write Self-Programming mechanism for downloading and uploading program code by the MCU itself. This feature allows flexible application software updates controlled by the MCU using a Flash-resident Boot Loader program. The Boot Loader program can use any available data interface and associated protocol to read code and write (program) that code into the Flash memory, or read the code from the program memory. The program code within the Boot Loader section has the capability to write into the entire Flash, including the Boot Loader memory. The Boot Loader can thus even modify itself, and it can also erase itself from the code if the feature is not needed anymore. The size of the Boot Loader memory is configurable with fuses and the Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

## 31.3. Application and Boot Loader Flash Sections

The Flash memory is organized in two main sections, the Application section and the Boot Loader section. The size of the different sections is configured by the BOOTSZ Fuses. These two sections can have different level of protection since they have different sets of Lock bits.

### 31.3.1. Application Section

The Application section is the section of the Flash that is used for storing the application code. The protection level for the Application section can be selected by the application Boot Lock bits (Boot Lock bits 0). The Application section can never store any Boot Loader code since the SPM instruction is disabled when executed from the Application section.

### 31.3.2. BLS – Boot Loader Section

While the Application section is used for storing the application code, the Boot Loader software must be located in the BLS since the SPM instruction can initiate a programming when executing from the BLS only. The SPM instruction can access the entire Flash, including the BLS itself. The protection level for the Boot Loader section can be selected by the Boot Loader Lock bits (Boot Lock bits 1).

## 31.4. Read-While-Write and No Read-While-Write Flash Sections

Whether the CPU supports Read-While-Write or if the CPU is halted during a Boot Loader software update is dependent on which address that is being programmed. In addition to the two sections that are configurable by the BOOTSZ Fuses as described above, the Flash is also divided into two fixed sections, the Read-While-Write (RWW) section and the No Read-While-Write (NRWW) section. The limit between the RWW- and NRWW sections is given in the *Boot Loader Parameters* section and Figure 31-2. The main difference between the two sections is:

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

The user software can never read any code that is located inside the RWW section during a Boot Loader software operation. The syntax “Read-While-Write section” refers to which section that is being programmed (erased or written), not which section that actually is being read during a Boot Loader software update.

**Related Links**

ATmega168A Boot Loader Parameters on page 360

ATmega88A Boot Loader Parameters on page 358

### 31.4.1. RWW – Read-While-Write Section

If a Boot Loader software update 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 an on-going programming, the software must ensure that the RWW section never is being read. If the user software is trying to read code that is located inside the RWW section (i.e., by a call/jmp/lpm or an interrupt) during programming, the software might end up in an unknown state. To avoid this, the interrupts should either be disabled or moved to the Boot Loader section. The Boot Loader section is always located in the NRWW section. The RWW Section Busy bit (RWWSB) in the Store Program Memory Control and Status Register (SPMCSR) will be read as logical one 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. Please refer to SPMCSR – Store Program Memory Control and Status Register in this chapter for details on how to clear RWWSB.

### 31.4.2. NRWW – No Read-While-Write Section

The code located in the NRWW section can be read when the Boot Loader software is updating a page in the RWW section. When the Boot Loader code updates the NRWW section, the CPU is halted during the entire Page Erase or Page Write operation.

**Table 31-1. Read-While-Write Features**

| Which Section does the Z-pointer Address during the Programming? | Which Section can be read during Programming? | CPU Halted? | Read-While-Write Supported? |
|---|---|---|---|
| RWW Section | NRWW Section | No | Yes |
| NRWW Section | None | Yes | No |

**Figure 31-1. Read-While-Write vs. No Read-While-Write**

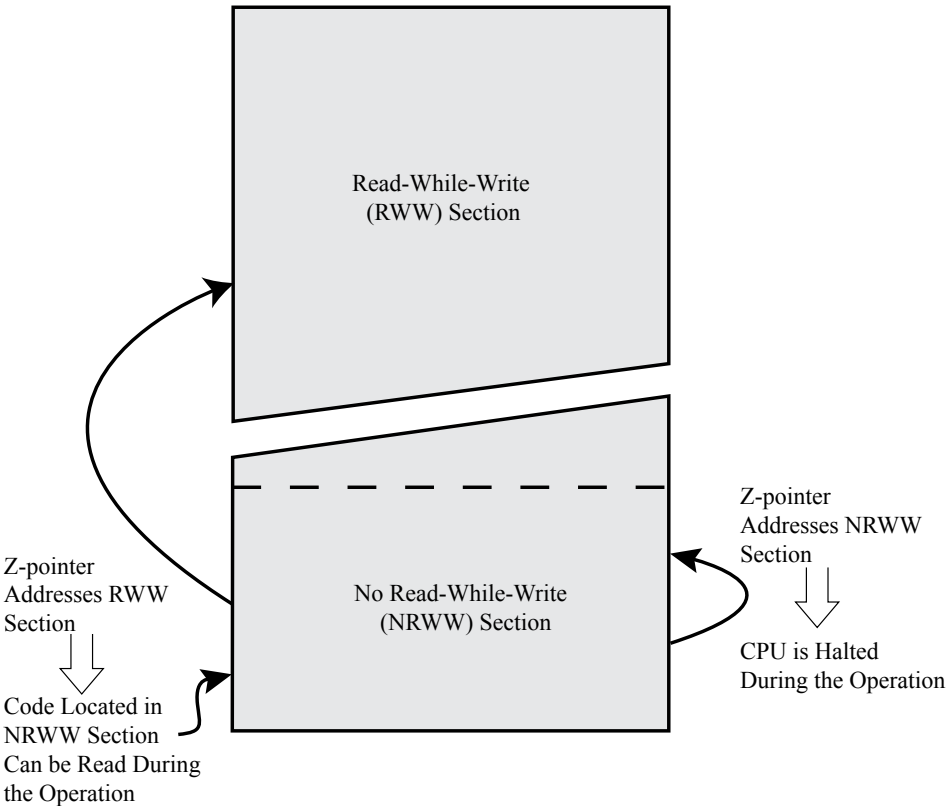
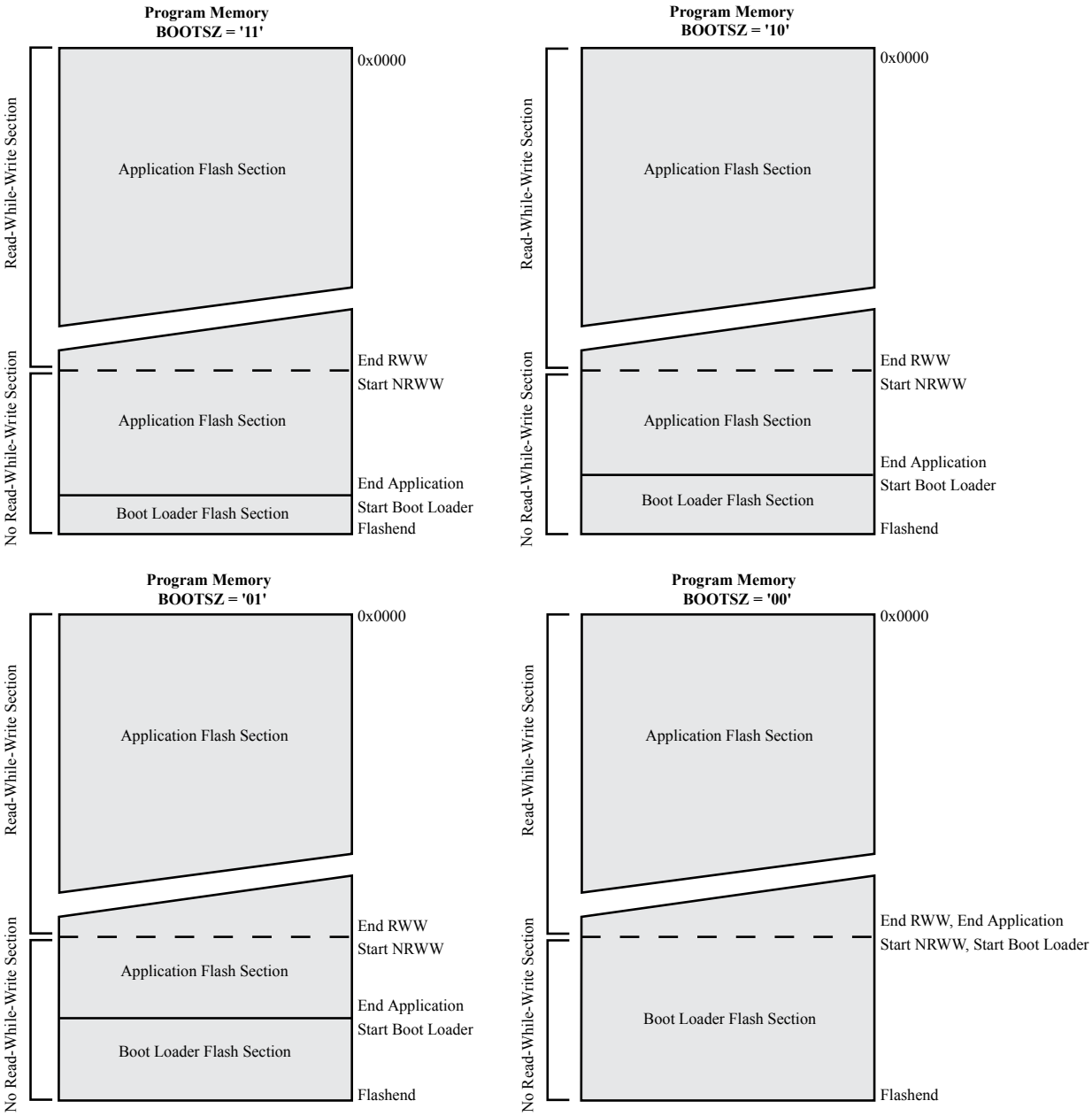


**Figure 31-2. Memory Sections**




**Related Links**

ATmega168A Boot Loader Parameters on page 360

ATmega88A Boot Loader Parameters on page 358

## 31.5. Boot Loader Lock Bits

If no Boot Loader capability is needed, the entire Flash is available for application code. 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.

The user can select:

- To protect the entire Flash from a software update by the MCU

- To protect only the Boot Loader Flash section from a software update by the MCU
- To protect only the Application Flash section from a software update by the MCU
- Allow software update in the entire Flash

The Boot Lock bits can be set in software and in Serial or Parallel Programming mode, but they can be cleared by a Chip Erase command only. The general Write Lock (Lock Bit mode 2) does not control the programming of the Flash memory by SPM instruction. Similarly, the general Read/Write Lock (Lock Bit mode 1) does not control reading nor writing by LPM/SPM, if it is attempted.

**Table 31-2. Boot Lock Bit0 Protection Modes (Application Section)**

| BLB0 Mode | BLB02 | BLB01 | Protection |
|---|---|---|---|
| 1 | 1 | 1 | No restrictions for SPM or LPM accessing the Application section. |
| 2 | 1 | 0 | SPM is not allowed to write to the Application section. |
| 3 | 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. |
| 4 | 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. |

**Note:** “1” means unprogrammed, “0” means programmed.

**Table 31-3. Boot Lock Bit1 Protection Modes (Boot Loader Section)**

| BLB1 Mode | BLB12 | BLB11 | Protection |
|---|---|---|---|
| 1 | 1 | 1 | No restrictions for SPM or LPM accessing the Boot Loader section. |
| 2 | 1 | 0 | SPM is not allowed to write to the Boot Loader section. |
| 3 | 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. |
| 4 | 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. |

**Note:** “1” means unprogrammed, “0” means programmed.

## 31.6. 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 interface. Alternatively, the Boot Reset Fuse can be programmed so that the Reset Vector is pointing to the Boot Flash start address after a reset. In this case, the Boot Loader is started after a reset. After the application code is loaded, the program can

start executing the application code. The fuses cannot be changed by the MCU itself. This means that once the Boot Reset Fuse is programmed, the Reset Vector will always point to the Boot Loader Reset and the fuse can only be changed through the serial or parallel programming interface.

**Table 31-4. Boot Reset Fuse**

| BOOTRST | Reset Address |
|---|---|
| 1 | Reset Vector = Application Reset (address 0x0000) |
| 0 | Reset Vector = Boot Loader Reset, as described by the Boot Loader Parameters |

**Note:** '1' means unprogrammed, '0' means programmed.

## 31.7. Addressing the Flash During Self-Programming

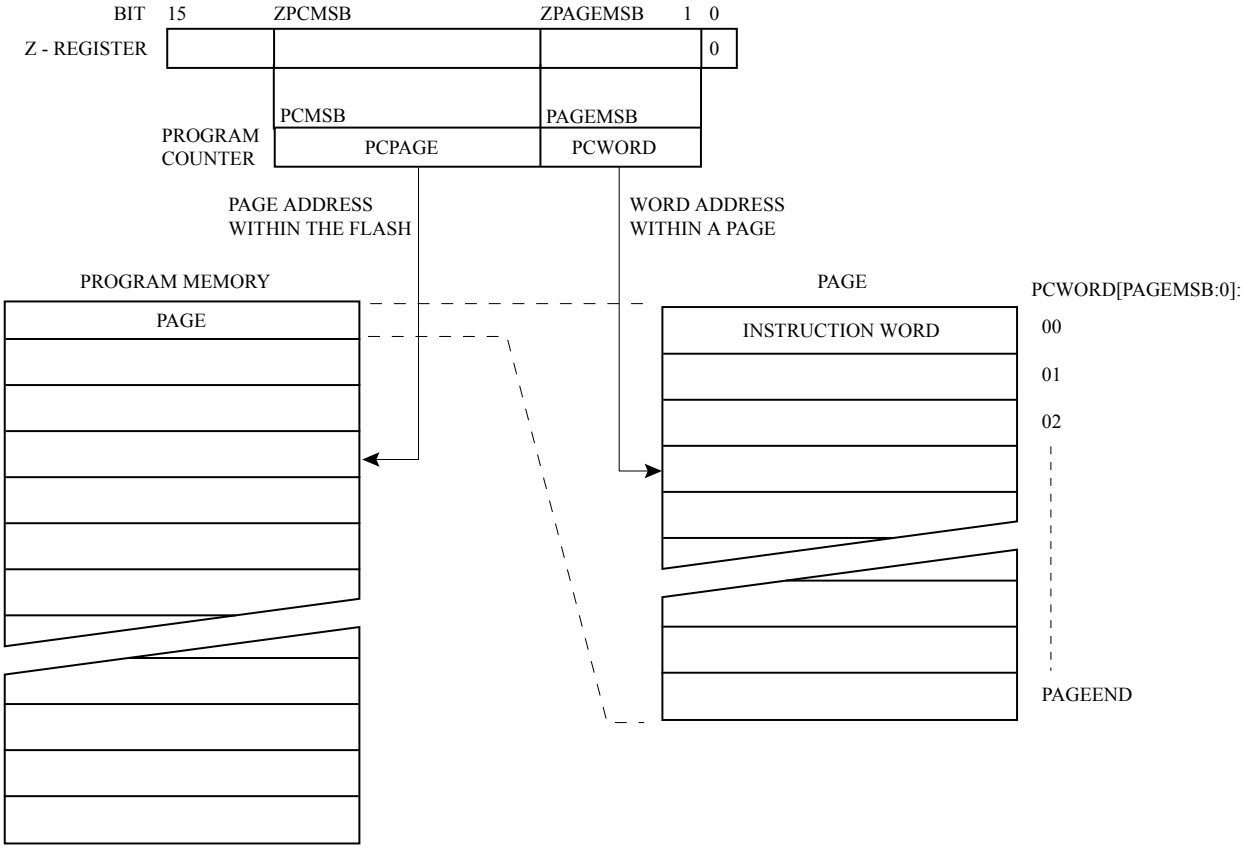
The Z-pointer is used to address the SPM commands.

| 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, 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 the following figure. The Page Erase and Page Write operations are addressed independently. Therefore it is of major importance that the Boot Loader software addresses the same page in both the Page Erase and Page Write operation. Once a programming operation is initiated, the address is latched and the Z-pointer can be used for other operations.

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 the operation. The LPM instruction does also use the Z-pointer to store the address. Since this instruction addresses the Flash byte-by-byte, also the LSB (bit Z0) of the Z-pointer is used.

**Figure 31-3. Addressing the Flash During SPM**




**Note:** The different variables used in this figure are listed in the Related Links.

**Related Links**

ATmega88PB Boot Loader Parameters on page 358

## 31.8. Self-Programming the Flash

The 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:

**Alternative 1, fill the buffer before a Page Erase**

- Fill temporary page buffer
- Perform a Page Erase
- Perform a Page Write

**Alternative 2, fill the buffer after Page Erase**

- Perform a Page Erase
- Fill temporary page buffer
- Perform a Page Write

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 rewritten. 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. 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. Please refer to Simple Assembly Code Example for a Boot Loader.

### 31.8.1. Performing Page Erase by SPM

To execute Page Erase, set up the address in the Z-pointer, write “0x0000011” to Store Program Memory Control and Status Register (SPMCSR) and execute SPM 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 will be ignored during this operation.

- Page Erase to the RWW section: The NRWW section can be read during the Page Erase.
- Page Erase to the NRWW section: The CPU is halted during the operation.

### 31.8.2. Filling the Temporary Buffer (Page Loading)

To write an instruction word, set up the address in the Z-pointer and data in [R1:R0], write “0x00000001” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The content of PCWORD ([Z5:Z1]) 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 (SPMCSR.RWWSRE). It is also erased after a system reset. 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.

### 31.8.3. Performing a Page Write

To execute Page Write, set up the address in the Z-pointer, write “0x0000101” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The data in R1 and R0 is ignored. The page address must be written to PCPAGE ([Z5:Z1]). Other bits in the Z-pointer must be written to zero during this operation.

- Page Write to the RWW section: The NRWW section can be read during the Page Write
- Page Write to the NRWW section: The CPU is halted during the operation

### 31.8.4. Using the SPM Interrupt

If the SPM interrupt is enabled, the SPM interrupt will generate a constant interrupt when the SPMEN bit in SPMCSR is cleared (SPMCSR.SPMEN). This means that the interrupt can be used instead of polling the SPMCSR Register in software. When using the SPM interrupt, the Interrupt Vectors should be moved to the Boot Loader Section (BLS) section to avoid that an interrupt is accessing the RWW section when it is blocked for reading. How to move the interrupts is described in *Interrupts* chapter.

**Related Links**

Interrupts on page 84

### 31.8.5. Consideration While Updating Boot Loader Section (BLS)

Special care must be taken if the user allows the Boot Loader Section (BLS) to be updated by leaving Boot Lock bit11 unprogrammed. An accidental write to the Boot Loader itself can corrupt the entire Boot Loader, and further software updates might be impossible. If it is not necessary to change the Boot Loader software itself, it is recommended to program the Boot Lock bit11 to protect the Boot Loader software from any internal software changes.

### 31.8.6. Prevent Reading the RWW Section During Self-Programming

During Self-Programming (either Page Erase or Page Write), the RWW section is always blocked for reading. The user software itself must prevent that this section is addressed during the self programming operation. The RWWSB in the SPMCSR (SPMCSR.RWWSB) will be set as long as the RWW section is busy. During Self-Programming the Interrupt Vector table should be moved to the BLS as described in *Watchdog Timer* chapter, or the interrupts must be disabled. Before addressing the RWW section after the programming is completed, the user software must clear the SPMCSR.RWWSB by writing the SPMCSR.RWWSRE. Please refer to Simple Assembly Code Example for a Boot Loader for an example.

**Related Links**

Watchdog System Reset on page 77

### 31.8.7. Setting the Boot Loader Lock Bits by SPM

To set the Boot Loader Lock bits and general Lock Bits, write the desired data to R0, write “0x0001001” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R0 | 1 | 1 | BLB12 | BLB11 | BLB02 | BLB01 | LB2 | LB1 |

The tables in Boot Loader Lock Bits show how the different settings of the Boot Loader bits affect the Flash access.

If bits 5...0 in R0 are cleared (zero), the corresponding Lock bit will be programmed if an SPM instruction is executed within four cycles after BLBSET and SPMEN are set in SPMCSR (SPMCSR.BLBSET and SPMCSR.SPMEN). The Z-pointer don’t care during this operation, but for future compatibility it is recommended to load the Z-pointer with 0x0001 (same as used for reading the $IO_{ck}$ 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.

### 31.8.8. EEPROM Write Prevents Writing to SPMCSR

An EEPROM write operation will block all software programming to Flash. Reading the 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 the EECR Register (EECR.EEPE) and verifies that the bit is cleared before writing to the SPMCSR Register.

### 31.8.9. Reading the Fuse and Lock Bits from Software

It is possible to read both the Fuse and Lock bits (LB) from software. To read the Lock bits, load the Z-pointer with 0x0001 and set the BLBSET and SPMEN bits in SPMCSR (SPMCSR.BLBSET and SPMCSR.SPMEN). When an LPM instruction is executed within three CPU cycles after the BLBSET and SPMEN bits are set in SPMCSR (SPMCSR.BLBSET and SPMCSR.SPMEN), the value of the Lock bits will be loaded in the destination register. The SPMCSR.BLBSET and SPMCSR.SPMEN will auto-clear upon completion of reading the Lock bits or if no LPM instruction is executed within three CPU cycles or no SPM instruction is executed within four CPU cycles. When SPMCSR.BLBSET and SPMCSR.SPMEN are cleared, LPM will work as described in the Instruction set Manual.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Rd | - | - | BLB12 | BLB11 | BLB02 | BLB01 | LB2 | LB1 |

The algorithm for reading the Fuse Low byte (FLB) is similar to the one described above for reading the Lock bits. To read the Fuse Low byte, load the Z-pointer with 0x0000 and set the BLBSET and SPMEN bits in SPMCSR (SPMCSR.BLBSET and SPMCSR.SPMEN). When an LPM instruction is executed within three cycles after the SPMCSR.BLBSET and SPMCSR.SPMEN are set, the value of the Fuse Low byte (FLB) will be loaded in the destination register as shown below.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Rd | FLB7 | FLB6 | FLB5 | FLB4 | FLB3 | FLB2 | FLB1 | FLB0 |

Similarly, when reading the Fuse High byte (FHB), load 0x0003 in the Z-pointer. When an LPM instruction is executed within three cycles after the SPMCSR.BLBSET and SPMCSR.SPMEN are set, the value of the Fuse High byte (FHB) will be loaded in the destination register as shown below.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Rd | FHB7 | FHB6 | FHB5 | FHB4 | FHB3 | FHB2 | FHB1 | FHB0 |

When reading the Extended Fuse byte (EFB), load 0x0002 in the Z-pointer. When an LPM instruction is executed within three cycles after the SPMCSR.BLBSET and SPMCSR.SPMEN are set, the value of the Extended Fuse byte (EFB) will be loaded in the destination register as shown below.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Rd | - | - | - | - | EFB3 | EFB2 | EFB1 | EFB0 |

Fuse and Lock bits that are programmed read as '0'. Fuse and Lock bits that are unprogrammed, will read as '1'.

### 31.8.10. Reading the Signature Row from Software

To read the Signature Row from software, load the Z-pointer with the signature byte address given in the following table and set the SIGRD and SPMEN bits in SPMCSR (SPMCSR.SIGRD and SPMCSR.SPMEN). When an LPM instruction is executed within three CPU cycles after the SPMCSR.SIGRD and SPMCSR.SPMEN are set, the signature byte value will be loaded in the destination register. The SPMCSR.SIGRD and SPMCSR.SPMEN will auto-clear upon completion of reading the Signature Row Lock bits or if no LPM instruction is executed within three CPU cycles. When SPMCSR.SIGRD and SPMCSR.SPMEN are cleared, LPM will work as described in the Instruction set Manual.

**Table 31-5. Signature Row Addressing**

| Signature Byte | Z-pointer Address |
|---|---|
| Device Signature Byte 1 | 0x0000 |
| Device Signature Byte 2 | 0x0002 |
| Device Signature Byte 3 | 0x0004 |
| RC Oscillator Calibration Byte | 0x0001 |

**Note:** All other addresses are reserved for future use.

### 31.8.11. 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. If there is no need for a Boot Loader update in the system, program the Boot Loader Lock bits to prevent any Boot Loader software updates.
2. 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.
3. 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.

### 31.8.12. Programming Time for Flash when Using SPM

The calibrated RC Oscillator is used to time Flash accesses. The following table shows the typical programming time for Flash accesses from the CPU.

**Table 31-6. SPM Programming Time**

| Symbol | Min. Programming Time | Max. Programming Time |
|---|---|---|
| Flash write (Page Erase, Page Write, and write Lock bits by SPM) | 3.2ms | 3.4ms |

**Note:** Minimum and maximum programming time is per individual operation.

### 31.8.13. Simple Assembly Code Example for a Boot Loader

```
;-the routine writes one page of data from RAM to Flash
; the first data location in RAM is pointed to by the Y pointer
; the first data location in Flash is pointed to by the Z-pointer
;-error handling is not included
 ;-the routine must be placed inside the Boot space
; (at least the Do_spm sub routine). Only code inside NRWW section can
; be read during Self-Programming (Page Erase and Page Write).
;-registers used: r0, r1, temp1 (r16), temp2 (r17), looplo (r24),
; loophi (r25), spmcrval (r20)
; storing and restoring of registers is not included in the routine
; register usage can be optimized at the expense of code size
;-It is assumed that either the interrupt table is moved to the Boot
; loader section or that the interrupts are disabled.
.equ PAGESIZEB = PAGESIZE*2 ;PAGESIZEB is page size in BYTES, not words
.org SMALLBOOTSTART
Write_page:
      ; Page Erase
      ldi spmcrval, (1<<PGERS) | (1<<SPMEN)
      call Do_spm
```

```
        ; re-enable the RWW section
        ldi spmcrval, (1<<RWWSRE) | (1<<SPMEN)
        call Do_spm

        ; transfer data from RAM to Flash page buffer
        ldi looplo, low(PAGESIZEB)  ;init loop variable
        ldi loophi, high(PAGESIZEB) ;not required for PAGESIZEB<=256
Wrloop:
        ld r0, Y+
        ld r1, Y+
        ldi spmcrval, (1<<SPMEN)
        call Do_spm
        adiw ZH:ZL, 2
        sbiw loophi:looplo, 2 ;use subi for PAGESIZEB<=256
        brne Wrloop

        ; execute Page Write
        subi ZL, low(PAGESIZEB) ;restore pointer
        sbci ZH, high(PAGESIZEB) ;not required for PAGESIZEB<=256
        ldi spmcrval, (1<<PGWRT) | (1<<SPMEN)
        call Do_spm

      ; re-enable the RWW section
      ldi spmcrval, (1<<RWWSRE) | (1<<SPMEN)
      call Do_spm

      ; read back and check, optional
      ldi looplo, low(PAGESIZEB) ;init loop variable
      ldi loophi, high(PAGESIZEB) ;not required for PAGESIZEB<=256
      subi YL, low(PAGESIZEB) ;restore pointer
      sbci YH, high(PAGESIZEB)
Rdloop:
      lpm r0, Z+
      ld r1, Y+
      cpse r0, r1
      jmp Error
      sbiw loophi:looplo, 1 ;use subi for PAGESIZEB<=256
      brne Rdloop
```

```
      ; return to RWW section
      ; verify that RWW section is safe to read
Return:
      in temp1, SPMCSR
      sbrs temp1, RWWSB ; If RWWSB is set, the RWW section is not ready yet
      ret
      ; re-enable the RWW section
      ldi spmcrval, (1<<RWWSRE) | (1<<SPMEN)
      call Do_spm
      rjmp Return

Do_spm:
      ; check for previous SPM complete
Wait_spm:
      in temp1, SPMCSR
      sbrc temp1, SPMEN
      rjmp Wait_spm

      ; input: spmcrval determines SPM action
      ; disable interrupts if enabled, store status
      in temp2, SREG
      cli
      ; check that no EEPROM write access is present
Wait_ee:
      sbic EECR, EEPE
      rjmp Wait_ee
      ; SPM timed sequence
      out SPMCSR, spmcrval
      spm
      ; restore SREG (to enable interrupts if originally enabled)
      out SREG, temp2
      ret
```

### 31.8.14. ATmega88A Boot Loader Parameters

The following tables are the parameters used in the description of the self programming are given.

**Table 31-7. Boot Size Configuration, ATmega88A**

| BOOTSZ1 | BOOTSZ0 | Boot Size | Pages | Application Flash Section | Boot Loader Flash Section | End Application Section | Boot Reset Address (Start Boot Loader Section) |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 128 words | 4 | 0x000 - 0xF7F | 0xF80 - 0xFFF | 0xF7F | 0xF80 |
| 1 | 0 | 256 words | 8 | 0x000 - 0xEFF | 0xF00 - 0xFFF | 0xEFF | 0xF00 |
| 0 | 1 | 512 words | 16 | 0x000 - 0xDFF | 0xE00 - 0xFFF | 0xDFF | 0xE00 |
| 0 | 0 | 1024 words | 32 | 0x000 - 0xBFF | 0xC00 - 0xFFF | 0xBFF | 0xC00 |

**Note:** The different BOOTSZ Fuse configurations are shown in Figure 31-2.

**Table 31-8. Read-While-Write Limit, ATmega88A**

| Section | Pages | Address |
|---|---|---|
| Read-While-Write section (RWW) | 96 | 0x000 - 0xBFF |
| No Read-While-Write section (NRWW) | 32 | 0xC00 - 0xFFF |

For details about these two section, please refer to NRWW – No Read-While-Write Section and RWW – Read-While-Write Section.

**Table 31-9. Explanation of Different Variables used in Figure 31-3**

| Variable | | Corresponding Z-value(1) | Description |
|---|---|---|---|
| PCMSB | 11 | | Most significant bit in the Program Counter. (The Program Counter is 12 bits PC[11:0]) |
| PAGEMSB | 4 | | Most significant bit which is used to address the words within one page (32 words in a page requires 5 bits PC [4:0]). |
| ZPCMSB | | Z12 | Bit in Z-register that is mapped to PCMSB. Because Z0 is not used, the ZPCMSB equals PCMSB + 1. |
| ZPAGEMSB | | Z5 | Bit in Z-register that is mapped to PAGEMSB. Because Z0 is not used, the ZPAGEMSB equals PAGEMSB + 1. |
| PCPAGE | PC[11:5] | Z12:Z6 | Program counter page address: Page select, for page erase and page write |
| PCWORD | PC[4:0] | Z5:Z1 | Program counter word address: Word select, for filling temporary buffer (must be zero during page write operation) |

**Note:** 1. Z15:Z13: always ignored

Z0: should be zero for all SPM commands, byte select for the LPM instruction.

Please refer to Addressing the Flash During Self-Programmingor details about the use of Z-pointer during Self- Programming.

### 31.8.15. ATmega168A Boot Loader Parameters

The following tables are the parameters used in the description of the self programming are given.

**Table 31-10. Boot Size Configuration, ATmega168A**

| BOOTSZ1 | BOOTSZ0 | Boot Size | Pages | Application Flash Section | Boot Loader Flash Section | End Application Section | Boot Reset Address (Start Boot Loader Section) |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 128 words | 2 | 0x0000 - 0x1F7F | 0x1F80 - 0x1FFF | 0x1F7F | 0x1F80 |
| 1 | 0 | 256 words | 4 | 0x0000 - 0x1EFF | 0x1F00 - 0x1FFF | 0x1EFF | 0x1F00 |
| 0 | 1 | 512 words | 8 | 0x0000 - 0x1DFF | 0x1E00 - 0x1FFF | 0x1DFF | 0x1E00 |
| 0 | 0 | 1024 words | 16 | 0x0000 - 0x1BFF | 0x1C00 - 0x1FFF | 0x1BFF | 0x1C00 |

**Note:** The different BOOTSZ Fuse configurations are shown in Figure 31-2.

**Table 31-11. Read-While-Write Limit, ATmega168A**

| Section | Pages | Address |
|---|---|---|
| Read-While-Write section (RWW) | 112 | 0x0000 - 0x1BFF |
| No Read-While-Write section (NRWW) | 16 | 0x1C00 - 0x1FFF |

For details about these two section, please refer to NRWW – No Read-While-Write Section and RWW – Read-While-Write Section.

**Table 31-12. Explanation of Different Variables used in Figure 31-3, ATmega168A**

| Variable | | Corresponding Z-value(1) | Description |
|---|---|---|---|
| PCMSB | 12 | | Most significant bit in the Program Counter. (The Program Counter is 13 bits PC[12:0]) |
| PAGEMSB | 5 | | Most significant bit which is used to address the words within one page (64 words in a page requires 6 bits PC [5:0]) |
| ZPCMSB | | Z13 | Bit in Z-register that is mapped to PCMSB. Because Z0 is not used, the ZPCMSB equals PCMSB + 1. |
| ZPAGEMSB | | Z6 | Bit in Z-register that is mapped to PAGEMSB. Because Z0 is not used, the ZPAGEMSB equals PAGEMSB + 1. |

| Variable | | Corresponding Z-value[1] | Description |
|---|---|---|---|
| PCPAGE | PC[12:6] | Z13:Z7 | Program counter page address: Page select, for page erase and page write |
| PCWORD | PC[5:0] | Z6:Z1 | Program counter word address: Word select, for filling temporary buffer (must be zero during page write operation) |

**Note:** 1. Z15:Z14: always ignored

Z0: should be zero for all SPM commands, byte select for the LPM instruction.

Please refer to Addressing the Flash During Self-Programming or details about the use of Z-pointer during Self- Programming.

## 31.9. Register Description

### 31.9.1. SPMCSR – Store Program Memory Control and Status Register

The Store Program Memory Control and Status Register contains the control bits needed to control the Boot Loader operations.

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

**Name:** SPMCSR
**Offset:** 0x57
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x37

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | SPMIE | RWWSB | SIGRD | RWWSRE | BLBSET | PGWRT | PGERS | SPMEN |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 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 in the SPMCSR Register is cleared.

**Bit 6 – RWWSB: Read-While-Write Section Busy**
When a Self-Programming (Page Erase or Page Write) operation to the RWW section is initiated, the RWWSB will be set (one) by hardware. When the RWWSB bit is set, the RWW section cannot be accessed. The RWWSB bit will be cleared if the RWWSRE bit is written to one after a Self-Programming operation is completed. Alternatively the RWWSB bit will automatically be cleared if a page load operation is initiated.

**Bit 5 – SIGRD: Signature Row Read**
If this bit is written to one at the same time as SPMEN, the next LPM instruction within three clock cycles will read a byte from the signature row into the destination register. Please refer to *Reading the Fuse and Lock Bits from Software* in this chapter. An SPM instruction within four cycles after SIGRD and SPMEN are set will have no effect. This operation is reserved for future use and should not be used.

**Bit 4 – RWWSRE: Read-While-Write Section Read Enable**
When programming (Page Erase or Page Write) to the RWW section, the RWW section is blocked for reading (the RWWSB will be set by hardware). To re-enable the RWW section, the user software must wait until the programming is completed (SPMEN will be cleared). Then, if the RWWSRE bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles re-enables the RWW section. The RWW section cannot be re-enabled while the Flash is busy with a Page Erase or a Page Write (SPMEN is set). If the RWWSRE bit is written while the Flash is being loaded, the Flash load operation will abort and the data loaded will be lost.

**Bit 3 – BLBSET: Boot Lock Bit Set**
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. The BLBSET bit will automatically be cleared upon completion of the Lock bit set, or if no SPM instruction is executed within four clock cycles.

An LPM instruction within three cycles after BLBSET and SPMEN are set in the SPMCSR Register (SPMCSR.BLBSET and SPMCSR.SPMEN), will read either the Lock bits or the Fuse bits (depending on Z0 in the Z-pointer) into the destination register. Please refer to *Reading the Fuse and Lock Bits from Software* in this chapter.

**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 Zpointer. 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 if the NRWW section is addressed.

**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 if the NRWW section is addressed.

**Bit 0 – SPMEN: Store Program Memory**
This bit enables the SPM instruction for the next four clock cycles. If written to one together with either RWWSRE, BLBSET, PGWRT or PGERS, the following SPM instruction will have a special meaning, see description above. 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.
Writing any other combination than “0x10001”, “0x01001”, “0x00101”, “0x00011” or “0x00001” in the lower five bits will have no effect.

# 32. MEMPROG- Memory Programming

## 32.1. Program And Data Memory Lock Bits

The ATmega48A provides two Lock bits and the other devices provides six Lock bits. These can be left unprogrammed ('1') or can be programmed ('0') to obtain the additional features listed in Table. Lock Bit Protection Modes in this section. The Lock bits can only be erased to “1” with the Chip Erase command.

The ATmega48A has no separate Boot Loader section, and the Store Program Memory (SPM) instruction is enabled for the whole Flash if the SELFPRGEN fuse is programmed (“0”). Otherwise the SPM instruction is disabled.

**Table 32-1. Lock Bit Byte(1)**

| Lock Bit Byte | Bit No. | Description | Default Value |
|---|---|---|---|
| | 7 | – | 1 (unprogrammed) |
| | 6 | – | 1 (unprogrammed) |
| BLB12(2) | 5 | Boot Lock bit | 1 (unprogrammed) |
| BLB11(2) | 4 | Boot Lock bit | 1 (unprogrammed) |
| BLB02(2) | 3 | Boot Lock bit | 1 (unprogrammed) |
| BLB01(2) | 2 | Boot Lock bit | 1 (unprogrammed) |
| LB2 | 1 | Lock bit | 1 (unprogrammed) |
| LB1 | 0 | Lock bit | 1 (unprogrammed) |

**Note:**

1. '1' means unprogrammed, '0' means programmed.
2. Only on ATmega88A and ATmega168A.

**Table 32-2. Lock Bit Protection Modes(1)(2)**

| Memory Lock Bits | | | Protection Type |
|---|---|---|---|
| LB Mode | LB2 | LB1 | |
| 1 | 1 | 1 | No memory lock features enabled. |
| 2 | 1 | 0 | Further programming of the Flash and EEPROM is disabled in Parallel and Serial Programming mode. The Fuse bits are locked in both Serial and Parallel Programming mode.(1) |
| 3 | 0 | 0 | Further programming and verification of the Flash and EEPROM is disabled in Parallel and Serial Programming mode. The Boot Lock bits and Fuse bits are locked in both Serial and Parallel Programming mode.(1) |

**Note:**

1. Program the Fuse bits and Boot Lock bits before programming the LB1 and LB2.
2. '1' means unprogrammed, '0' means programmed.

**Table 32-3. Lock Bit Protection - BLB0 Mode[1][2]. (Only ATmega88A and ATmega168A.)**

| BLB0 Mode | BLB02 | BLB01 | |
|---|---|---|---|
| 1 | 1 | 1 | No restrictions for SPM or Load Program Memory (LPM) instruction accessing the Application section. |
| 2 | 1 | 0 | SPM is not allowed to write to the Application section. |
| 3 | 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. |
| 4 | 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. |

**Table 32-4. Lock Bit Protection - BLB1 Mode[1][2] (Only ATmega88A and ATmega168A.)**

| BLB1 Mode | BLB12 | BLB11 | |
|---|---|---|---|
| 1 | 1 | 1 | No restrictions for SPM or LPM accessing the Boot Loader section. |
| 2 | 1 | 0 | SPM is not allowed to write to the Boot Loader section. |
| 3 | 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. |
| 4 | 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. |

**Note:**

1. Program the Fuse bits and Boot Lock bits before programming the LB1 and LB2.
2. '1' means unprogrammed; '0' means programmed.

## 32.2. Fuse Bits

The device has three Fuse bytes. The following tables describe briefly the functionality of all the fuses and how they are mapped into the Fuse bytes. Note that the fuses are read as logical zero, “0”, if they are programmed.

**Table 32-5. Extended Fuse Byte for ATmega48A**

| Extended Fuse Byte | Bit No. | Description | Default Value |
|---|---|---|---|
| – | 7 | – | 1 |
| – | 6 | – | 1 |

| Extended Fuse Byte | Bit No. | Description | Default Value |
|---|---|---|---|
| – | 5 | – | 1 |
| – | 4 | – | 1 |
| – | 3 | – | 1 |
| - | 2 | - | 1 (unprogrammed) |
| - | 1 | - | 1 (unprogrammed) |
| SELFPRGEN | 0 | Self Programming Enable | 1 (unprogrammed) |

**Table 32-6. Extended Fuse Byte for ATmega88A/ATmega168A**

| Extended Fuse Byte | Bit No. | Description | Default Value |
|---|---|---|---|
| – | 7 | – | 1 |
| – | 6 | – | 1 |
| – | 5 | – | 1 |
| – | 4 | – | 1 |
| – | 3 | – | 1 |
| BOOTSZ1 | 2 | Select Boot Size(1) | 0 (programmed)(2) |
| BOOTSZ0 | 1 | Select Boot Size(1) | 0 (programmed)(2) |
| BOOTRST | 0 | Select Reset Vector | 1 (unprogrammed) |

**Note:**

1. See Boot Size Configuration in the Boot Loader Parameters section.
2. The default value of BOOTSZ[1:0] results in maximum Boot Size. See Table 32-12.

**Table 32-7. Fuse High Byte**

| High Fuse Byte | Bit No. | Description | Default Value |
|---|---|---|---|
| RSTDISBL(1) | 7 | External Reset Disable | 1 (unprogrammed) |
| DWEN | 6 | debugWIRE Enable | 1 (unprogrammed) |
| SPIEN(2) | 5 | Enable Serial Program and Data Downloading | 0 (programmed, SPI programming enabled) |
| WDTON(3) | 4 | Watchdog Timer Always On | 1 (unprogrammed) |
| EESAVE | 3 | EEPROM memory is preserved through the Chip Erase | 1 (unprogrammed), EEPROM not reserved |
| BODLEVEL2(4) | 2 | Brown-out Detector trigger level | 1 (unprogrammed) |
| BODLEVEL1(4) | 1 | Brown-out Detector trigger level | 1 (unprogrammed) |
| BODLEVEL0(4) | 0 | Brown-out Detector trigger level | 1 (unprogrammed) |

**Note:**

1. Please refer to *Alternate Functions of Port C* in I/O-Ports chapter for description of RSTDISBL Fuse.
2. The SPIEN Fuse is not accessible in serial programming mode.
3. Please refer to *WDTCSR – Watchdog Timer Control Register* for details.
4. Please refer to Table. BODLEVEL Fuse Coding in *System and Reset Characteristics* for BODLEVEL Fuse decoding.

**Table 32-8. Fuse Low Byte**

| Low Fuse Byte | Bit No. | Description | Default Value |
|---|---|---|---|
| CKDIV8(4) | 7 | Divide clock by 8 | 0 (programmed) |
| CKOUT(3) | 6 | Clock output | 1 (unprogrammed) |
| SUT1 | 5 | Select start-up time | 1 (unprogrammed)(1) |
| SUT0 | 4 | Select start-up time | 0 (programmed)(1) |
| CKSEL3 | 3 | Select Clock source | 0 (programmed)(2) |
| CKSEL2 | 2 | Select Clock source | 0 (programmed)(2) |
| CKSEL1 | 1 | Select Clock source | 1 (unprogrammed)(2) |
| CKSEL0 | 0 | Select Clock source | 0 (programmed)(2) |

**Note:**

1. The default value of SUT[1:0] results in maximum start-up time for the default clock source. See Table. Start-up times for the internal calibrated RC Oscillator clock selection in *Calibrated Internal RC Oscillator* of System Clock and Clock Options chapter for details.
2. The default setting of CKSEL[3:0] results in internal RC Oscillator @ 8MHz. See Table. Internal Calibrated RC Oscillator Operating Modes in *Calibrated Internal RC Oscillator* of the System Clock and Clock Options chapter for details.
3. The CKOUT Fuse allows the system clock to be output on PORTB0. Please refer to *Clock Output Buffer* section in the System Clock and Clock Options chapter for details.
4. Please refer to *System Clock Prescaler* section in the System Clock and Clock Options chapter for details.

The status of the Fuse bits is not affected by Chip Erase. Note that the Fuse bits are locked if Lock bit1 (LB1) is programmed. Program the Fuse bits before programming the Lock bits.

**Related Links**

Alternate Port Functions on page 108
WDTCSR on page 82

### 32.2.1. 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.

## 32.3. Signature Bytes

The device have a three-byte signature code. This code can be read in both serial and parallel mode, also when the device is locked. The three bytes reside in a separate address space. For the device the signature bytes are given in the following table.

**Table 32-9. Device ID**

| Part | Signature Bytes Address | | |
|---|---|---|---|
| | 0x000 | 0x001 | 0x002 |
| ATmega48A | 0x1E | 0x92 | 0x05 |
| ATmega88A | 0x1E | 0x93 | 0x0A |
| ATmega168A | 0x1E | 0x94 | 0x06 |

## 32.4. Calibration Byte

The device has a byte calibration value for the Internal RC Oscillator. This byte resides in the high byte of address 0x000 in the signature address space. During reset, this byte is automatically written into the OSCCAL Register to ensure correct frequency of the calibrated RC Oscillator.

**Related Links**

Calibrated Internal RC Oscillator on page 57

## 32.5. Page Size

**Table 32-10. No. of Words in a Page and No. of Pages in the Flash**

| Device | Flash Size | Page Size | PCWORD | No. of Pages | PCPAGE | PCMSB |
|---|---|---|---|---|---|---|
| ATmega48A | 2K words (4Kbytes) | 32 words | PC[4:0] | 64 | PC[10:5] | 10 |
| ATmega88A | 4K words (8Kbytes) | 32 words | PC[4:0] | 128 | PC[11:5] | 11 |
| ATmega168A | 8K words (16Kbytes) | 64 words | PC[5:0] | 128 | PC[12:6] | 12 |

**Table 32-11. No. of Words in a Page and No. of Pages in the EEPROM**

| Device | EEPROM Size | Page Size | PCWORD | No. of Pages | PCPAGE | EEAMSB |
|---|---|---|---|---|---|---|
| ATmega48A | 256bytes | 4bytes | EEA[1:0] | 64 | EEA[7:2] | 7 |
| ATmega88A | 512bytes | 4bytes | EEA[1:0] | 128 | EEA[8:2] | 8 |
| ATmega168A | 512bytes | 4bytes | EEA[1:0] | 128 | EEA[8:2] | 8 |

## 32.6. Parallel Programming Parameters, Pin Mapping, and Commands

This section describes how to parallel program and verify Flash Program memory, EEPROM Data memory, Memory Lock bits, and Fuse bits in the device. Pulses are assumed to be at least 250ns unless otherwise noted.

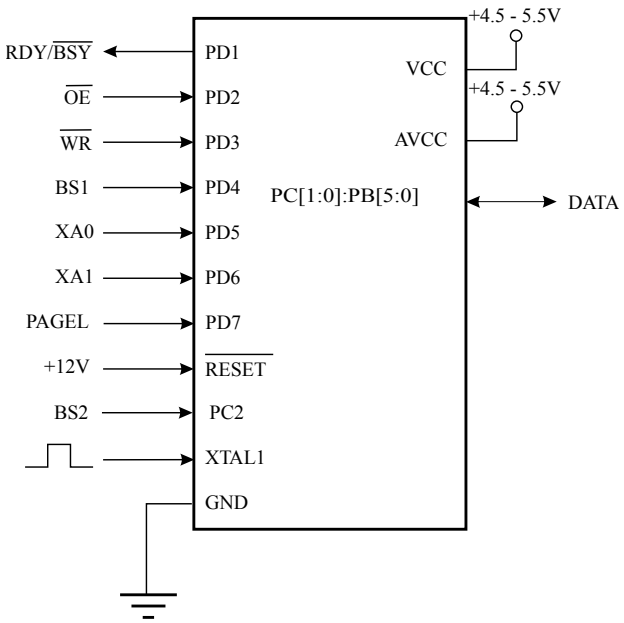
### 32.6.1. Signal Names

In this section, some pins of this device are referenced by signal names describing their functionality during parallel programming, please refer to Figure. Parallel Programming and Table. Pin Name Mapping in this section. Pins not described in the following table are referenced by pin names.

The XA1/XA0 pins determine the action executed when the XTAL1 pin is given a positive pulse. The bit coding is shown in the table, XA1 and XA0 Coding.

When pulsing $\overline{WR}$ or $\overline{OE}$, the command loaded determines the action executed. The different Commands are shown in the table, Command Byte Bit Coding Command Byte Command Executed.

**Figure 32-1. Parallel Programming**




Note: $V_{CC}$ - 0.3V < $AV_{CC}$ < $V_{CC}$ + 0.3V, however, $AV_{CC}$ should always be within 4.5 - 5.5V

**Table 32-12. Pin Name Mapping**

| Signal Name in Programming Mode | Pin Name | I/O | Function |
|---|---|---|---|
| RDY/$\overline{BSY}$ | PD1 | O | 0: Device is busy programming, 1: Device is ready for new command |
| $\overline{OE}$ | PD2 | I | Output Enable (Active low) |
| $\overline{WR}$ | PD3 | I | Write Pulse (Active low) |
| BS1 | PD4 | I | Byte Select 1 (“0” selects Low byte, “1” selects High byte) |
| XA0 | PD5 | I | XTAL Action Bit 0 |
| XA1 | PD6 | I | XTAL Action Bit 1 |
| PAGEL | PD7 | I | Program memory and EEPROM Data Page Load |
| BS2 | PC2 | I | Byte Select 2 (“0” selects Low byte, “1” selects 2’nd High byte) |
| DATA | {PC[1:0]: PB[5:0]} | I/O | Bi-directional Data bus (Output when OE is low) |

**Table 32-13. Pin Values Used to Enter Programming Mode**

| Pin | Symbol | Value |
|---|---|---|
| PAGEL | Prog_enable[3] | 0 |
| XA1 | Prog_enable[2] | 0 |
| XA0 | Prog_enable[1] | 0 |
| BS1 | Prog_enable[0] | 0 |

**Table 32-14. XA1 and XA0 Coding**

| XA1 | XA0 | Action when XTAL1 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 |

**Table 32-15. Command Byte Bit Coding**

| Command Byte | Command Executed |
|---|---|
| 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 |

## 32.7. Parallel Programming

### 32.7.1. Enter Programming Mode

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

1. Set Prog_enable pins listed in *Pin Values Used to Enter Programming Mode* of Signal Names section “0x0000”, $\overline{\text{RESET}}$ 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 1.8V within the next 20µs.
3. Wait 20 - 60µs, and apply 11.5 - 12.5V to $\overline{\text{RESET}}$.
4. Keep the Prog_enable pins unchanged for at least 10µs after the High-voltage has been applied to ensure the Prog_enable Signature has been latched.
5. Wait at least 300µs before giving any parallel programming commands.

6. Exit Programming mode by power the device down or by bringing $\overline{\text{RESET}}$ 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 listed in *Pin Values Used to Enter Programming Mode* of Signal Names section to “0000”, $\overline{\text{RESET}}$ 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 $\overline{\text{RESET}}$.
4. Keep the Prog_enable pins unchanged for at least 10µs after the High-voltage has been applied to ensure the 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 power the device down or by bringing $\overline{\text{RESET}}$ pin to 0V.

### 32.7.2. Considerations for Efficient Programming

The loaded command and address are retained in the device during programming. For efficient programming, the following should be considered.

- The command needs only be loaded once when writing or reading multiple memory locations.
- Skip writing the data value 0xFF, that is the contents of the entire EEPROM (unless the EESAVE Fuse is programmed) and Flash after a Chip Erase.
- Address high byte needs only be loaded before programming or reading a new 256 word window in Flash or 256byte EEPROM. This consideration also applies to Signature bytes reading.

### 32.7.3. Chip Erase

The Chip Erase will erase the Flash, the SRAM and the EEPROM memories plus Lock bits. The Lock bits are not reset until the program memory has been completely erased. The Fuse bits are not changed. A Chip Erase must be performed before the Flash and/or EEPROM are reprogrammed.

**Note:** The EEPRPOM memory is preserved during Chip Erase if the EESAVE Fuse is programmed.

Load Command “Chip Erase”:

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 XTAL1 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.

### 32.7.4. Programming the Flash

The Flash is organized in pages as number of Words in a Page and number of Pages in the Flash. When programming the Flash, the program data is 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:

**Step 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 XTAL1 a positive pulse. This loads the command.

**Step 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 XTAL1 a positive pulse. This loads the address low byte.

**Step 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 XTAL1 a positive pulse. This loads the data byte.

**Step 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 XTAL1 a positive pulse. This loads the data byte.

**Step E. Latch Data**

1. Set BS1 to “1”. This selects high data byte.
2. Give PAGEL a positive pulse. This latches the data bytes. (Please refer to the figure, Programming the Flash Waveforms, in this section for signal waveforms)

**Step 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 the following figure, Addressing the Flash Which is Organized in Pages, in this section. 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.

**Step G. Load Address High Byte**

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

**Step 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 (Please refer to the figure, Programming the Flash Waveforms, in this section for signal waveforms).

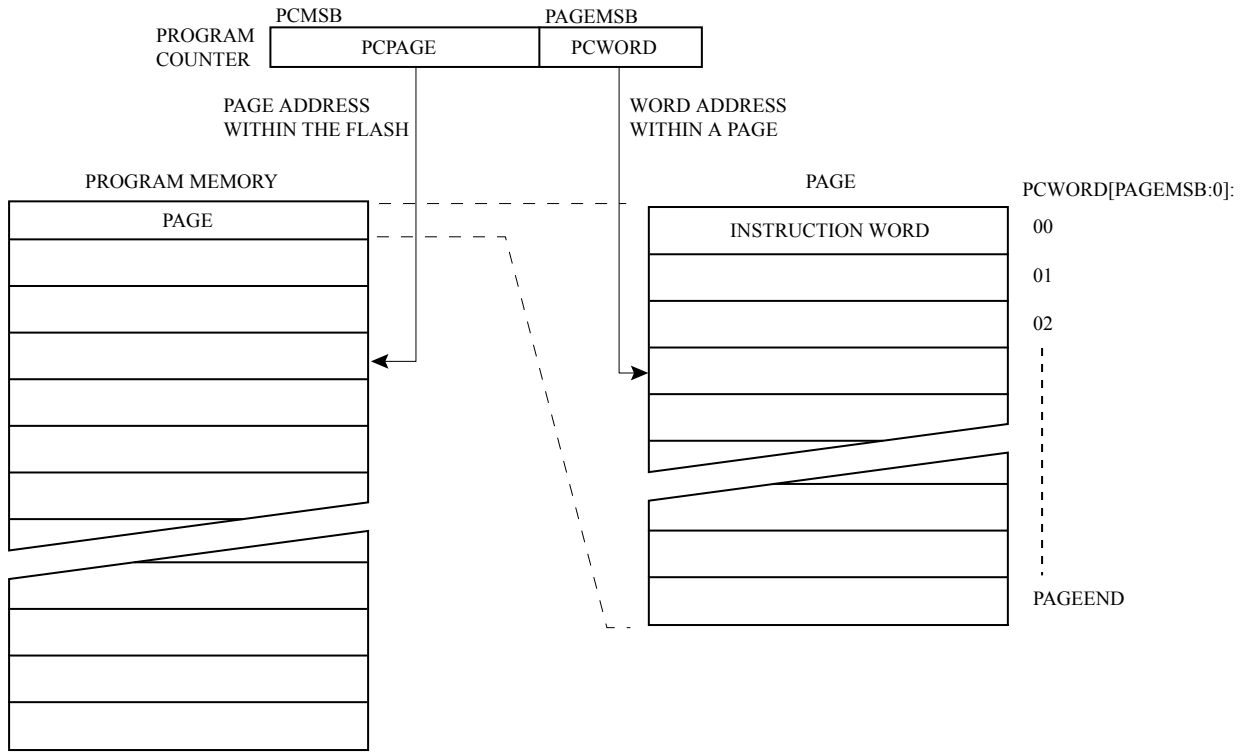
**Step I. Repeat B Through H Until the Entire Flash Is Programmed or Until All Data Has Been Programmed**

**Step J. End Page Programming**

1. 1. Set XA1, XA0 to “10”. This enables command loading.

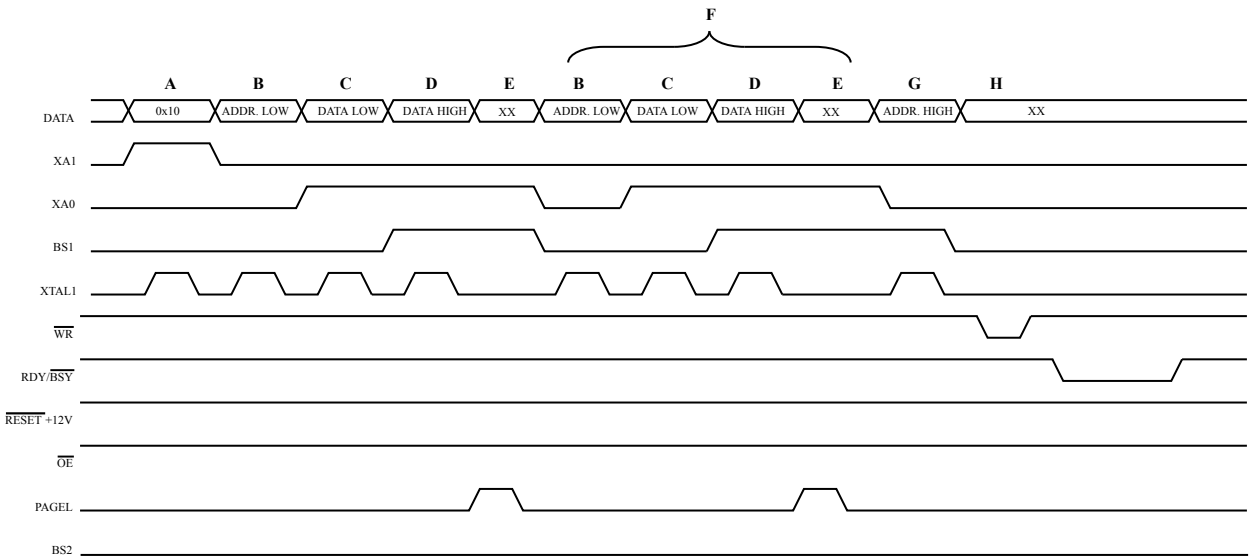
2. Set DATA to “0000 0000”. This is the command for No Operation.
3. Give XTAL1 a positive pulse. This loads the command, and the internal write signals are reset.

**Figure 32-2. Addressing the Flash Which Is Organized in Pages**




**Note:** PCPAGE and PCWORD are listed in the table of *No. of Words in a Page and No. of Pages* in the Flash in *Page Size* section.

Programming the Flash Waveforms




**Note:** “XX” is don’t care. The letters refer to the programming description above.

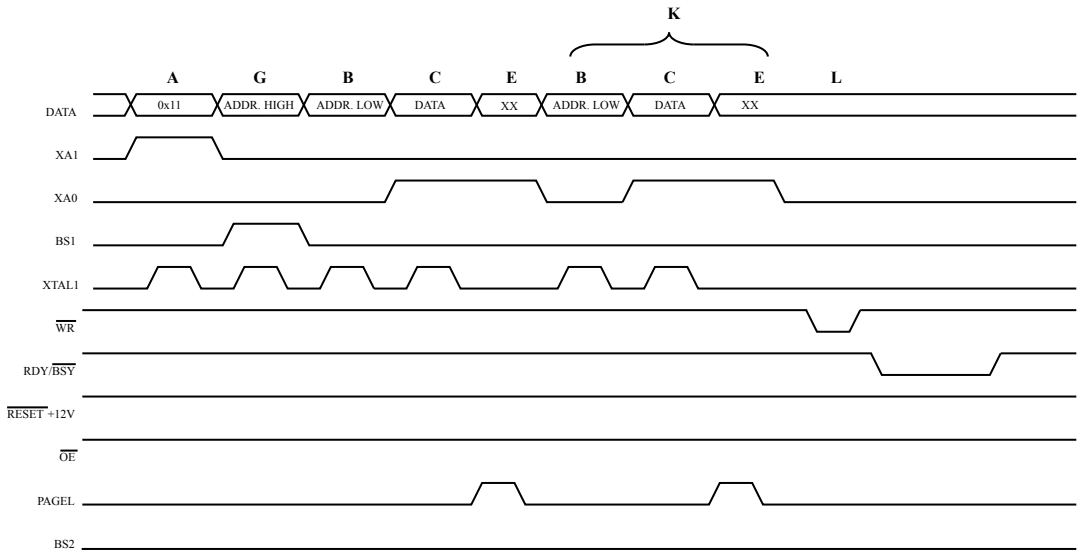
### 32.7.5. Programming the EEPROM

The EEPROM is organized in pages, please refer to table, No. of Words in a Page and No. of Pages in the EEPROM, in the Page Size section. 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 (For details on Command, Address and Data loading, please refer to Programming the Flash):

1. Step A: Load Command “0001 0001”.
2. Step G: Load Address High Byte (0x00 - 0xFF).
3. Step B: Load Address Low Byte (0x00 - 0xFF).
4. Step C: Load Data (0x00 - 0xFF).
5. Step E: Latch data (give PAGEL a positive pulse).
6. Step K:Repeat 3 through 5 until the entire buffer is filled.
7. Step L: Program EEPROM page
   - 7.1. Set BS1 to “0”.
   - 7.2. Give WR a negative pulse. This starts programming of the EEPROM page. RDY/BSY goes low.
   - 7.3. Wait until to RDY/BSY goes high before programming the next page (Please refer to the following figure for signal waveforms).

**Figure 32-3. Programming the EEPROM Waveforms**




### 32.7.6. Reading the Flash

The algorithm for reading the Flash memory is as follows (Please refer to Programming the Flash in this chapter for details on Command and Address loading):

1. Step A: Load Command “0000 0010”.
2. Step G: Load Address High Byte (0x00 - 0xFF).
3. Step B: Load Address Low Byte (0x00 - 0xFF).
4. Set $\overline{OE}$ to “0”, and BS1 to “0”. The Flash word low byte can now be read at DATA.
5. Set BS1 to “1”. The Flash word high byte can now be read at DATA.
6. Set $\overline{OE}$ to “1”.

### 32.7.7. Reading the EEPROM

The algorithm for reading the EEPROM memory is as follows (Please refer to Programming the Flash for details on Command and Address loading):

1. Step A: Load Command “0000 0011”.
2. Step G: Load Address High Byte (0x00 - 0xFF).
3. Step B: Load Address Low Byte (0x00 - 0xFF).
4. Set $\overline{OE}$ to “0”, and BS1 to “0”. The EEPROM Data byte can now be read at DATA.
5. Set $\overline{OE}$ to “1”.

### 32.7.8. Programming the Fuse Low Bits

The algorithm for programming the Fuse Low bits is as follows (Please refer to Programming the Flash for details on Command and Data loading):

1. Step A: Load Command “0100 0000”.
2. Step C: Load Data Low Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.

### 32.7.9. Programming the Fuse High Bits

The algorithm for programming the Fuse High bits is as follows (Please refer to Programming the Flash for details on Command and Data loading):

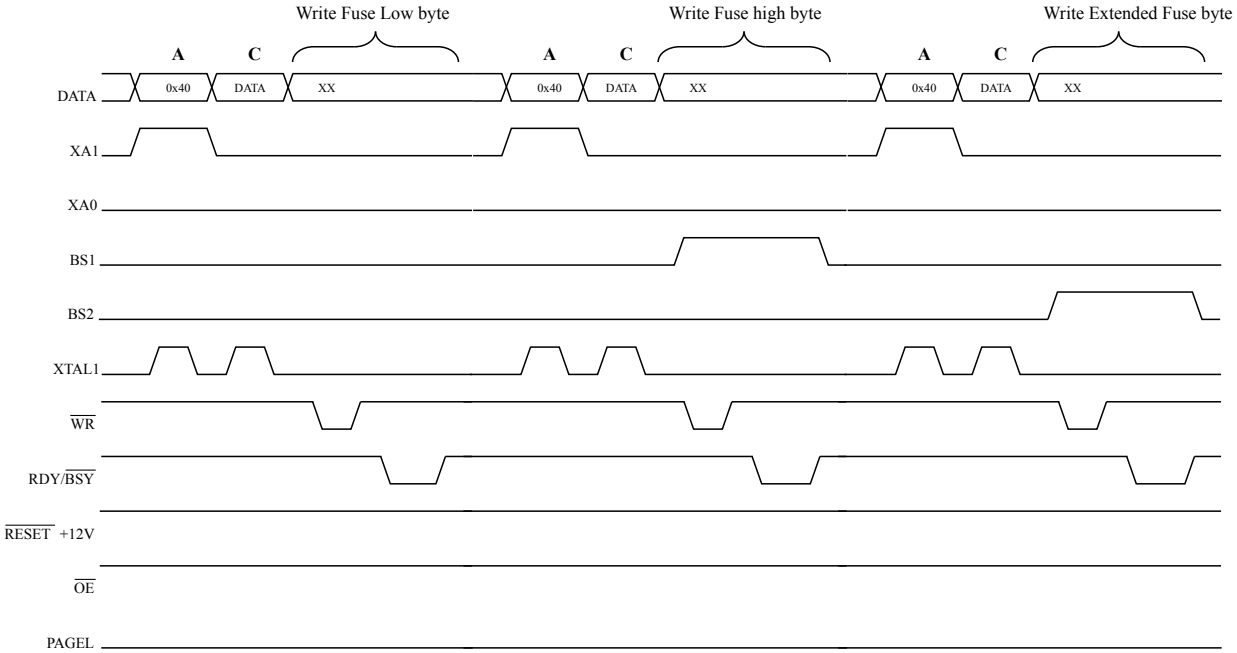
1. Step A: Load Command “0100 0000”.
2. Step C: Load Data Low Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Set BS1 to “1” and BS2 to “0”. This selects high data byte.
4. Give WR a negative pulse and wait for RDY/BSY to go high.
5. Set BS1 to “0”. This selects low data byte.

### 32.7.10. Programming the Extended Fuse Bits

The algorithm for programming the Extended Fuse bits is as follows (Please refer to Programming the Flash for details on Command and Data loading):

1. Step A: Load Command “0100 0000”.
2. Step C: Load Data Low Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Set BS1 to “0” and BS2 to “1”. This selects extended data byte.
4. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.
5. Set BS2 to “0”. This selects low data byte.

**Figure 32-4. Programming the FUSES Waveforms**




### 32.7.11. Programming the Lock Bits

The algorithm for programming the Lock bits is as follows (Please refer to Programming the Flash for details on Command and Data loading):

1. Step A: Load Command “0010 0000”.
2. Step C: Load Data Low Byte. Bit n = “0” programs the Lock bit. If LB mode 3 is programmed (LB1 and LB2 is programmed), it is not possible to program the Boot Lock bits by any External Programming mode.
3. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.

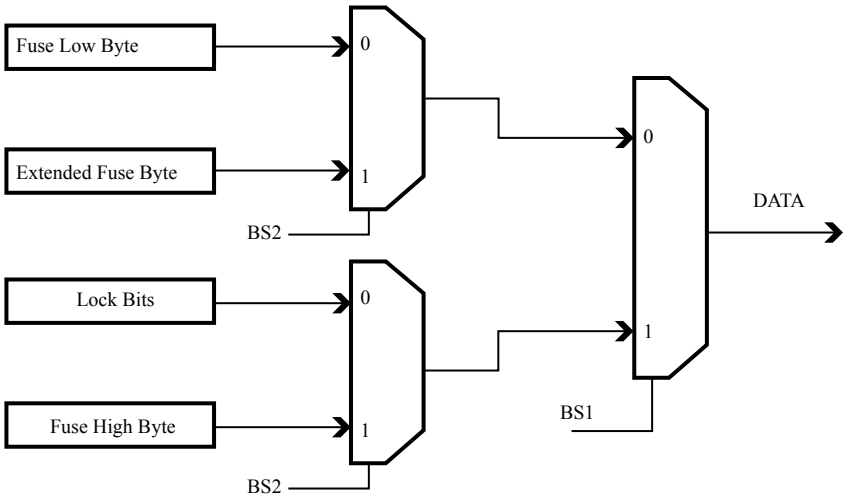
The Lock bits can only be cleared by executing Chip Erase.

### 32.7.12. Reading the Fuse and Lock Bits

The algorithm for reading the Fuse and Lock bits is as follows (Please refer to Programming the Flash for details on Command loading):

1. Step A: Load Command “0000 0100”.
2. Set $\overline{OE}$ to “0”, BS2 to “0” and BS1 to “0”. The status of the Fuse Low bits can now be read at DATA (“0” means programmed).
3. Set $\overline{OE}$ to “0”, BS2 to “1” and BS1 to “1”. The status of the Fuse High bits can now be read at DATA (“0” means programmed).
4. Set $\overline{OE}$ to “0”, BS2 to “1”, and BS1 to “0”. The status of the Extended Fuse bits can now be read at DATA (“0” means programmed).
5. Set $\overline{OE}$ to “0”, BS2 to “0” and BS1 to “1”. The status of the Lock bits can now be read at DATA (“0” means programmed).
6. Set $\overline{OE}$ to “1”.

**Figure 32-5. Mapping Between BS1, BS2 and the Fuse and Lock Bits During Read**




### 32.7.13. Reading the Signature Bytes

The algorithm for reading the Signature bytes is as follows (Please refer to Programming the Flash for details on Command and Address loading):

1. Step A: Load Command “0000 1000”.
2. Step 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”.

### 32.7.14. Reading the Calibration Byte

The algorithm for reading the Calibration byte is as follows (Please refer to Programming the Flash for details on Command and Address loading):

1. Step A: Load Command “0000 1000”.
2. Step 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”.

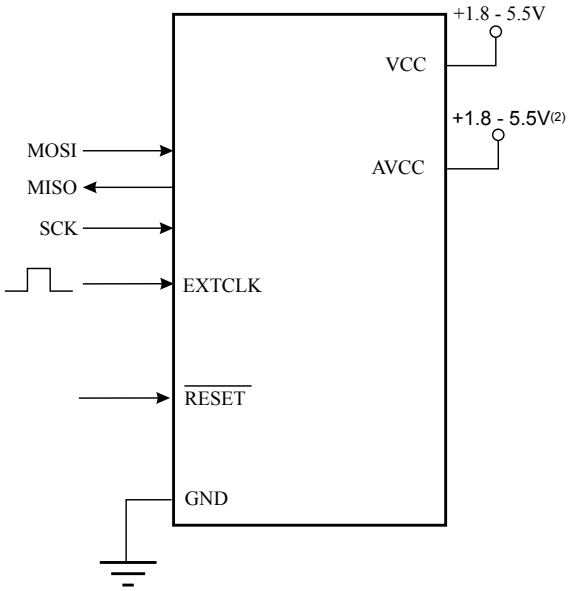
### 32.7.15. Parallel Programming Characteristics

For characteristics of the Parallel Programming, please refer to *Parallel Programming Characteristics*.

## 32.8. Serial Downloading

Both the Flash and EEPROM memory arrays can 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 first before program/erase operations can be executed.

**Figure 32-6. Serial Programming and Verify**




**Note:**

1. If the device is clocked by the internal Oscillator, it is no need to connect a clock source to the XTAL1 pin.
2. $V_{CC}$ - 0.3V < AVCC < $V_{CC}$ + 0.3V, however, AVCC should always be within 1.8 - 5.5V

When programming the EEPROM, an auto-erase cycle is built into the self-timed programming operation (in the Serial mode ONLY) and there is no need to first execute the Chip Erase instruction. The Chip Erase operation turns the content of every memory location in both the Program 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:

- Low: > 2 CPU clock cycles for $f_{ck}$ < 12MHz, 3 CPU clock cycles for $f_{ck}$ ≥ 12MHz
- High: > 2 CPU clock cycles for $f_{ck}$ < 12MHz, 3 CPU clock cycles for $f_{ck}$ ≥ 12MHz

### 32.8.1. Serial Programming Pin Mapping

**Table 32-16. Pin Mapping Serial Programming**

| Symbol | Pins | I/O | Description |
|---|---|---|---|
| MOSI | PB3 | I | Serial Data in |
| MISO | PB4 | O | Serial Data out |
| SCK | PB5 | I | Serial Clock |

**Note:** The pin mapping for SPI programming is listed. Not all parts use the SPI pins dedicated for the internal SPI interface.

### 32.8.2. Serial Programming Algorithm

When writing serial data to the device, data is clocked on the rising edge of SCK.

When reading data from the device, data is clocked on the falling edge of SCK. Please refer to the figure, Serial Programming Waveforms in SPI Serial Programming Characteristics section for timing details.

To program and verify the device in the serial programming mode, the following sequence is recommended (See Serial Programming Instruction set in Table 32-18:

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, RESET must be given a positive pulse of at least two CPU clock cycles duration after SCK has been set to “0”.
2. Wait for at least 20ms and enable serial programming by sending the Programming Enable serial instruction to pin MOSI.
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. Whether 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, the data low byte must be loaded before data high byte is applied for a given address. 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 . Accessing the serial programming interface before the Flash write operation completes can result in incorrect programming.
5. A: The EEPROM array is programmed one byte at a time by supplying the address and data together with the appropriate Write instruction. An EEPROM memory location is first 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. In a chip erased device, no 0xFFs in the data file(s) need to be programmed.
   B: 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 is altered. The remaining locations remain unchanged. If polling (RDY/$\overline{\text{BSY}}$) is not used, the used must wait at least $t_{WD_EEPROM}$ before issuing the next byte. 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 serial output 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 needed):
   Set $\overline{\text{RESET}}$ to “1”.

   Turn $V_{CC}$ power off.

**Table 32-17. Typical Wait Delay Before Writing the Next Flash or EEPROM Location**

| Symbol | Minimum Wait Delay |
|---|---|
| $t_{WD_FLASH}$ | 2.6ms |
| $t_{WD_EEPROM}$ | 3.6ms |

| Symbol | Minimum Wait Delay |
|---|---|
| $t_{WD_ERASE}$ | 10.5ms |
| $t_{WD_FUSE}$ | 4.5ms |

### 32.8.3. Serial Programming Instruction Set

This section describes the Instruction Set.

**Table 32-18. Serial Programming Instruction Set (Hexadecimal values)**

| Instruction/Operation | Instruction Format | | | |
|---|---|---|---|---|
| | Byte 1 | Byte 2 | Byte 3 | Byte 4 |
| Programming Enable | 0xAC | 0x53 | 0x00 | 0x00 |
| Chip Erase (Program Memory/EEPROM) | 0xAC | 0x80 | 0x00 | 0x00 |
| Poll RDY/$\overline{BSY}$ | 0xF0 | 0x00 | 0x00 | data byte out |
| Load Instructions | | | | |
| Load Extended Address byte(1) | 0x4D | 0x00 | Extended adr | 0x00 |
| Load Program Memory Page, High byte | 0x48 | 0x00 | adr LSB | high data byte in |
| Load Program Memory Page, Low byte | 0x40 | 0x00 | adr LSB | low data byte in |
| Load EEPROM Memory Page (page access) | 0xC1 | 0x00 | 0000 000aa | data byte in |
| Read Instructions | | | | |
| Read Program Memory, High byte | 0x28 | adr MSB | adr LSB | high data byte out |
| Read Program Memory, Low byte | 0x20 | adr MSB | adr LSB | low data byte out |
| Read EEPROM Memory | 0xA0 | 0000 00aa | aaaa aaaa | data byte out |
| Read Lock bits | 0x58 | 0x00 | 0x00 | data byte out |
| Read Signature Byte | 0x30 | 0x00 | 0000 000aa | data byte out |
| Read Fuse bits | 0x50 | 0x00 | 0x00 | data byte out |
| Read Fuse High bits | 0x58 | 0x08 | 0x00 | data byte out |
| Read Extended Fuse Bits | 0x50 | 0x08 | 0x00 | data byte out |
| Read Calibration Byte | 0x38 | 0x00 | 0x00 | data byte out |
| Write Instructions(6) | | | | |
| Write Program Memory Page | 0x4C | adr MSB(8) | adr LSB(8) | 0x00 |
| Write EEPROM Memory | 0xC0 | 0000 00aa | aaaa aaaa | data byte in |
| Write EEPROM Memory Page (page access) | 0xC2 | 0000 00aa | aaaa aa00 | 0x00 |
| Write Lock bits | 0xAC | 0xE0 | 0x00 | data byte in |
| Write Fuse bits | 0xAC | 0xA0 | 0x00 | data byte in |

| Instruction/Operation | Instruction Format | | | |
|---|---|---|---|---|
| | Byte 1 | Byte 2 | Byte 3 | Byte 4 |
| Write Fuse High bits | 0xAC | 0xA8 | 0x00 | data byte in |
| Write Extended Fuse Bits | 0xAC | 0xA4 | 0x00 | data byte in |

**Note:**

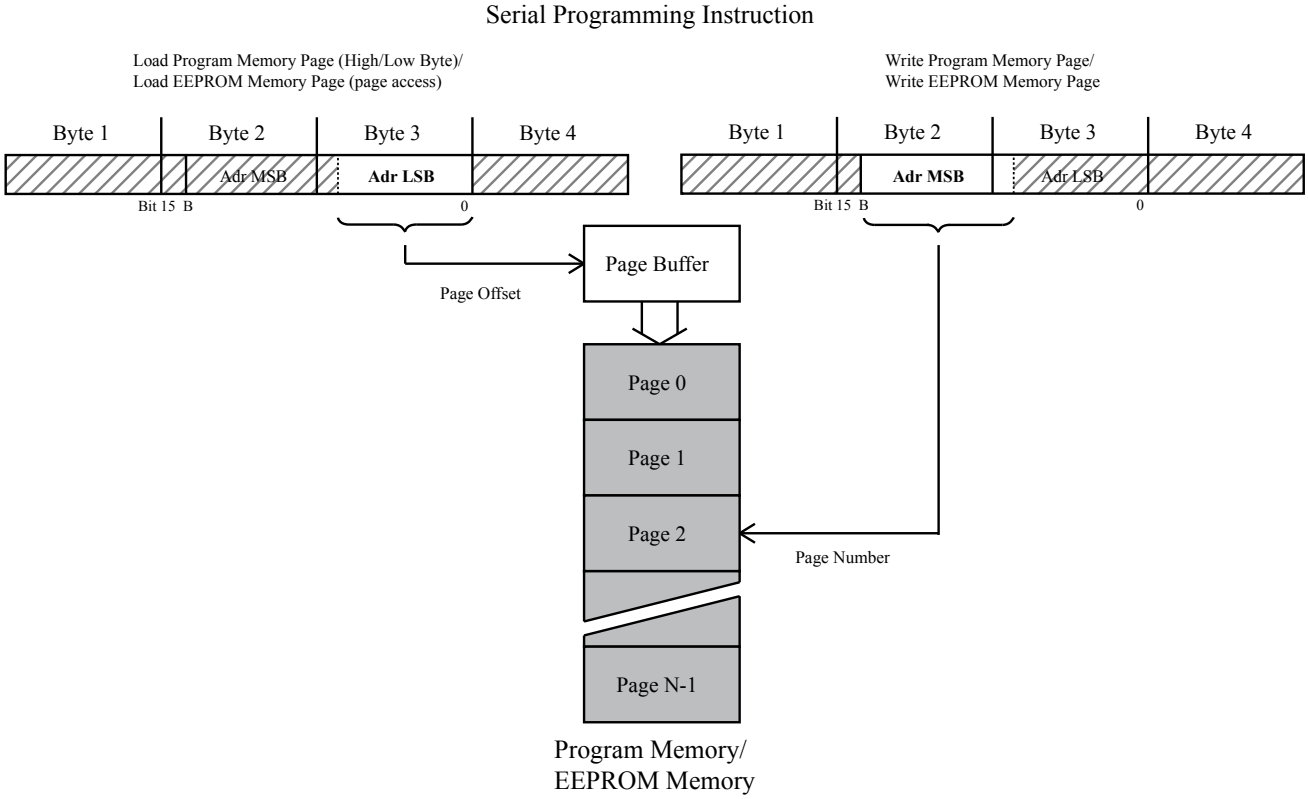
1. Not all instructions are applicable for all parts.
2. a = address.
3. Bits are programmed ‘0’, unprogrammed ‘1’.
4. To ensure future compatibility, unused Fuses and Lock bits should be unprogrammed (‘1’) .
5. Refer to the corresponding section for Fuse and Lock bits, Calibration and Signature bytes and Page size.
6. Instructions accessing program memory use a word address. This address may be random within the page range.
7. See http://www.atmel.com/avr for Application Notes regarding programming and programmers.
8. WORDS.

If the LSB in 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.

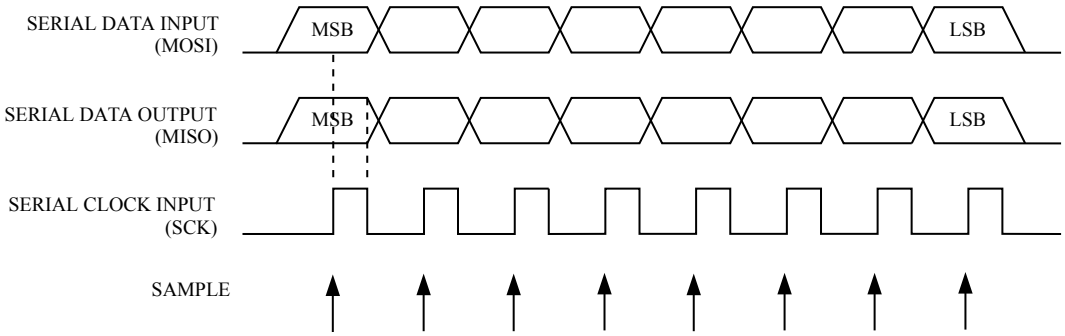
After data is loaded to the page buffer, program the EEPROM page, Please refer to the following figure.

**Figure 32-7. Serial Programming Instruction example**



### 32.8.4. SPI Serial Programming Characteristics

**Figure 32-8. Serial Programming Waveforms**



# 33. Electrical Characteristics

## 33.1. Absolute Maximum Ratings

**Table 33-1. 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 | 40.0mA |
| DC Current $V_{CC}$ and GND Pins | 200.0mA |

**Note:** 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.

## 33.2. Common DC Characteristics

**Table 33-2. Common DC characteristics $T_A$ = -40°C to 85°C, $V_{CC}$ = 1.8V to 5.5V (unless otherwise noted)**

| Symbol | Parameter | Condition | Min. | Typ. | Max. | Units |
|---|---|---|---|---|---|---|
| $V_{IL}$ | Input Low Voltage, except XTAL1 and $\overline{\text{RESET}}$ pin | $V_{CC}$ = 1.8V - 2.4V | -0.5 | | $0.2V_{CC}$(1) | V |
| | | $V_{CC}$ = 2.4V - 5.5V | -0.5 | | $0.3V_{CC}$(1) | |
| $V_{IH}$ | Input High Voltage, except XTAL1 and $\overline{\text{RESET}}$ pins | $V_{CC}$ = 1.8V - 2.4V | $0.7V_{CC}$(2) | | $V_{CC}$ + 0.5 | V |
| | | $V_{CC}$ = 2.4V - 5.5V | $0.6V_{CC}$(2) | | $V_{CC}$ + 0.5 | |
| $V_{IL1}$ | Input Low Voltage, XTAL1 pin | $V_{CC}$ = 1.8V - 5.5V | -0.5 | | $0.1V_{CC}$(1) | V |
| $V_{IH1}$ | Input High Voltage, XTAL1 pin | $V_{CC}$ = 1.8V - 2.4V | $0.8V_{CC}$(2) | | $V_{CC}$ + 0.5 | V |
| | | $V_{CC}$ = 2.4V - 5.5V | $0.7V_{CC}$(2) | | $V_{CC}$ + 0.5 | |
| $V_{IL2}$ | Input Low Voltage, $\overline{\text{RESET}}$ pin | $V_{CC}$ = 1.8V - 5.5V | -0.5 | | $0.1V_{CC}$(1) | V |
| $V_{IH2}$ | Input High Voltage, $\overline{\text{RESET}}$ pin | $V_{CC}$ = 1.8V - 5.5V | $0.9V_{CC}$(2) | | $V_{CC}$ + 0.5 | V |
| $V_{IL3}$ | Input Low Voltage, $\overline{\text{RESET}}$ pin as I/O | $V_{CC}$ = 1.8V - 2.4V | -0.5 | | $0.2V_{CC}$(1) | V |
| | | $V_{CC}$ = 2.4V - 5.5V | -0.5 | | $0.3V_{CC}$(1) | |

| Symbol | Parameter | Condition | | Min. | Typ. | Max. | Units |
|---|---|---|---|---|---|---|---|
| $V_{IH3}$ | Input High Voltage, $\overline{\text{RESET}}$ pin as I/O | $V_{CC}$ = 1.8V - 2.4V | | 0.7$V_{CC}$(2) | | $V_{CC}$ + 0.5 | V |
| | | $V_{CC}$ = 2.4V - 5.5V | | 0.6$V_{CC}$(2) | | $V_{CC}$ + 0.5 | |
| $V_{OL}$ | Output Low Voltage(4) except RESET pin | $I_{OL}$ = 20mA, $V_{CC}$ = 5V | $T_A$=85°C | | | 0.9 | V |
| | | $I_{OL}$ = 10mA, $V_{CC}$ = 3V | $T_A$=85°C | | | 0.6 | V |
| $V_{OH}$ | Output High Voltage(3) except Reset pin | $I_{OH}$ = -20mA, $V_{CC}$ = 5V | $T_A$=85°C | 4.2 | | | V |
| | | $I_{OH}$ = -10mA, $V_{CC}$ = 3V | $T_A$=85°C | 2.3 | | | V |
| $I_{IL}$ | Input Leakage Current I/O Pin | $V_{CC}$ = 5.5V, pin low (absolute value) | | | | 1 | µA |
| $I_{IH}$ | Input Leakage Current I/O Pin | $V_{CC}$ = 5.5V, pin high (absolute value) | | | | 1 | µA |
| $R_{RST}$ | Reset Pull-up Resistor | | | 30 | | 60 | kΩ |
| $R_{PU}$ | I/O Pin Pull-up Resistor | | | 20 | | 50 | kΩ |
| $V_{ACIO}$ | Analog Comparator Input Offset Voltage | $V_{CC}$ = 5V $V_{in}$ = $V_{CC}$/2 | | | <10 | 40 | mV |
| $I_{ACLK}$ | Analog Comparator Input Leakage Current | $V_{CC}$ = 5V $V_{in}$ = $V_{CC}$/2 | | -50 | | 50 | nA |
| $t_{ACID}$ | Analog Comparator Propagation Delay | $V_{CC}$ = 2.7V $V_{CC}$ = 4.0V | | | 750 500 | | ns |

**Note:**

1. “Max.” means the highest value where the pin is guaranteed to be read as low.
2. “Min.” means the lowest value where the pin is guaranteed to be read as high.
3. Although each I/O port can source more than the test conditions (20mA at $V_{CC}$ = 5V, 10mA at $V_{CC}$ = 3V) under steady state conditions (non-transient), the following must be observed:
   3.1. The sum of all $I_{OH}$, for ports C0 - C5, D0- D4, ADC7, $\overline{\text{RESET}}$ should not exceed 100mA.
   3.2. The sum of all $I_{OH}$, for ports B0 - B5, D5 - D7, ADC6, XTAL1, XTAL2 should not exceed 100mA.

If $II_{OH}$ exceeds the test condition, $V_{OH}$ may exceed the related specification. Pins are not guaranteed to source current greater than the listed test condition.

4. Although each I/O port can sink more than the test conditions (20mA at $V_{CC}$ = 5V, 10mA at $V_{CC}$ = 3V) under steady state conditions (non-transient), the following must be observed:
   4.1. The sum of all $I_{OL}$, for ports C0 - C5, ADC7, ADC6 should not exceed 100mA.
   4.2. The sum of all $I_{OL}$, for ports B0 - B5, D5 - D7, XTAL1, XTAL2 should not exceed 100mA.
   4.3. The sum of all $I_{OL}$, for ports D0 - D4, $\overline{\text{RESET}}$ should not exceed 100mA.

   If $I_{OL}$ exceeds the test condition, $V_{OL}$ may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition.

**Related Links**

Minimizing Power Consumption on page 68

### 33.2.1. ATmega48A DC Characteristics – Current Consumption

**Table 33-3. DC characteristics - $T_A$ = -40°C to 85°C, $V_{CC}$ = 1.8V to 5.5V (unless otherwise noted)**

| Symbol | Parameter | Condition | | Min. | Typ.(2) | Max. | Units |
|---|---|---|---|---|---|---|---|
| $I_{CC}$ | Power Supply Current(1) | Active 1MHz, $V_{CC}$ = 2V | T = 85°C | | 0.2 | 0.55 | mA |
| | | Active 4MHz, $V_{CC}$ = 3V | T = 85°C | | 1.2 | 3.5 | |
| | | Active 8MHz, $V_{CC}$ = 5V | T = 85°C | | 4.0 | 12 | |
| | | Idle 1MHz, $V_{CC}$ = 2V | T = 85°C | | 0.03 | 0.5 | |
| | | Idle 4MHz, $V_{CC}$ = 3V | T = 85°C | | 0.21 | 1.5 | |
| | | Idle 8MHz,$V_{CC}$ = 5V | T = 85°C | | 0.9 | 5.5 | |
| | Power-save mode(3) | 32kHz TOSC enabled, $V_{CC}$ = 1.8V | T = 85°C | | 0.75 | | µA |
| | | 32kHz TOSC enabled, $V_{CC}$ = 3V | T = 85°C | | 0.9 | | |
| | Power-down mode(3) | WDT enabled, $V_{CC}$ = 3V | T = 85°C | | 3.9 | 15 | |
| | | WDT disabled, $V_{CC}$ = 3V | T = 85°C | | 0.1 | 2 | |

**Note:**

1. Values with *Minimizing Power Consumption* enabled (0xFF).
2. Typical values at 25°C. Maximum values are test limits in production.
3. The current consumption values include input leakage current.

### 33.2.2. ATmega88A DC Characteristics – Current Consumption

**Table 33-4. DC characteristics - $T_A$ = -40°C to 85°C, $V_{CC}$ = 1.8V to 5.5V (unless otherwise noted)**

| Symbol | Parameter | Condition | | Min. | Typ.(2) | Max. | Units |
|---|---|---|---|---|---|---|---|
| $I_{CC}$ | Power Supply Current(1) | Active 1MHz, $V_{CC}$ = 2V | T = 85°C | | 0.2 | 0.55 | mA |
| | | Active 4MHz, $V_{CC}$ = 3V | T = 85°C | | 1.2 | 3.5 | |
| | | Active 8MHz, $V_{CC}$ = 5V | T = 85°C | | 4.1 | 12 | |
| | | Idle 1MHz, $V_{CC}$ = 2V | T = 85°C | | 0.03 | 0.5 | |
| | | Idle 4MHz, $V_{CC}$ = 3V | T = 85°C | | 0.18 | 1.5 | |
| | | Idle 8MHz,$V_{CC}$ = 5V | T = 85°C | | 0.8 | 5.5 | |
| | Power-save mode(3) | 32kHz TOSC enabled, $V_{CC}$ = 1.8V | T = 85°C | | 0.8 | | µA |
| | | 32kHz TOSC enabled, $V_{CC}$ = 3V | T = 85°C | | 0.9 | | |
| | Power-down mode(3) | WDT enabled, $V_{CC}$ = 3V | T = 85°C | | 3.9 | 15 | |
| | | WDT disabled, $V_{CC}$ = 3V | T = 85°C | | 0.1 | 2 | |

**Note:**

1. Values with *Minimizing Power Consumption* enabled (0xFF).
2. Typical values at 25°C. Maximum values are test limits in production.
3. The current consumption values include input leakage current.

### 33.2.3. ATmega168A DC Characteristics – Current Consumption

**Table 33-5. DC characteristics - $T_A$ = -40°C to 85°C, $V_{CC}$ = 1.8V to 5.5V (unless otherwise noted)**

| Symbol | Parameter | Condition | | Min. | Typ.(2) | Max. | Units |
|---|---|---|---|---|---|---|---|
| $I_{CC}$ | Power Supply Current(1) | Active 1MHz, $V_{CC}$ = 2V | T = 85°C | | 0.2 | 0.55 | mA |
| | | Active 4MHz, $V_{CC}$ = 3V | T = 85°C | | 1.2 | 3.5 | |
| | | Active 8MHz, $V_{CC}$ = 5V | T = 85°C | | 4.2 | 12 | |
| | | Idle 1MHz, $V_{CC}$ = 2V | T = 85°C | | 0.03 | 0.5 | |
| | | Idle 4MHz, $V_{CC}$ = 3V | T = 85°C | | 0.2 | 1.5 | |
| | | Idle 8MHz,$V_{CC}$ = 5V | T = 85°C | | 0.9 | 5.5 | |
| | Power-save mode(3) | 32kHz TOSC enabled, $V_{CC}$ = 1.8V | T = 85°C | | 0.75 | | µA |
| | | 32kHz TOSC enabled, $V_{CC}$ = 3V | T = 85°C | | 0.83 | | |
| | Power-down mode(3) | WDT enabled, $V_{CC}$ = 3V | T = 85°C | | 4.1 | 15 | |
| | | WDT disabled, $V_{CC}$ = 3V | T = 85°C | | 0.1 | 2 | |

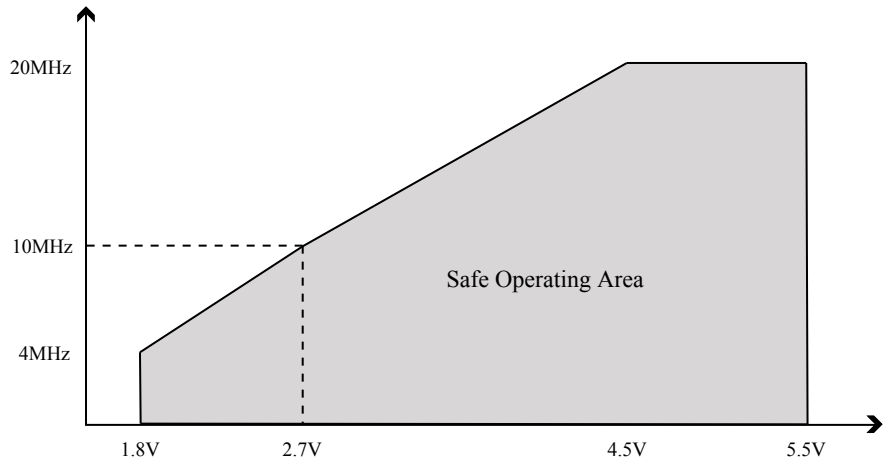
**Note:**

1. Values with *Minimizing Power Consumption* enabled (0xFF).
2. Typical values at 25°C. Maximum values are test limits in production.
3. The current consumption values include input leakage current.

## 33.3. Speed Grades

Maximum frequency is dependent on $V_{CC}$. As shown in Figure. Maximum Frequency vs. $V_{CC}$, the Maximum Frequency vs. $V_{CC}$ curve is linear between 1.8V < $V_{CC}$ < 2.7V and between 2.7V < $V_{CC}$ < 4.5V.

**Figure 33-1. Maximum Frequency vs. $V_{CC}$**




## 33.4. Clock Characteristics

**Related Links**

Calibrated Internal RC Oscillator on page 57

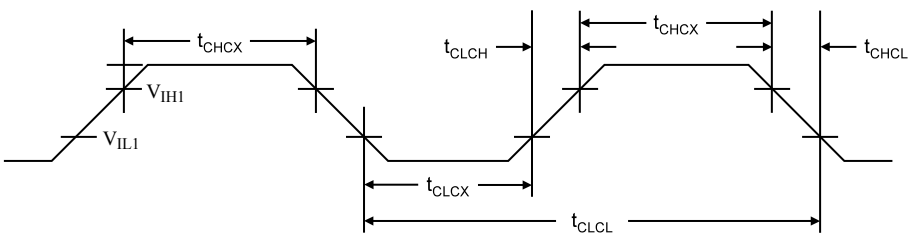
### 33.4.1. Calibrated Internal RC Oscillator Accuracy

**Table 33-6. Calibration Accuracy of Internal RC Oscillator**

| | Frequency | $V_{CC}$ | Temperature | Calibration Accuracy |
|---|---|---|---|---|
| Factory Calibration | 8.0MHz | 3.0V | 25°C | ±10% |
| User Calibration | 7.3 - 8.1MHz | 1.8V - 5.5V | -40°C to - 85°C | ±1% |

### 33.4.2. External Clock Drive Waveforms

**Figure 33-2. External Clock Drive Waveforms**



### 33.4.3. External Clock Drive

**Table 33-7. External Clock Drive**

| Symbol | Parameter | $V_{CC}$= 1.8 - 5.5V | | $V_{CC}$= 2.7 - 5.5V | | $V_{CC}$= 4.5 - 5.5V | | Units |
|---|---|---|---|---|---|---|---|---|
| | | Min. | Max. | Min. | Max. | Min. | Max. | |
| $1/t_{CLCL}$ | Oscillator Frequency | 0 | 4 | 0 | 10 | 0 | 20 | MHz |
| $t_{CLCL}$ | Clock Period | 250 | - | 100 | - | 50 | - | 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 |
| $\Delta t_{CLCL}$ | Change in period from one clock cycle to the next | - | 2 | - | 2 | - | 2 | % |

## 33.5. System and Reset Characteristics

**Table 33-8. Reset, Brown-out and Internal Voltage Characteristics(1)**

| Symbol | Parameter | Condition | Min. | Typ | Max | Units |
|---|---|---|---|---|---|---|
| $V_{POT}$ | Power-on Reset Threshold Voltage (rising) | | 1.1 | 1.5 | 1.7 | V |
| | Power-on Reset Threshold Voltage (falling)(2) | | 0.6 | 1.0 | 1.7 | V |
| $SR_{ON}$ | Power-on Slope Rate | | 0.01 | - | 10 | V/ms |
| $V_{RST}$ | $\overline{\text{RESET}}$ Pin Threshold Voltage | | 0.2 $V_{CC}$ | - | 0.9 $V_{CC}$ | V |
| $t_{RST}$ | Minimum pulse width on $\overline{\text{RESET}}$ Pin | | - | - | 2.5 | µs |
| $V_{HYST}$ | Brown-out Detector Hysteresis | | - | 50 | - | mV |
| $t_{BOD}$ | Min. Pulse Width on Brown-out Reset | | - | 2 | - | µs |
| $V_{BG}$ | Bandgap reference voltage | $V_{CC}$=2.7 $T_A$=25°C | 1.0 | 1.1 | 1.2 | V |
| $t_{BG}$ | Bandgap reference start-up time | $V_{CC}$=2.7 $T_A$=25°C | - | 40 | 70 | µs |
| $I_{BG}$ | Bandgap reference current consumption | $V_{CC}$=2.7 $T_A$=25°C | - | 10 | - | µA |

**Note:**

1. Values are guidelines only.
2. The Power-on Reset will not work unless the supply voltage has been below VPOT (falling)

**Table 33-9. BODLEVEL Fuse Coding[1][2]**

| BODLEVEL [2:0] Fuses | Min. $V_{BOT}$ | Typ. $V_{BOT}$ | Max $V_{BOT}$ | Units |
|---|---|---|---|---|
| 111 | BOD Disabled | | | |
| 110 | 1.7 | 1.8 | 2.0 | V |
| 101 | 2.5 | 2.7 | 2.9 | |
| 100 | 4.1 | 4.3 | 4.5 | |
| 011 | Reserved | | | |
| 010 | | | | |
| 001 | | | | |
| 000 | | | | |

**Note:** $V_{BOT}$ may be below nominal minimum operating voltage for some devices. For devices where this is the case, the device is tested down to $V_{CC} = V_{BOT}$ during the production test. This guarantees that a Brown-Out Reset will occur before $V_{CC}$ drops to a voltage where correct operation of the microcontroller is no longer guaranteed. The test is performed using BODLEVEL = 110, 101 and 100.

**Note:** $V_{BOT}$ tested at 25°C and 85°C in production

## 33.6. SPI Timing Characteristics

**Table 33-10. SPI Timing Parameters**

| Description | Mode | Min. | Typ | Max | Units |
|---|---|---|---|---|---|
| SCK period | Master | - | See Table. Relationship Between SCK and the Oscillator Frequency in "SPCR – SPI Control Register" | - | ns |
| SCK high/low | Master | - | 50% duty cycle | - | |
| Rise/Fall time | Master | - | 3.6 | - | |
| Setup | Master | - | 10 | - | |
| Hold | Master | - | 10 | - | |
| Out to SCK | Master | - | $0.5 \cdot t_{sck}$ | - | |
| SCK to out | Master | - | 10 | - | |
| SCK to out high | Master | - | 10 | - | |
| SS low to out | Slave | - | 15 | - | |
| SCK period | Slave | $4 \cdot t_{ck}$ | - | - | |
| SCK high/low(1) | Slave | $2 \cdot t_{ck}$ | - | - | |
| Rise/Fall time | Slave | - | - | 1600 | |
| Setup | Slave | 10 | - | - | |
| Hold | Slave | $t_{ck}$ | - | - | |
| SCK to out | Slave | - | 15 | - | |
| SCK to SS high | Slave | 20 | - | - | |
| SS high to tri-state | Slave | | 10 | - | |
| SS low to SCK | Slave | $2 \cdot t_{ck}$ | - | - | |

**Note:** In SPI Programming mode the minimum SCK high/low period is:

- $2 \cdot t_{CLCLCL}$ for $f_{CK} < 12MHz$
- $3 \cdot t_{CLCL}$ for $f_{CK} > 12MHz$

**Figure 33-3. SPI Interface Timing Requirements (Master Mode)**

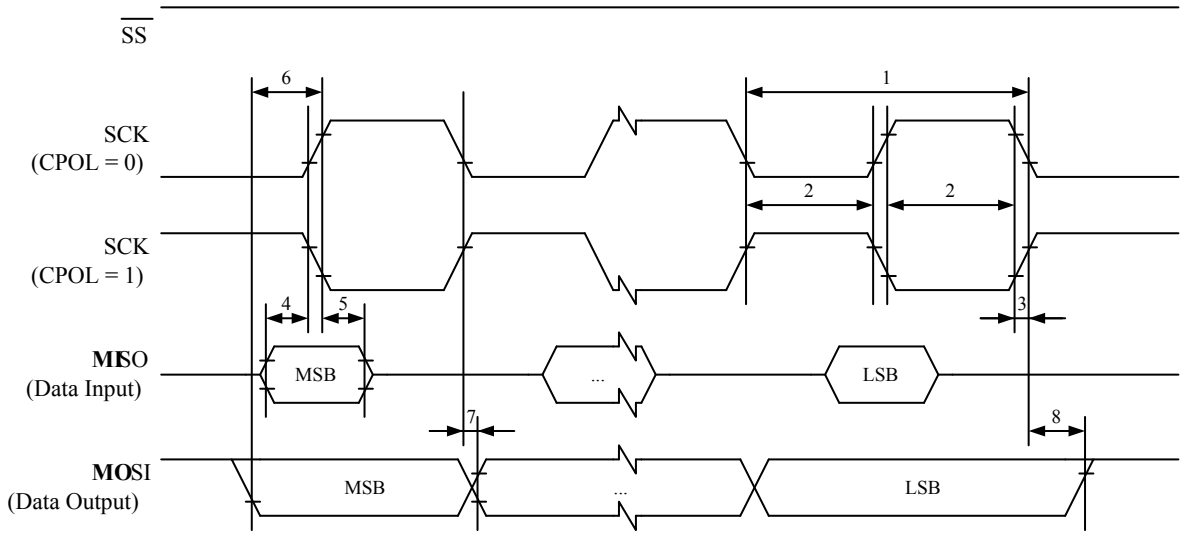
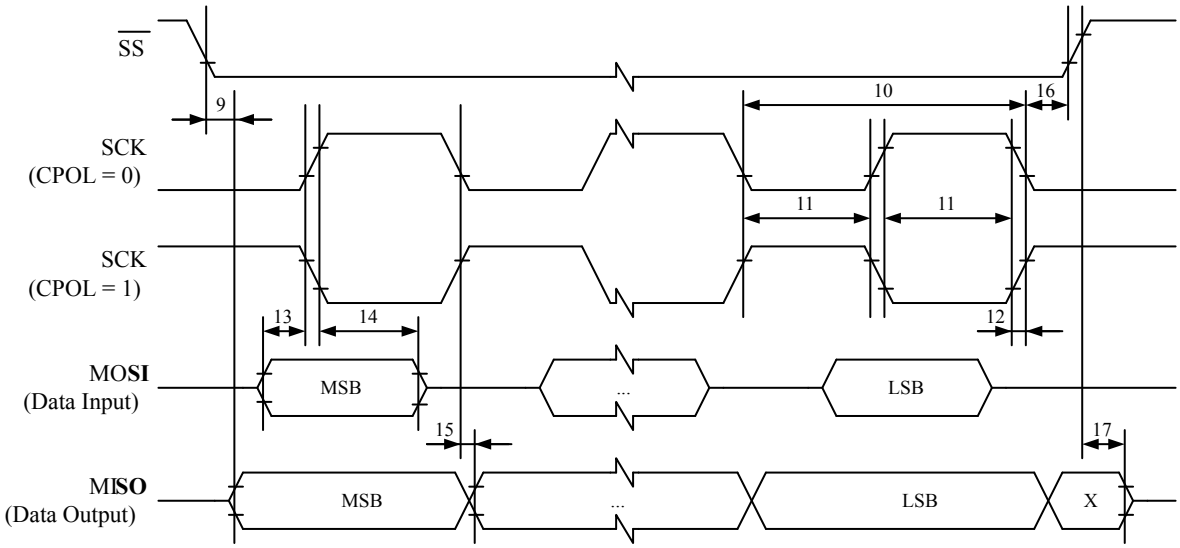



**Figure 33-4. SPI Interface Timing Requirements (Slave Mode)**




## 33.7. Two-wire Serial Interface Characteristics

Table in this section describes the requirements for devices connected to the 2-wire Serial Bus. The 2-wire Serial Interface meets or exceeds these requirements under the noted conditions.

Timing symbols refer to Figure 33-5.

**Table 33-11. Two-wire Serial Bus Requirements**

| Symbol | Parameter | Condition | Min. | Max | Units |
|---|---|---|---|---|---|
| $V_{IL}$ | Input Low-voltage | | -0.5 | 0.3 $V_{CC}$ | V |
| $V_{IH}$ | Input High-voltage | | 0.7 $V_{CC}$ | $V_{CC}$ + 0.5 | V |
| $V_{hys}$(1) | Hysteresis of Schmitt Trigger Inputs | | 0.05 $V_{CC}$(2) | – | V |

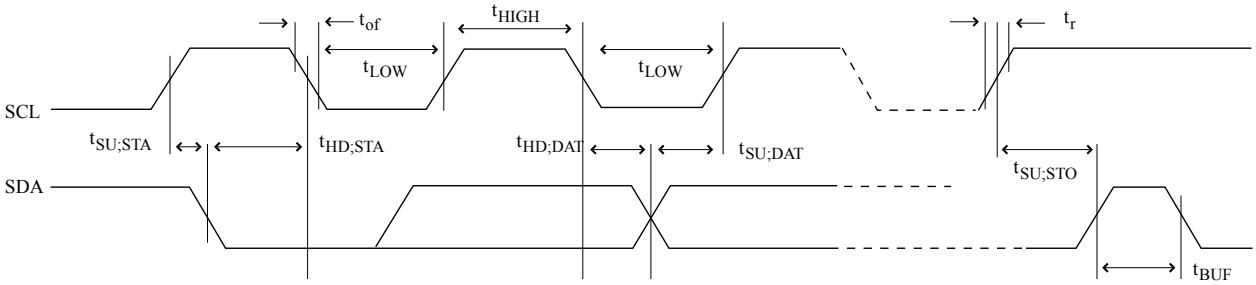
| Symbol | Parameter | Condition | Min. | Max | Units |
|---|---|---|---|---|---|
| $V_{OL}$(1) | Output Low-voltage | 3mA sink current | 0 | 0.4 | V |
| $t_r$(1) | Rise Time for both SDA and SCL | | 20 + 0.1$C_b$(3)(2) | 300 | ns |
| $t_{of}$(1) | Output Fall Time from $V_{IHmin}$ to $V_{ILmax}$ | 10pF < $C_b$ < 400pF(3) | 20 + 0.1$C_b$(3)(2) | 250 | ns |
| $t_{SP}$(1) | Spikes Suppressed by Input Filter | | 0 | 50(2) | ns |
| $I_i$ | Input Current each I/O Pin | 0.1$V_{CC}$ < $V_i$ < 0.9$V_{CC}$ | -10 | 10 | µA |
| $C_i$(1) | Capacitance for each I/O Pin | | – | 10 | pF |
| $f_{SCL}$ | SCL Clock Frequency | $f_{CK}$(4) > max(16$f_{SCL}$, 250kHz)(5) | 0 | 400 | kHz |
| Rp | Value of Pull-up resistor | $f_{SCL}$ ≤ 100kHz | $\frac{V_{CC} - 0.4V}{3mA}$ | $\frac{1000ns}{C_b}$ | Ω |
| | | $f_{SCL}$ > 100kHz | $\frac{V_{CC} - 0.4V}{3mA}$ | $\frac{300ns}{C_b}$ | Ω |
| $t_{HD;STA}$ | Hold Time (repeated) START Condition | $f_{SCL}$ ≤ 100kHz | 4.0 | – | µs |
| | | $f_{SCL}$ > 100kHz | 0.6 | – | µs |
| $t_{LOW}$ | Low Period of the SCL Clock | $f_{SCL}$ ≤ 100kHz | 4.7 | – | µs |
| | | $f_{SCL}$ > 100kHz | 1.3 | – | µs |
| $t_{HIGH}$ | High period of the SCL clock | $f_{SCL}$ ≤ 100kHz | 4.0 | – | µs |
| | | $f_{SCL}$ > 100kHz | 0.6 | – | µs |
| $t_{SU;STA}$ | Set-up time for a repeated START condition | $f_{SCL}$ ≤ 100kHz | 4.7 | – | µs |
| | | $f_{SCL}$ > 100kHz | 0.6 | – | µs |
| $t_{HD;DAT}$ | Data hold time | $f_{SCL}$ ≤ 100kHz | 0 | 3.45 | µs |
| | | $f_{SCL}$ > 100kHz | 0 | 0.9 | µs |
| $t_{SU;DAT}$ | Data setup time | $f_{SCL}$ ≤ 100kHz | 250 | – | ns |
| | | $f_{SCL}$ > 100kHz | 100 | – | ns |
| $t_{SU;STO}$ | Setup time for STOP condition | $f_{SCL}$ ≤ 100kHz | 4.0 | – | µs |
| | | $f_{SCL}$ > 100kHz | 0.6 | – | µs |
| $t_{BUF}$ | Bus free time between a STOP and START condition | $f_{SCL}$ ≤ 100kHz | 4.7 | – | µs |
| | | $f_{SCL}$ > 100kHz | 1.3 | – | µs |

**Note:**

1. This parameter is characterized and not 100% tested.
2. Required only for $f_{SCL}$ > 100kHz.
3. $C_b$ = capacitance of one bus line in pF.

4. $f_{CK}$ = CPU clock frequency.
5. This requirement applies to all 2-wire Serial Interface operation. Other devices connected to the 2-wire Serial Bus need only obey the general $f_{SCL}$ requirement.

**Figure 33-5. Two-wire Serial Bus Timing**




## 33.8. ADC Characteristics

**Table 33-12. ADC Characteristics**

| Symbol | Parameter | Condition | Min. | Typ | Max | Units |
|---|---|---|---|---|---|---|
| | Resolution | | - | 10 | - | Bits |
| | Absolute accuracy (Including INL, DNL, quantization error, gain and offset error) | $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200kHz | - | 2 | - | LSB |
| | | $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 1MHz | - | 4 | - | LSB |
| | | $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200kHz Noise Reduction Mode | - | 2 | - | LSB |
| | | $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 1MHz Noise Reduction Mode | - | 4 | - | LSB |
| | Integral Non-Linearity (INL) | $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200kHz | - | 0.5 | - | LSB |
| | Differential Non-Linearity (DNL) | $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200kHz | - | 0.25 | - | LSB |
| | Gain Error | $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200kHz | - | 2 | - | LSB |
| | Offset Error | $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200kHz | - | 2 | - | LSB |
| | Conversion Time | Free Running Conversion | 13 | - | 260 | µs |
| | Clock Frequency | | 50 | - | 1000 | kHz |
| $AV_{CC}$(1) | Analog Supply Voltage | | $V_{CC}$ - 0.3 | - | $V_{CC}$ + 0.3 | V |
| VREF | Reference Voltage | | 1.0 | - | $AV_{CC}$ | V |
| $V_{IN}$ | Input Voltage | | GND | - | $V_{REF}$ | V |

| Symbol | Parameter | Condition | Min. | Typ | Max | Units |
|---|---|---|---|---|---|---|
| | Input Bandwidth | | - | 38.5 | | kHz |
| $V_{INT}$ | Internal Voltage Reference | | 1.0 | 1.1 | 1.2 | V |
| $R_{REF}$ | Reference Input Resistance | | - | 50 | - | kΩ |
| $R_{AIN}$ | Analog Input Resistance | | - | 100 | - | MΩ |

**Note:**

1. $AV_{CC}$ absolute min./max: 1.8V/5.5V

## 33.9. Parallel Programming Characteristics

**Table 33-13. Parallel Programming Characteristics, $V_{CC}$ = 5V ± 10%**

| Symbol | Parameter | Min. | 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 XTAL1 High | 67 | - | ns |
| $t_{XLXH}$ | XTAL1 Low to XTAL1 High | 200 | - | ns |
| $t_{XHXL}$ | XTAL1 Pulse Width High | 150 | - | ns |
| $t_{XLDX}$ | Data and Control Hold after XTAL1 Low | 67 | - | ns |
| $t_{XLWL}$ | XTAL1 Low to $\overline{WR}$ Low | 0 | - | ns |
| $t_{XLPH}$ | XTAL1 Low to PAGEL high | 0 | - | ns |
| $t_{PLXH}$ | PAGEL low to XTAL1 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 RDY/$\overline{BSY}$ high | 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) | 3.2 | 3.4 | ms |
| $t_{WLRH_CE}$ | $\overline{WR}$ Low to RDY/$\overline{BSY}$ High for Chip Erase(2) | 9.8 | 10.5 | ms |
| $t_{XLOL}$ | XTAL1 Low to $\overline{OE}$ Low | 0 | - | ns |
| $t_{BVDV}$ | BS1 Valid to DATA valid | 0 | 350 | ns |

| Symbol | Parameter | Min. | Max | Units |
|---|---|---|---|---|
| $t_{OLDV}$ | $\overline{OE}$ Low to DATA Valid | - | 350 | ns |
| $t_{OHDZ}$ | $\overline{OE}$ High to DATA Tri-stated | - | 250 | ns |

**Note:**

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.

**Figure 33-6. Parallel Programming Timing, Including some General Timing Requirements**

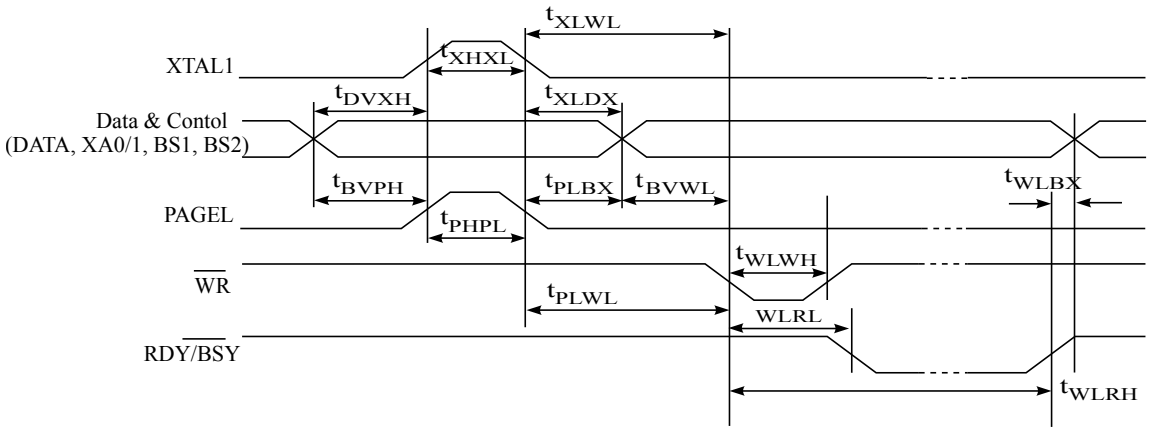
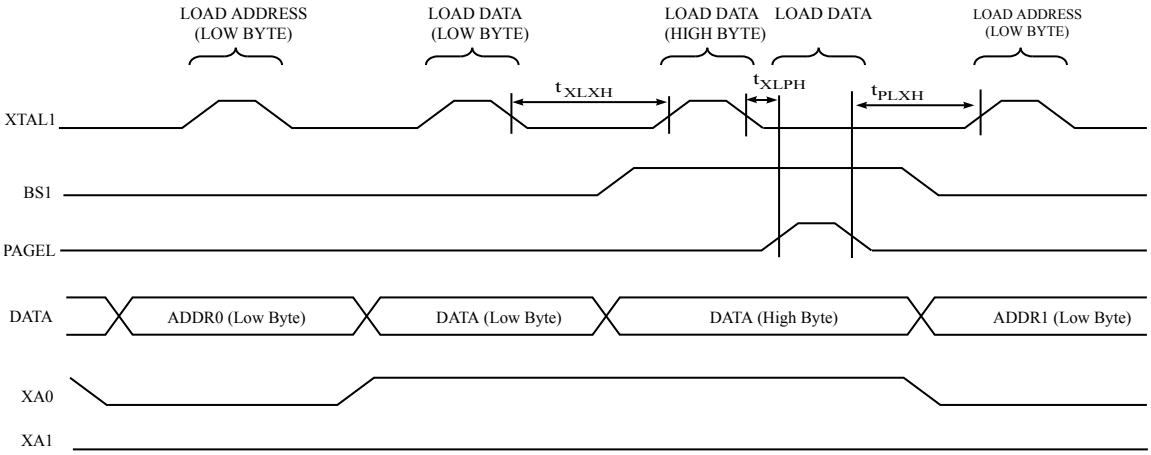
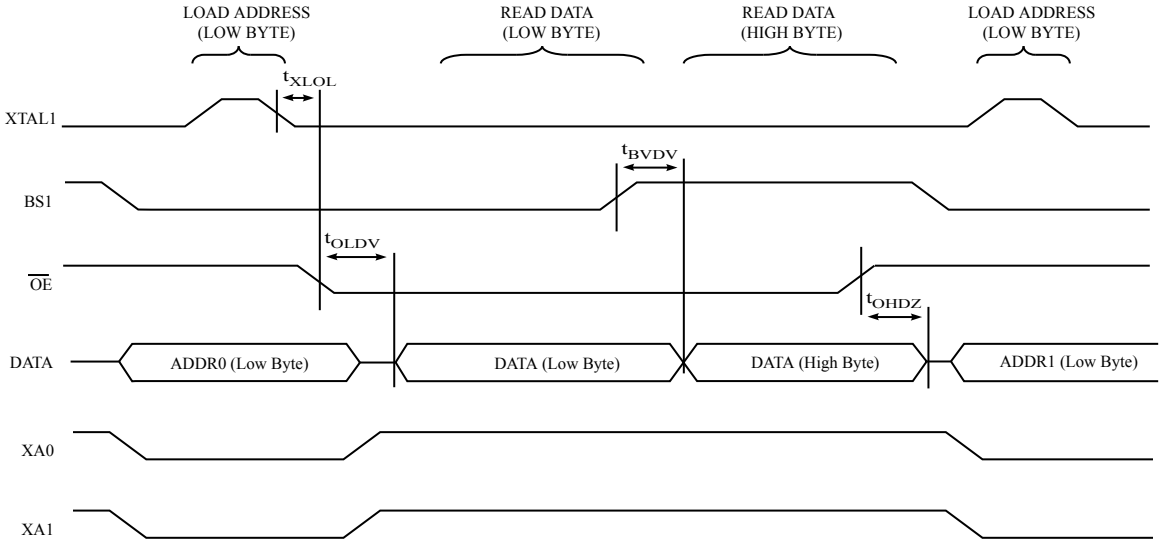



**Figure 33-7. Parallel Programming Timing, Loading Sequence with Timing Requirements**




**Note:** The timing requirements shown in Parallel Programming Characteristics (i.e., $t_{DVXH}$, $t_{XHXL}$, and $t_{XLDX}$) also apply to loading operation

**Figure 33-8. Parallel Programming Timing, Reading Sequence (within the Same Page) with Timing Requirements**




**Note:** The timing requirements shown in Parallel Programming Characteristics (i.e., $t_{DVXH}$, $t_{XHXL}$, and $t_{XLDX}$) also apply to reading operation.

# 34. Typical Characteristics ($T_A$ = -40°C to 85°C)

The following charts show typical behavior. These figures are not tested during manufacturing. All current consumption measurements are performed with all I/O pins configured as inputs and with internal pull-ups enabled. A sine wave generator with rail-to-rail output is used as clock source.

The power consumption in Power-down mode is independent of clock selection.

The 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.

The current drawn from capacitive loaded pins may be estimated (for one pin) as $C_L$*$V_{CC}$*f where $C_L$ = load capacitance, $V_{CC}$ = operating voltage and f = average switching frequency of I/O pin.

The parts are characterized at frequencies higher than test limits. Parts are not guaranteed to function properly at frequencies higher than the ordering code indicates.

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.

## 34.1. ATmega48A Typical Characteristics

### 34.1.1. Active Supply Current

**Figure 34-1. ATmega48A: Active Supply Current vs. Low Frequency (0.1MHz - 1.0MHz)**

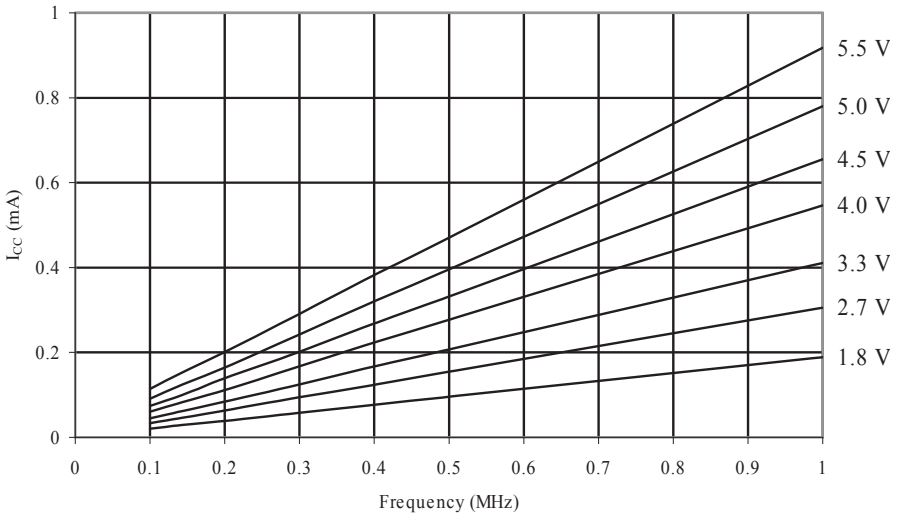


**Figure 34-2. ATmega48A: Active Supply Current vs. Frequency (1MHz - 20MHz)**

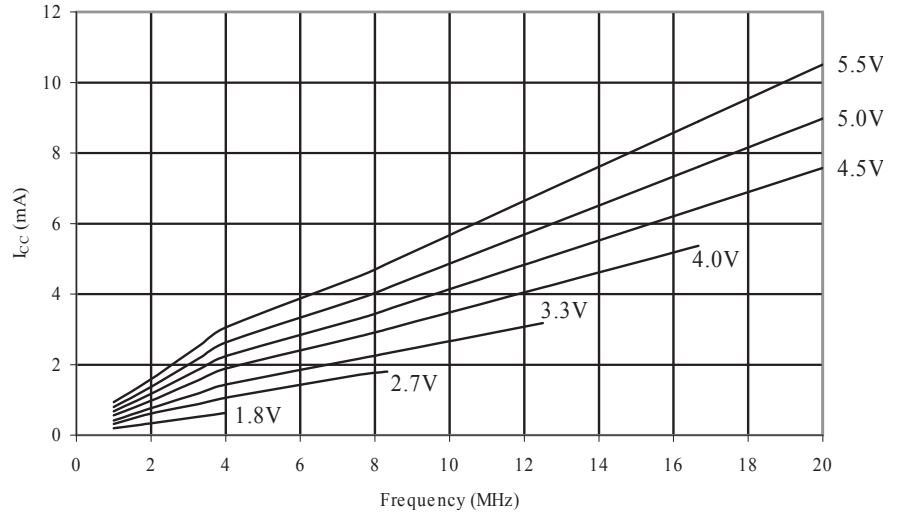



**Figure 34-3. ATmega48A: Active Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 128kHz)**

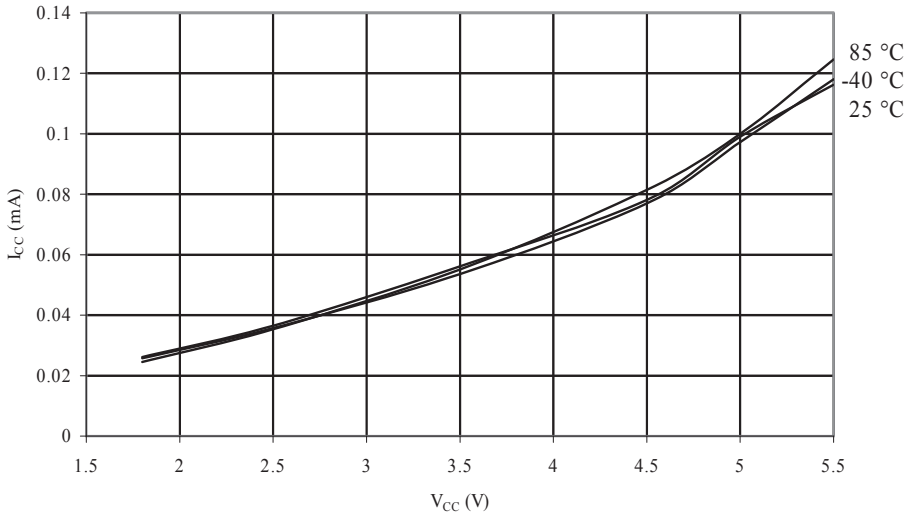



**Figure 34-4. ATmega48A: Active Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 1MHz)**

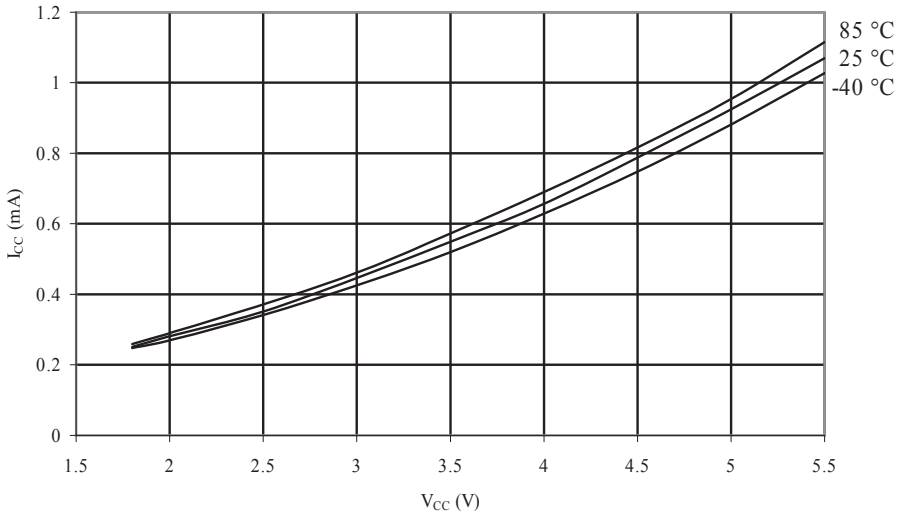
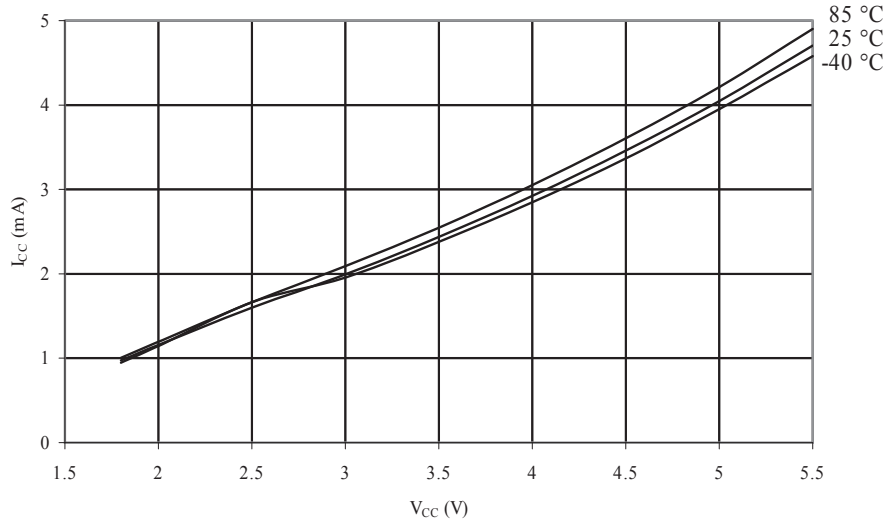


**Figure 34-5. ATmega48A: Active Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 8MHz)**

cc




### 34.1.2. Idle Supply Current

**Figure 34-6. ATmega48A: Idle Supply Current vs. Low Frequency (0.1MHz - 1.0MHz)**

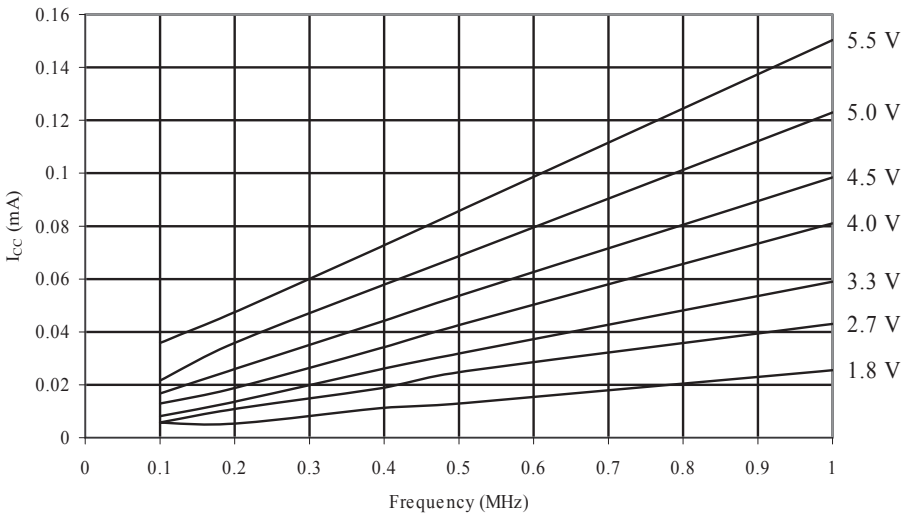


**Figure 34-7. ATmega48A: Idle Supply Current vs. Frequency (1MHz - 20MHz)**

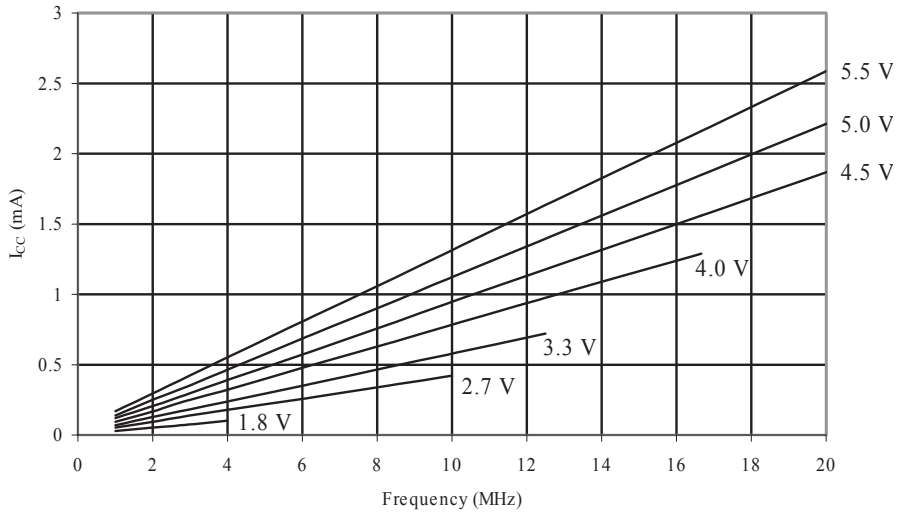



**Figure 34-8. ATmega48A: Idle Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 128kHz)**

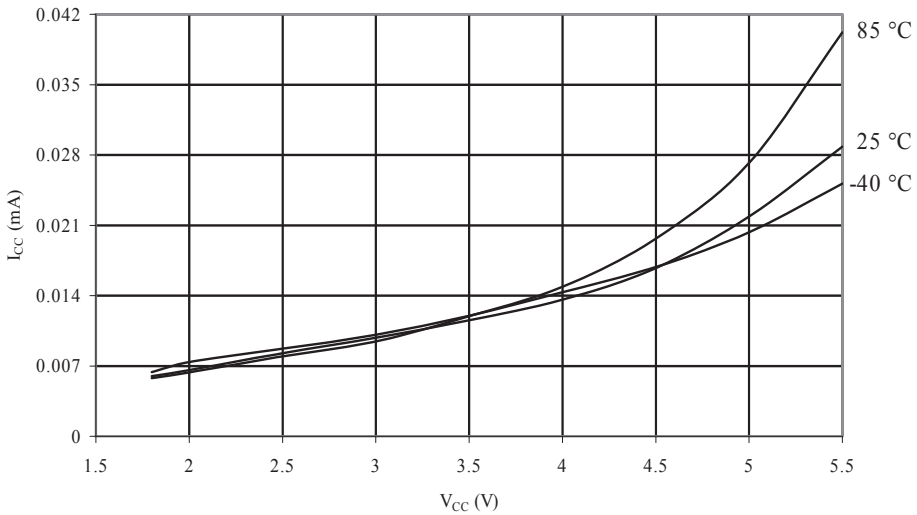



**Figure 34-9. ATmega48A: Idle Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 1MHz)**

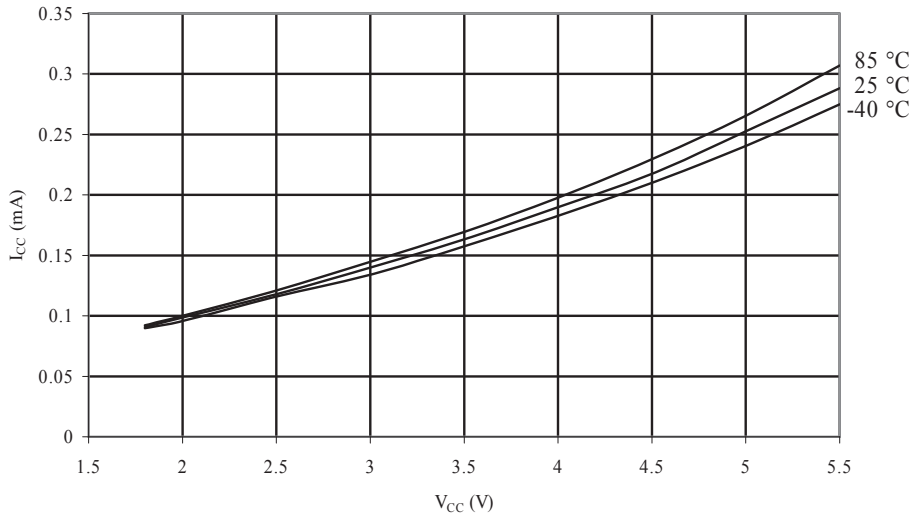
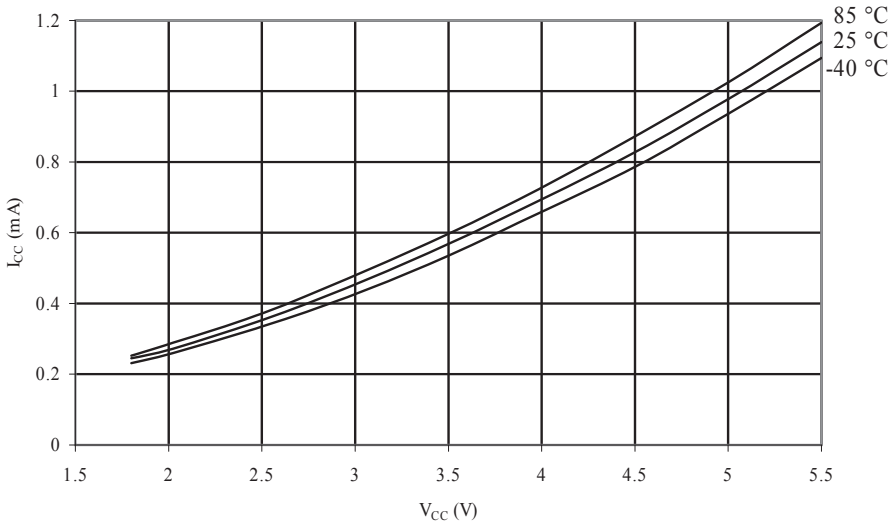


**Figure 34-10. ATmega48A: Idle Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 8MHz)**




### 34.1.3. Supply Current of IO Modules

The tables and formulas below can be used to calculate the additional current consumption for the different I/O modules in Active and Idle mode. The enabling or disabling of the I/O modules are controlled by the Power Reduction Register. See “Power Reduction Register” for details.

**Table 34-1. ATmega48A: Additional Current Consumption for the different I/O modules (absolute values)**

| PRR bit | Typical numbers (µA) | | |
|---|---|---|---|
| | $V_{CC}$ = 2V, F = 1MHz | $V_{CC}$ = 3V, F = 4MHz | $V_{CC}$ = 5V, F = 8MHz |
| PRUSART0 | 2.9 | 20.7 | 97.4 |
| PRTWI | 6.0 | 44.8 | 219.7 |
| PRTIM2 | 5.0 | 34.5 | 141.3 |
| PRTIM1 | 3.6 | 24.4 | 107.7 |
| PRTIM0 | 1.4 | 9.5 | 38.4 |
| PRSPI | 5.0 | 38.0 | 190.4 |
| PRADC | 6.1 | 47.4 | 244.7 |

**Table 34-2. ATmega48A: Additional Current Consumption (percentage) in Active and Idle mode**

| PRR bit | Additional Current consumption compared to Active with external clock (see Figure 34-1 and Figure 34-2) | Additional Current consumption compared to Idle with external clock (see Figure 34-6 and Figure 34-7) |
|---|---|---|
| PRUSART0 | 1.8% | 11.4% |
| PRTWI | 3.9% | 20.6% |
| PRTIM2 | 2.9% | 15.7% |
| PRTIM1 | 2.1% | 11.2% |
| PRTIM0 | 0.8% | 4.2% |

| PRR bit | Additional Current consumption compared to Active with external clock (see Figure 34-1 and Figure 34-2) | Additional Current consumption compared to Idle with external clock (see Figure 34-6 and Figure 34-7) |
|---|---|---|
| PRSPI | 3.3% | 17.6% |
| PRADC | 4.2% | 22.1% |

It is possible to calculate the typical current consumption based on the numbers from the above table for other $V_{CC}$ and frequency settings.

**Related Links**

PRR on page 73

#### 34.1.3.1. Example

Calculate the expected current consumption in idle mode with TIMER1, ADC, and SPI enabled at $V_{CC}$ = 2.0V and F = 1MHz. From Table *Additional Current Consumption (percentage) in Active and Idle mode* in the previous section, third column, we see that we need to add 11.2% for the TIMER1, 22.1% for the ADC, and 17.6% for the SPI module. Reading from Figure *Idle Supply Current vs. Low Frequency (0.1-1.0MHz)*, we find that the idle current consumption is ~0.028mA at $V_{CC}$ = 2.0V and F = 1MHz. The total current consumption in idle mode with TIMER1, ADC, and SPI enabled, gives:

$$I_{CCtotal} \simeq 0.028\ \text{mA} \cdot (1 + 0.112 + 0.221 + 0.176) \simeq 0.042\ \text{mA}$$

### 34.1.4. Power-down Supply Current

**Figure 34-11. ATmega48A: Power-Down Supply Current vs. $V_{CC}$ (Watchdog Timer Disabled)**

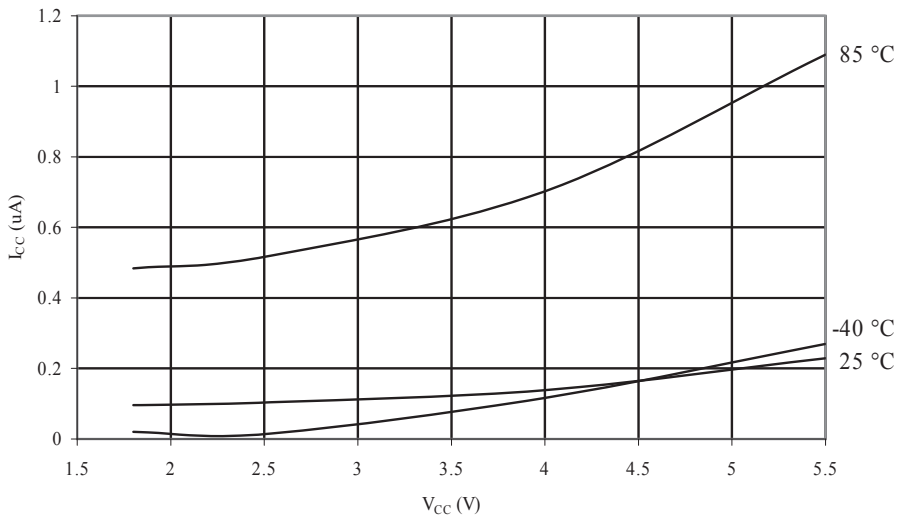
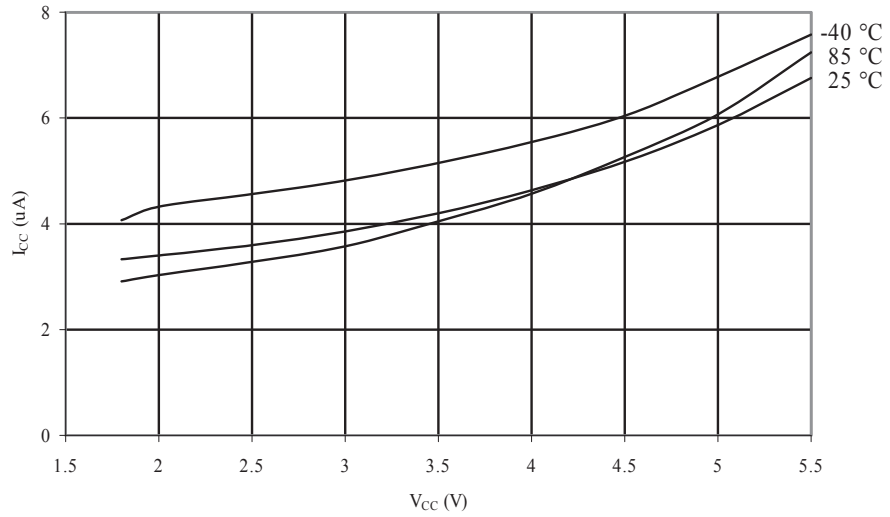
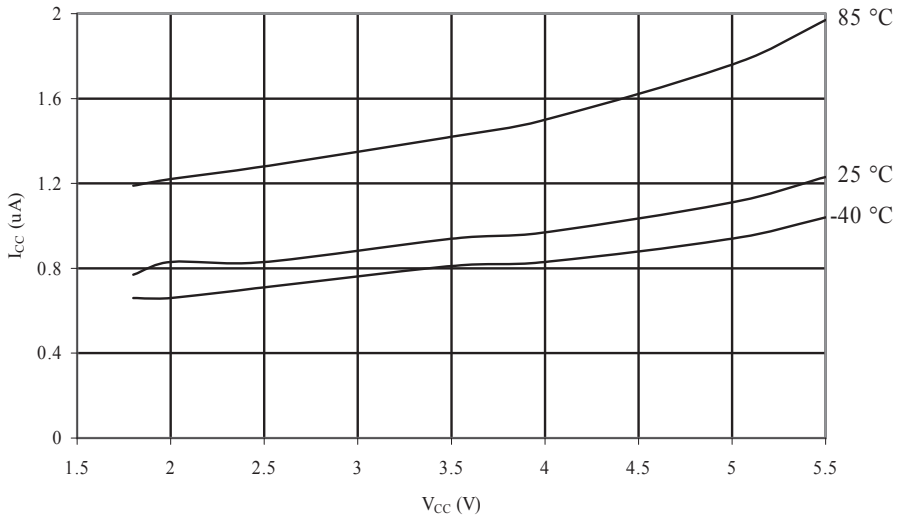


**Figure 34-12. ATmega48A: Power-Down Supply Current vs. $V_{CC}$ (Watchdog Timer Enabled)**



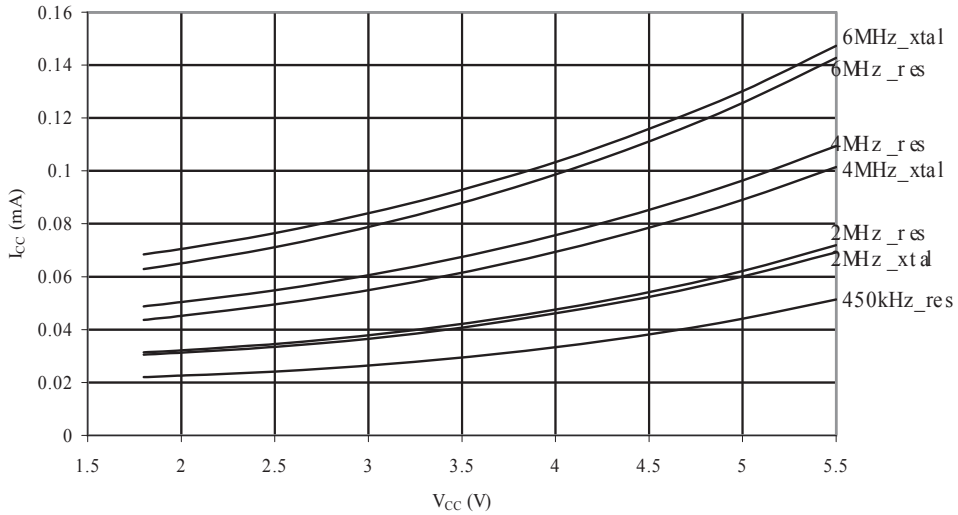

### 34.1.5. Power-save Supply Current

**Figure 34-13. Power-Save Supply Current vs. $V_{CC}$ (Watchdog Timer Disabled and 32kHz Crystal Oscillator Running)**



### 34.1.6. Standby Supply Current

**Figure 34-14. ATmega48A: Standby Supply Current vs. $V_{cc}$ (Watchdog Timer Disabled)**




### 34.1.7. Pin Pull-Up

**Figure 34-15. ATmega48A: I/O Pin Pull-up Resistor Current vs. Input Voltage ($V_{CC}$ = 1.8V)**

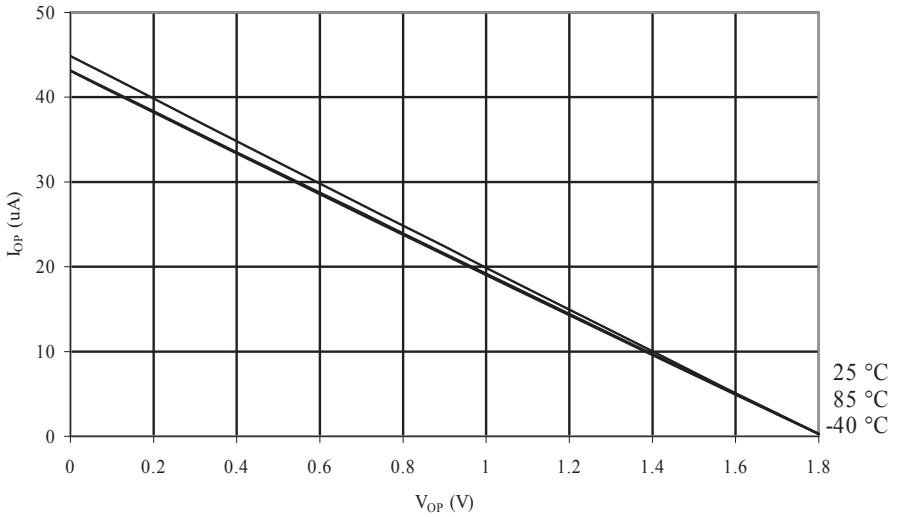


**Figure 34-16. ATmega48A: I/O Pin Pull-up Resistor Current vs. Input Voltage ($V_{CC}$ = 2.7V)**

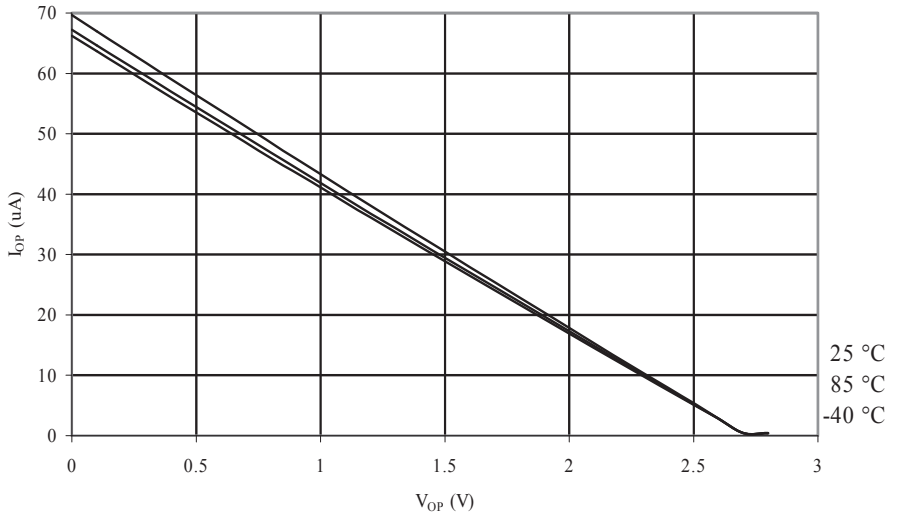



**Figure 34-17. ATmega48A: I/O Pin Pull-up Resistor Current vs. Input Voltage ($V_{CC}$ = 5V)**

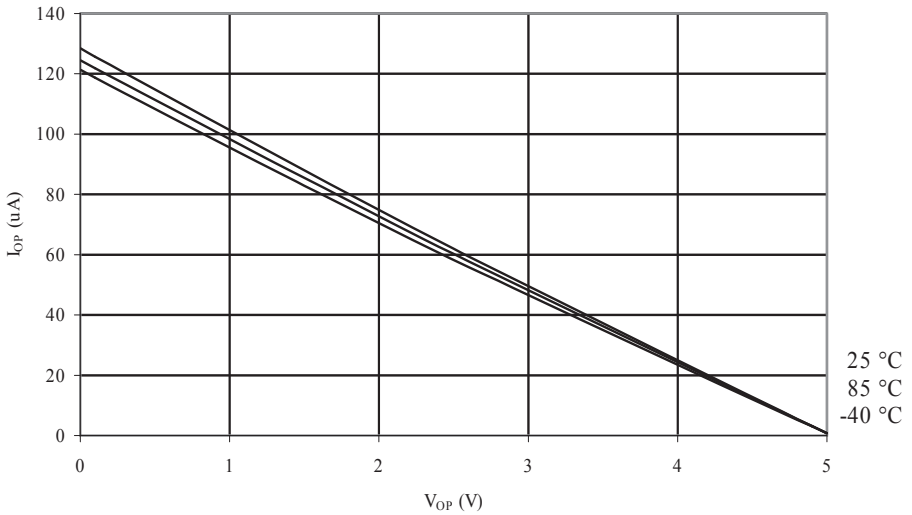



**Figure 34-18. ATmega48A: Reset Pull-up Resistor Current vs. Reset Pin Voltage ($V_{CC}$ = 1.8V)**

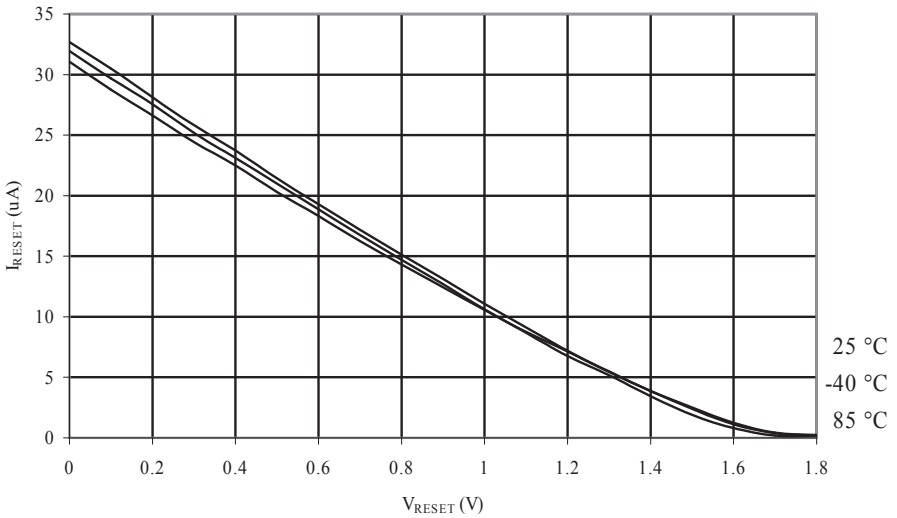


**Figure 34-19. ATmega48A: Reset Pull-up Resistor Current vs. Reset Pin Voltage ($V_{CC}$ = 2.7V)**

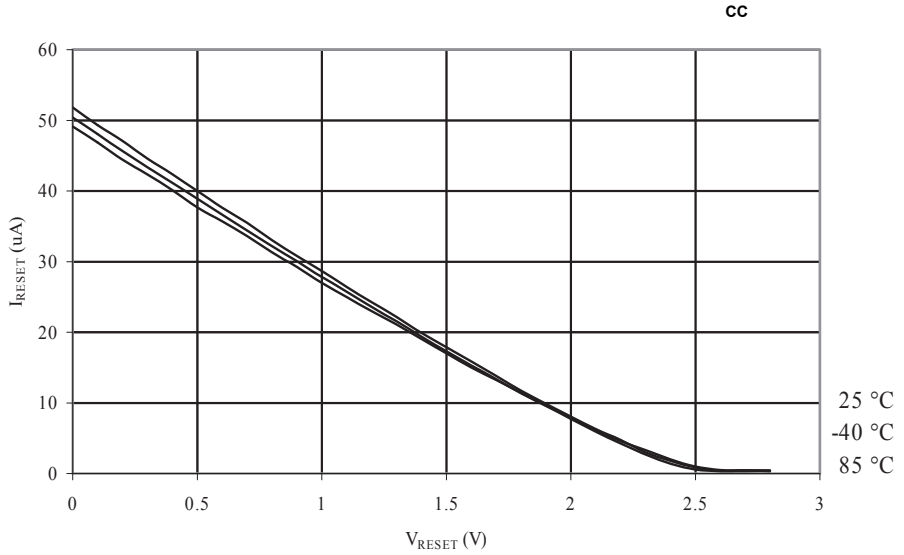
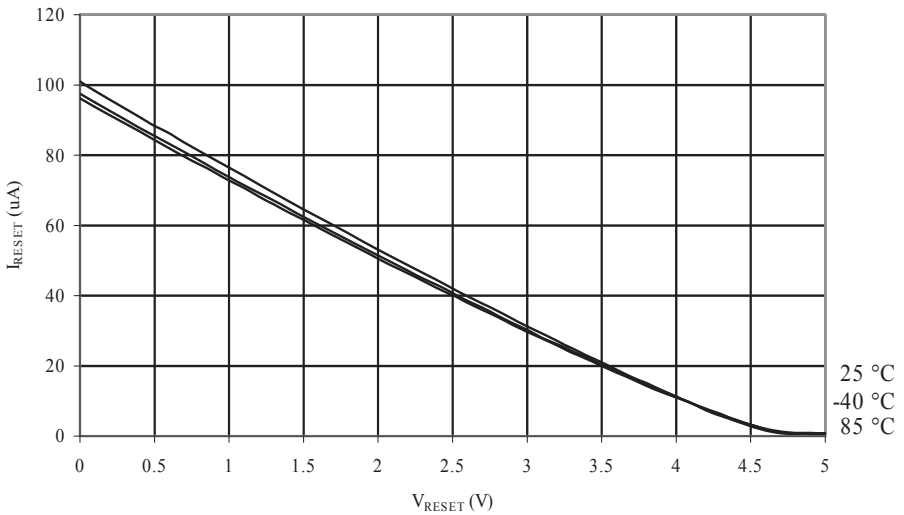



**Figure 34-20. ATmega48A: Reset Pull-up Resistor Current vs. Reset Pin Voltage ($V_{CC}$ = 5V)**



### 34.1.8. Pin Driver Strength

**Figure 34-21. ATmega48A: I/O Pin Output Voltage vs. Sink Current ($V_{CC}$ = 3V)**

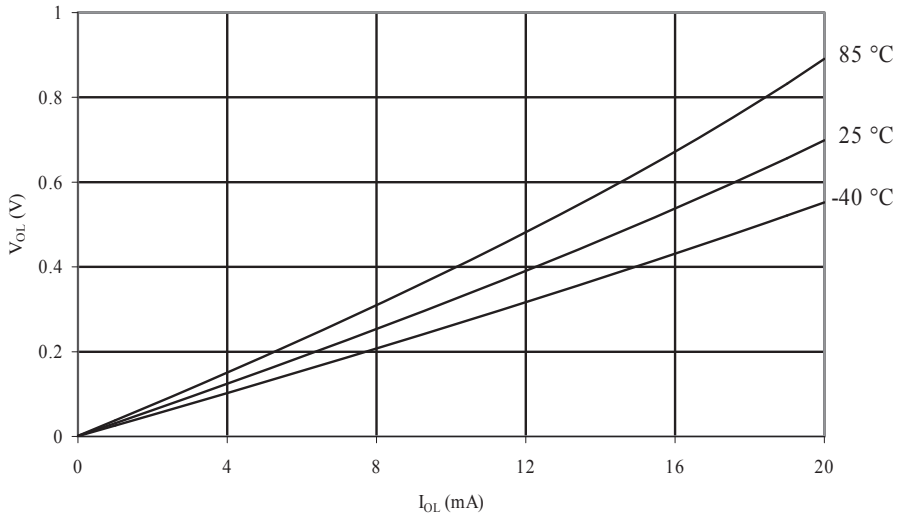



**Figure 34-22. ATmega48A: I/O Pin Output Voltage vs. Sink Current ($V_{CC}$ = 5V)**

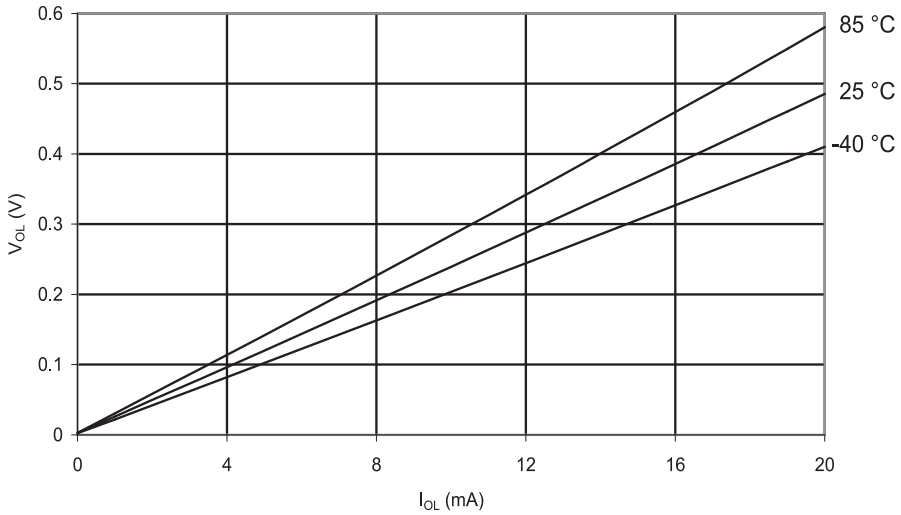


**Figure 34-23. ATmega48A: I/O Pin Output Voltage vs. Source Current ($V_{cc}$ = 3V)**

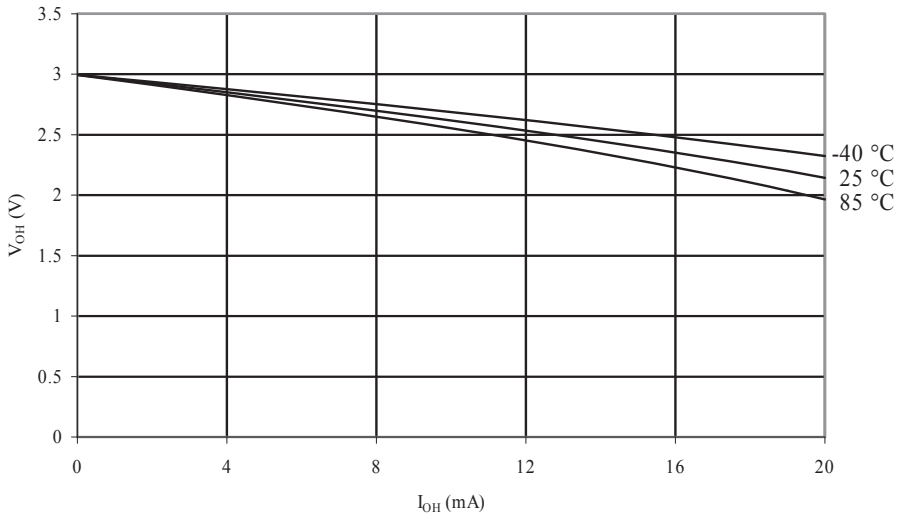
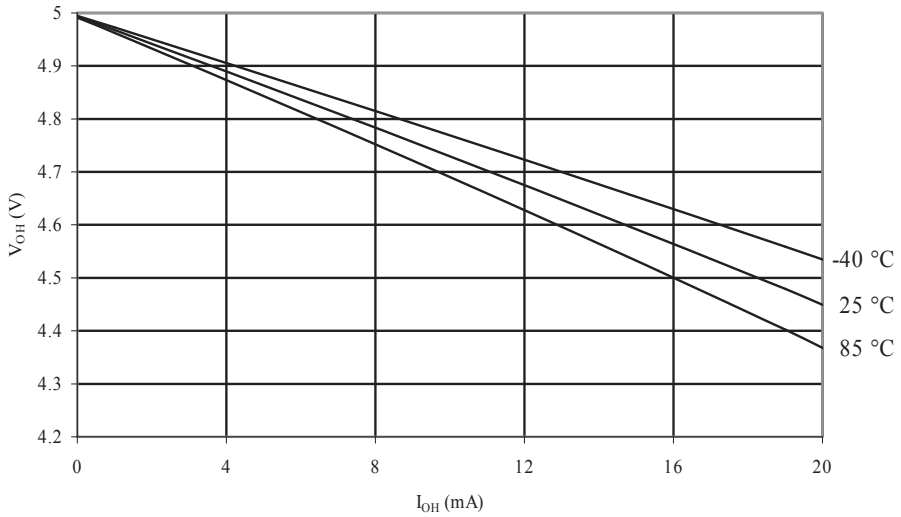



**Figure 34-24. ATmega48A: I/O Pin Output Voltage vs. Source Current ($V_{CC}$ = 5V)**



### 34.1.9. Pin Threshold and Hysteresis

**Figure 34-25. ATmega48A: I/O Pin Input Threshold Voltage vs. $V_{CC}$ ($V_{IH}$, I/O Pin read as '1')**

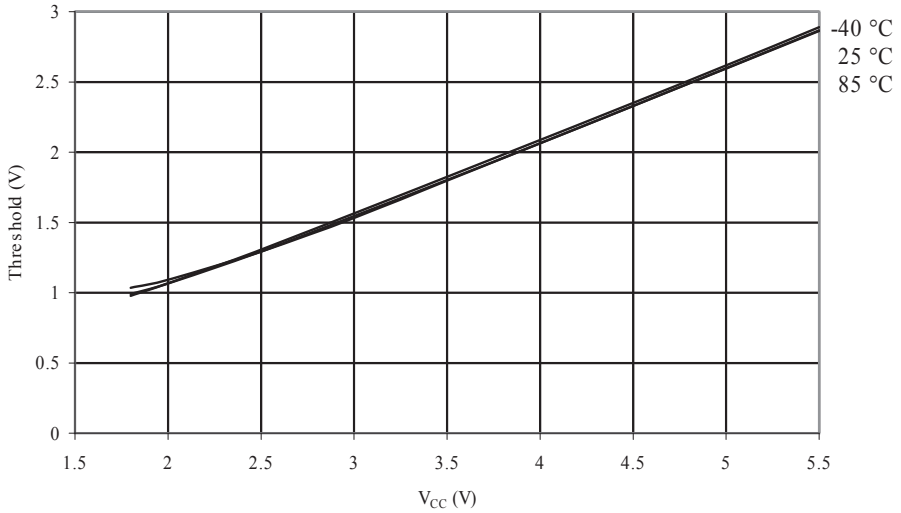



**Figure 34-26. ATmega48A: I/O Pin Input Threshold Voltage vs. $V_{CC}$ ($V_{IL}$, I/O Pin read as '0'**

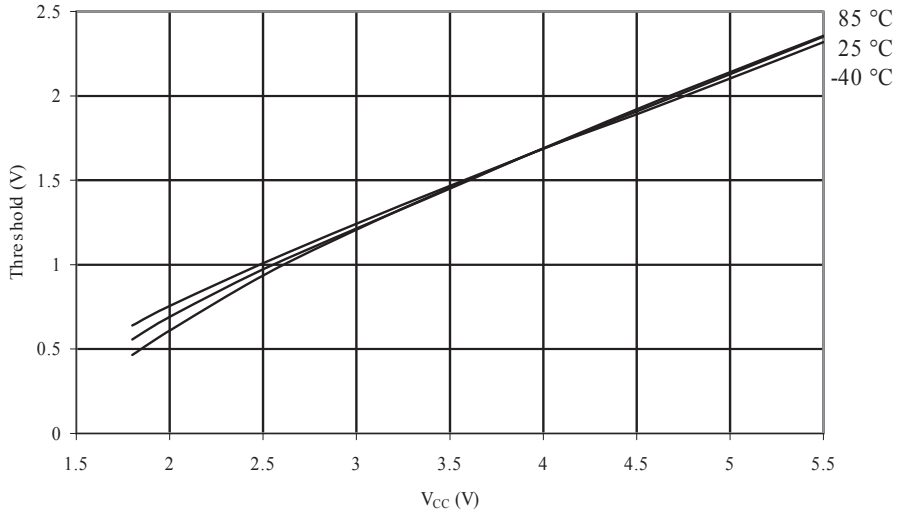



**Figure 34-27. ATmega48A: I/O Pin Input Hysteresis vs. $V_{CC}$**

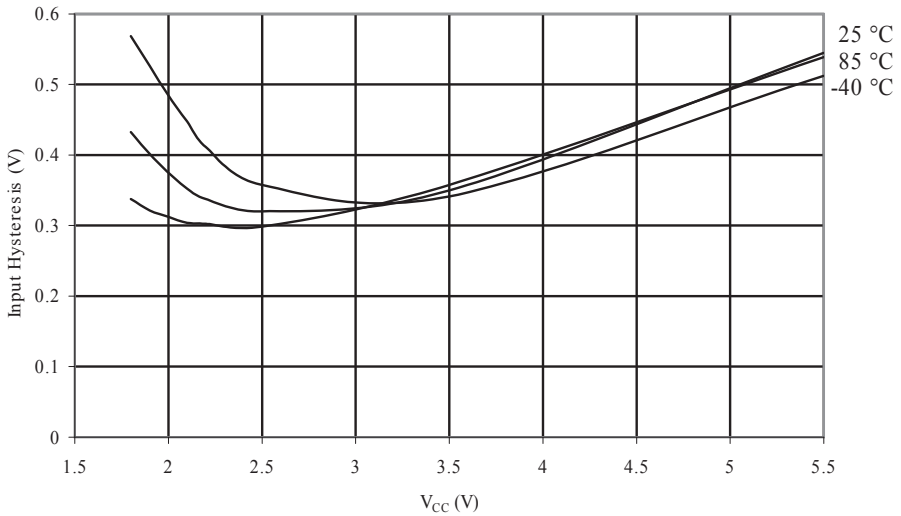


**Figure 34-28. ATmega48A: Reset Input Threshold Voltage vs. $V_{CC}$ ($V_{IH}$, I/O Pin read as '1')**

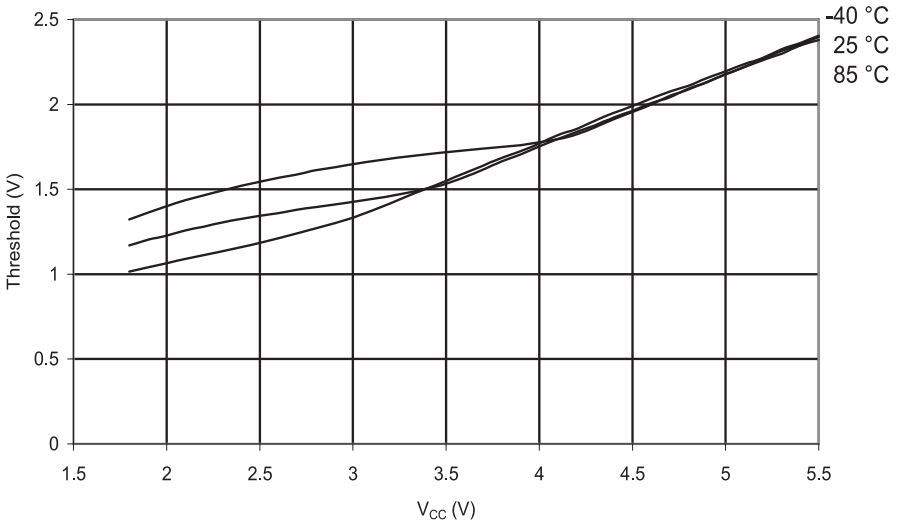



**Figure 34-29. ATmega48A: Reset Input Threshold Voltage vs. $V_{CC}$ ($V_{IL}$, I/O Pin read as '0')**

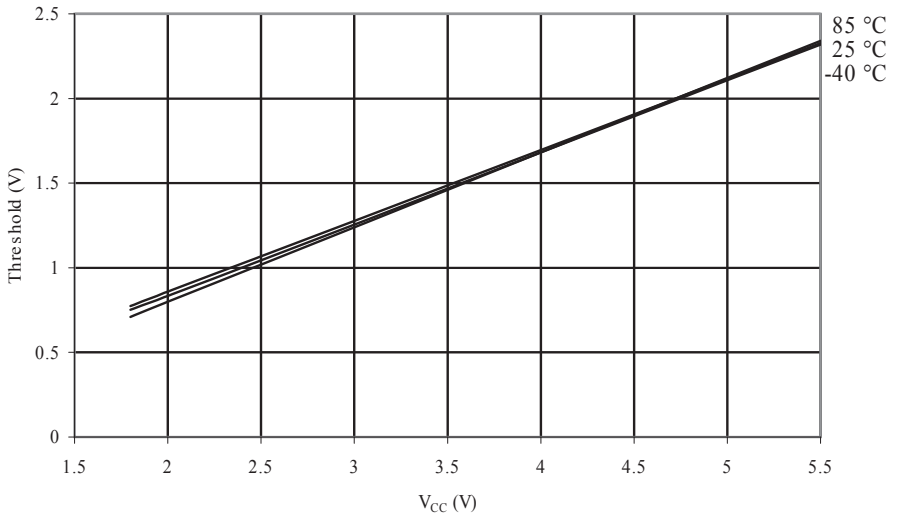
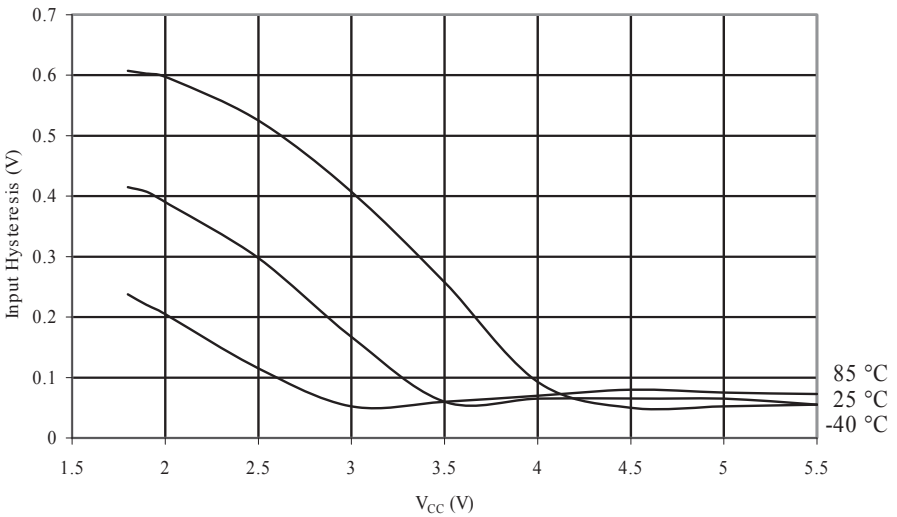



**Figure 34-30. ATmega48A: Reset Pin Input Hysteresis vs. $V_{CC}$**



### 34.1.10. BOD Threshold

**Figure 34-31. ATmega48A: BOD Thresholds vs. Temperature (BODLEVEL is 1.8V)**

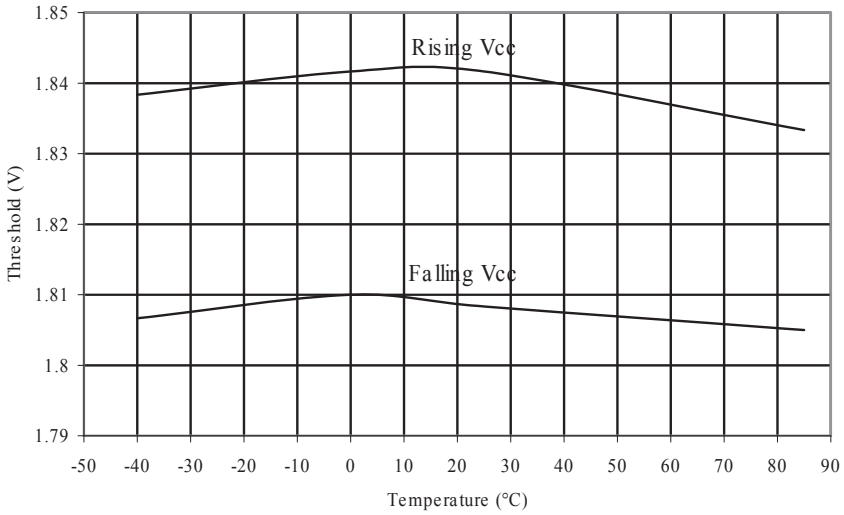



**Figure 34-32. ATmega48A: BOD Thresholds vs. Temperature (BODLEVEL is 2.7V)**

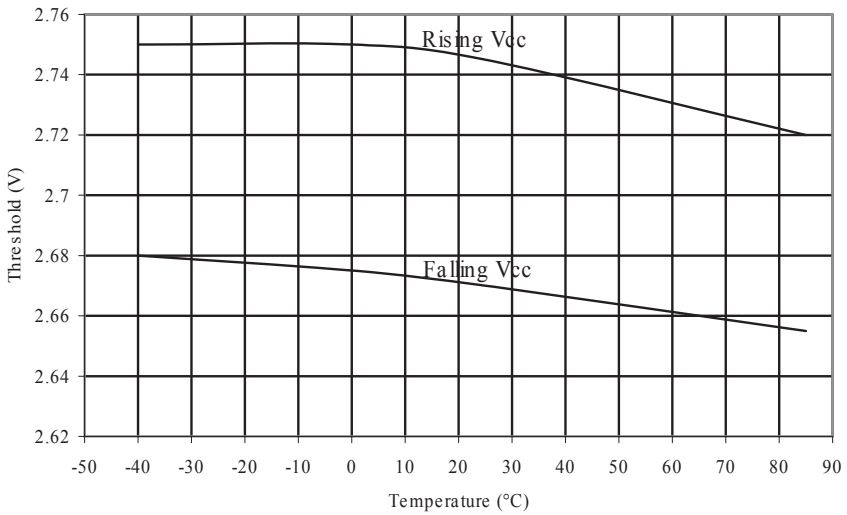


**Figure 34-33. ATmega48A: BOD Thresholds vs. Temperature (BODLEVEL is 4.3V)**

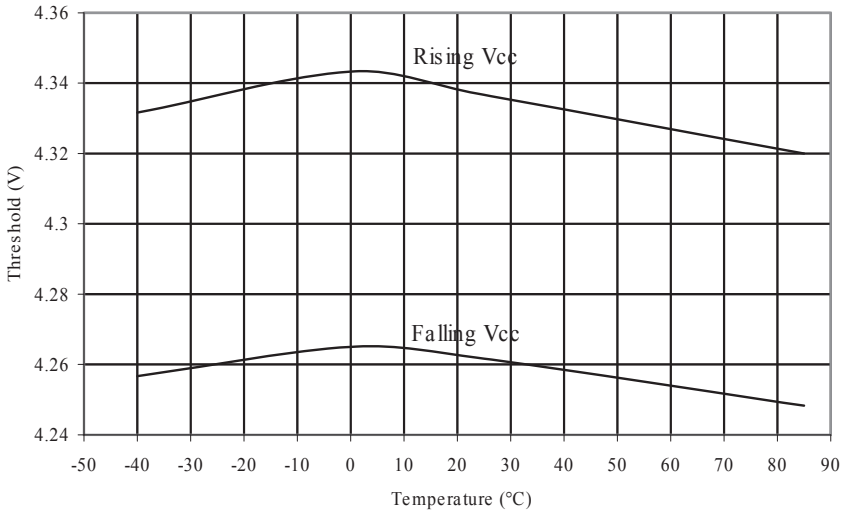
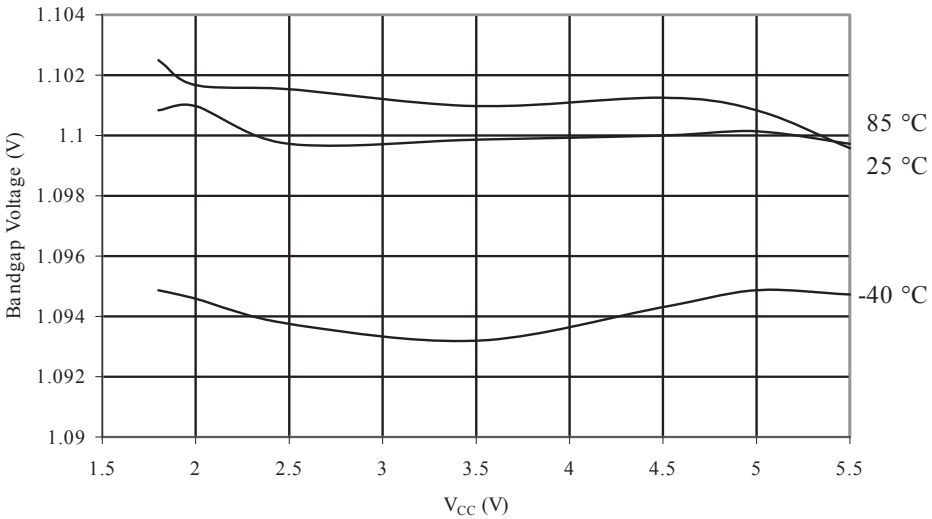



**Figure 34-34. ATmega48A:Bandgap Voltage vs. $V_{CC}$**




## 34.1.11. Internal Oscillator Speed

**Figure 34-35. ATmega48A: Watchdog Oscillator Frequency vs. Temperature**

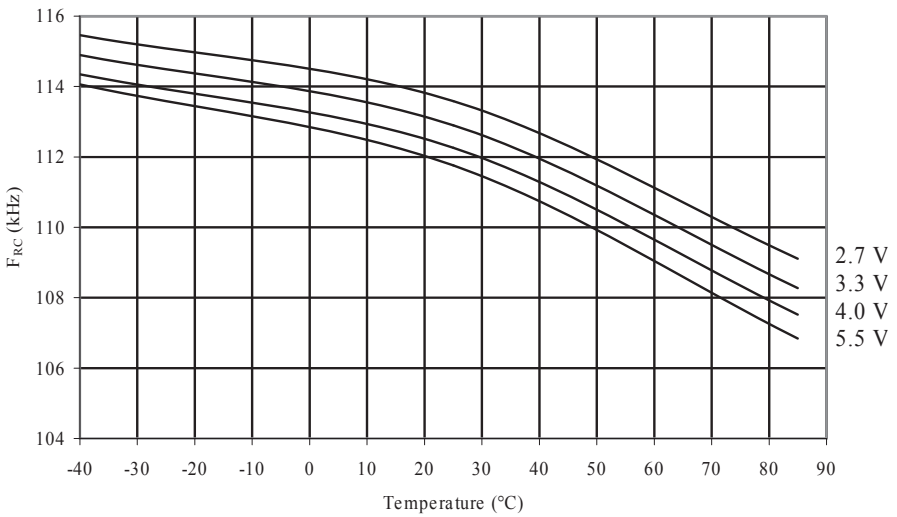


**Figure 34-36. ATmega48A: Watchdog Oscillator Frequency vs. $V_{CC}$**

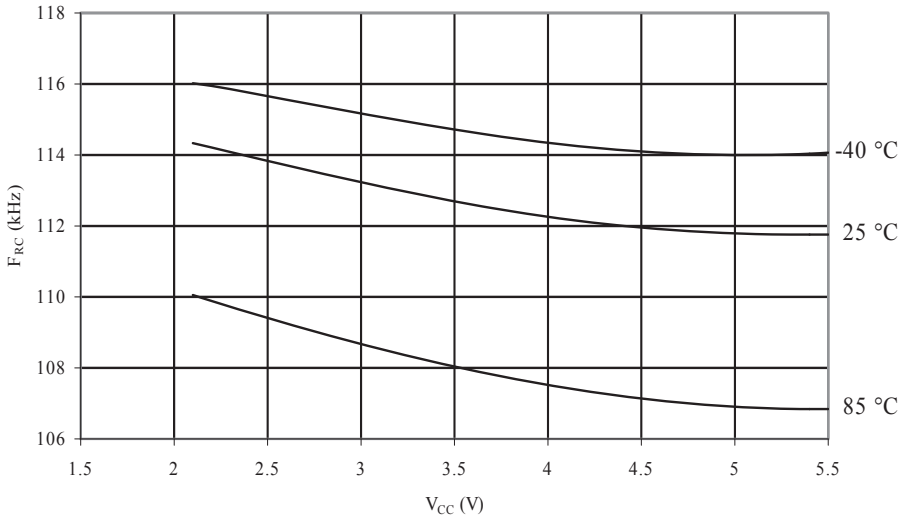



**Figure 34-37. ATmega48A: Calibrated 8 MHz RC Oscillator Frequency vs. $V_{CC}$**

cc

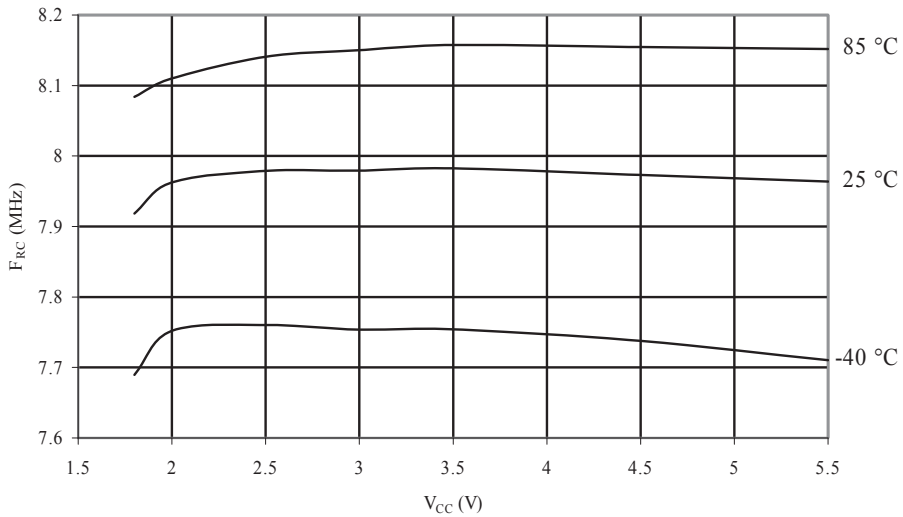



**Figure 34-38. ATmega48A: Calibrated 8MHz RC Oscillator Frequency vs. Temperature**

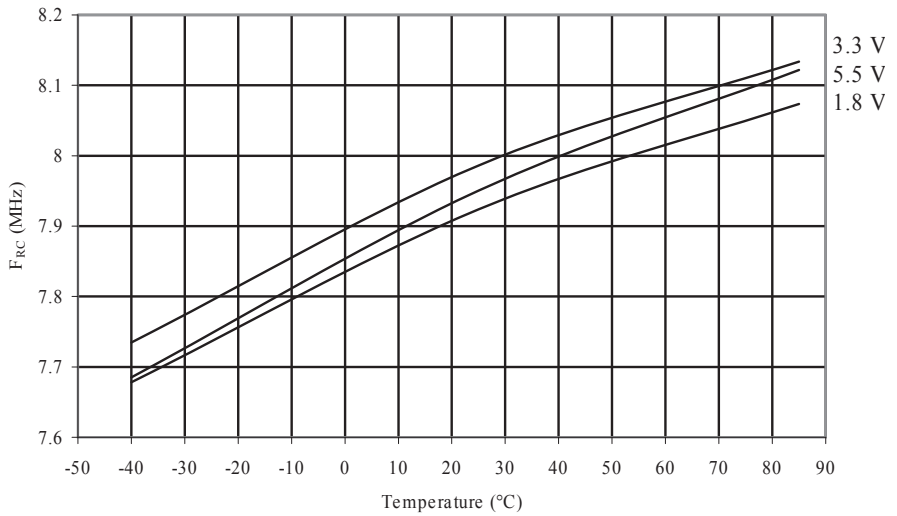
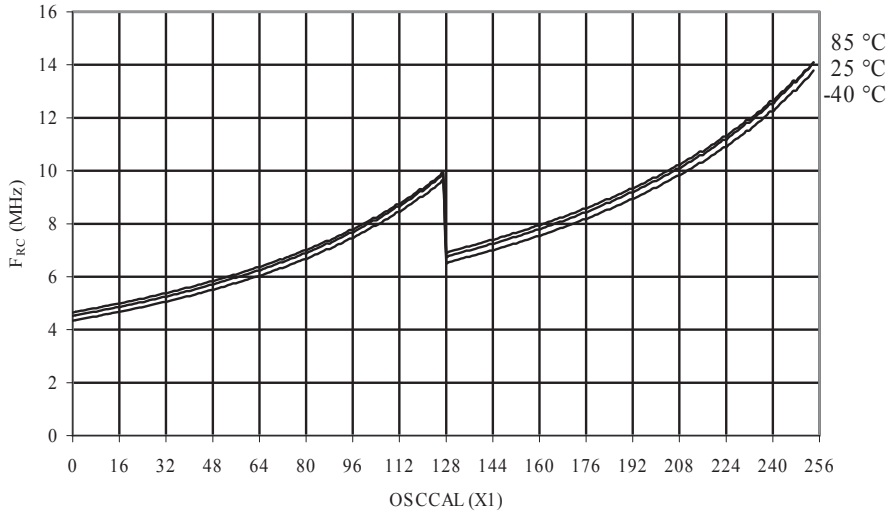


**Figure 34-39. ATmega48A: Calibrated 8MHz RC Oscillator Frequency vs. OSCCAL Value**




### 34.1.12. Current Consumption of Peripheral Units

**Figure 34-40. ATmega48A: ADC Current vs. $V_{CC}$ ($AREF = AV_{CC}$)**

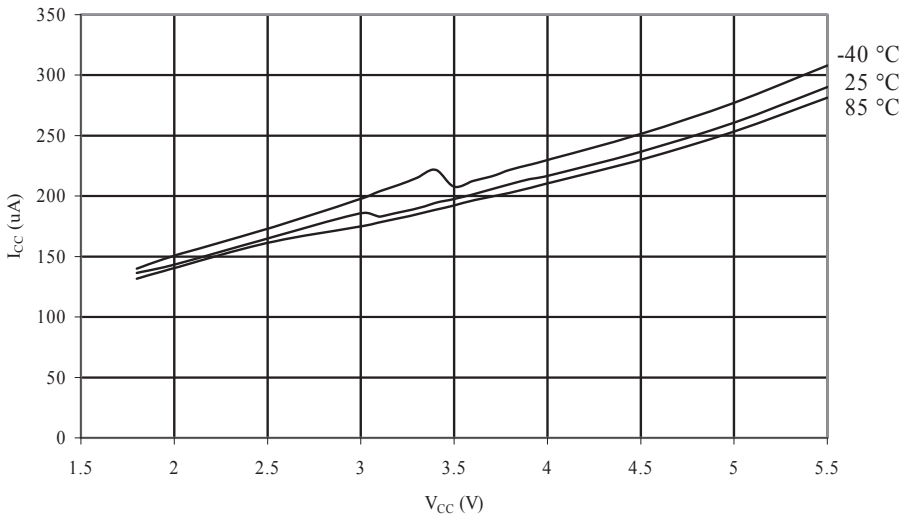


**Figure 34-41. ATmega48A: Analog Comparator Current vs. $V_{CC}$**

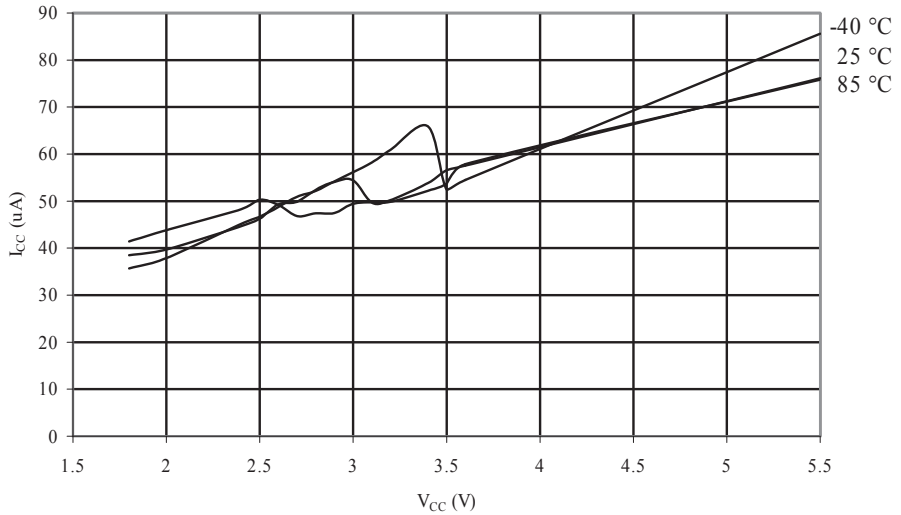



**Figure 34-42. ATmega48A: AREF External Reference Current vs. $V_{CC}$**

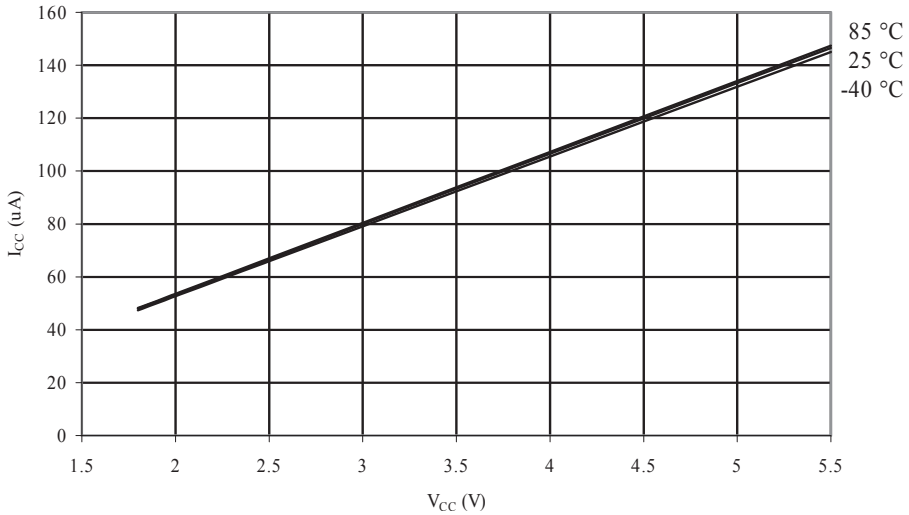



**Figure 34-43. ATmega48A: Brownout Detector Current vs. $V_{CC}$**

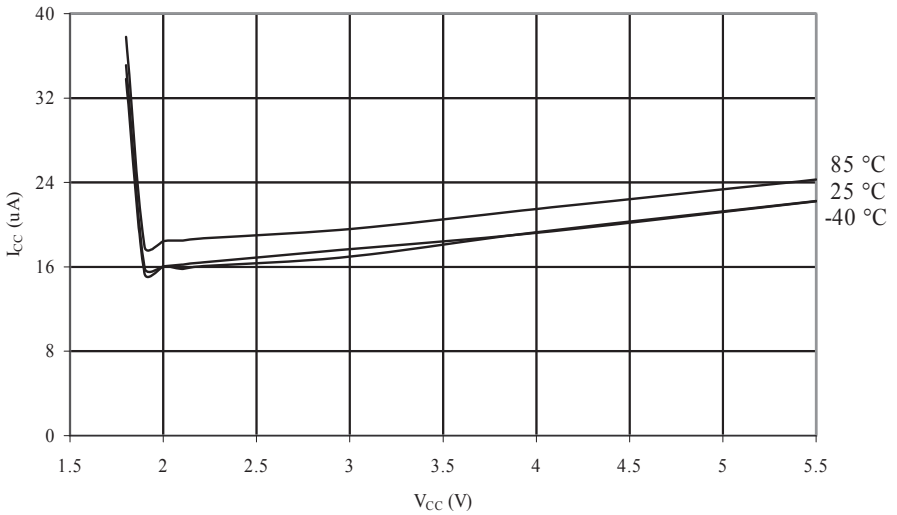
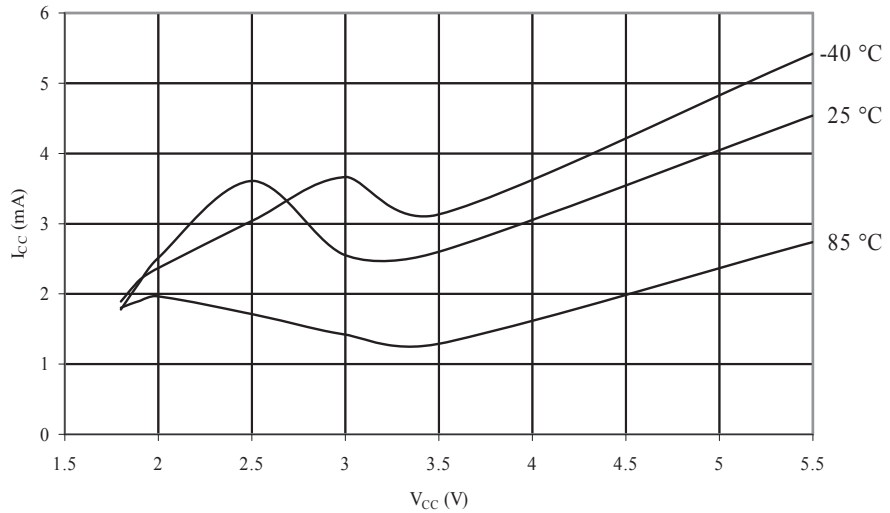


**Figure 34-44. ATmega48A: Programming Current vs. $V_{CC}$**




### 34.1.13. Current Consumption in Reset and Reset Pulsewidth

**Figure 34-45. ATmega48A: Reset Supply Current vs. Low Frequency (0.1MHz - 1.0MHz)**

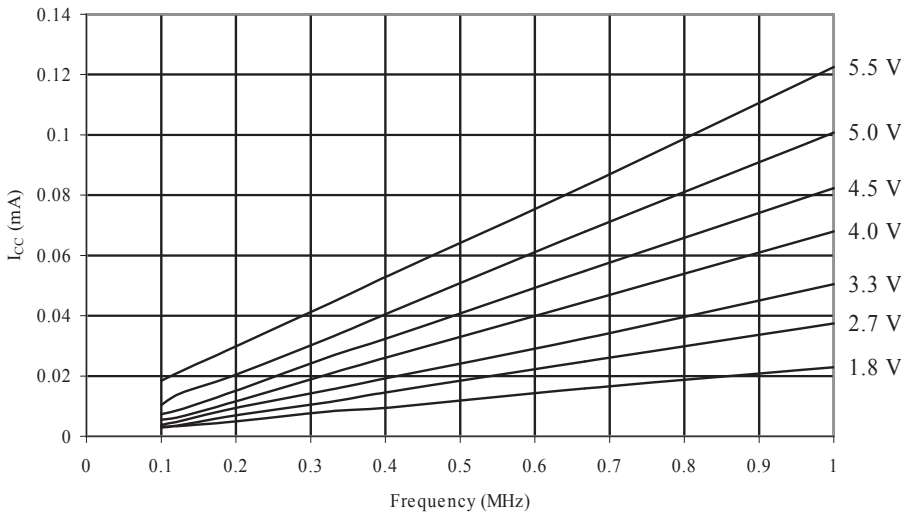


**Figure 34-46. ATmega48A: Reset Supply Current vs. Frequency (1MHz - 20MHz)**

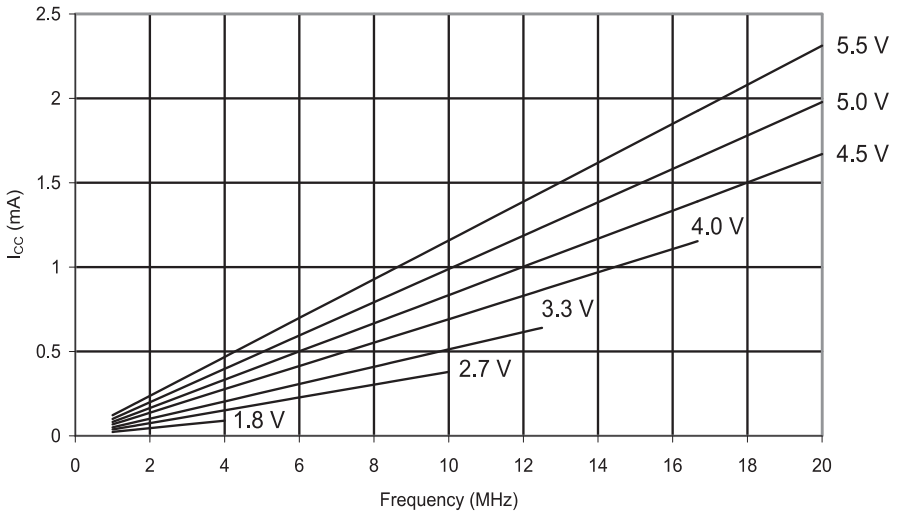
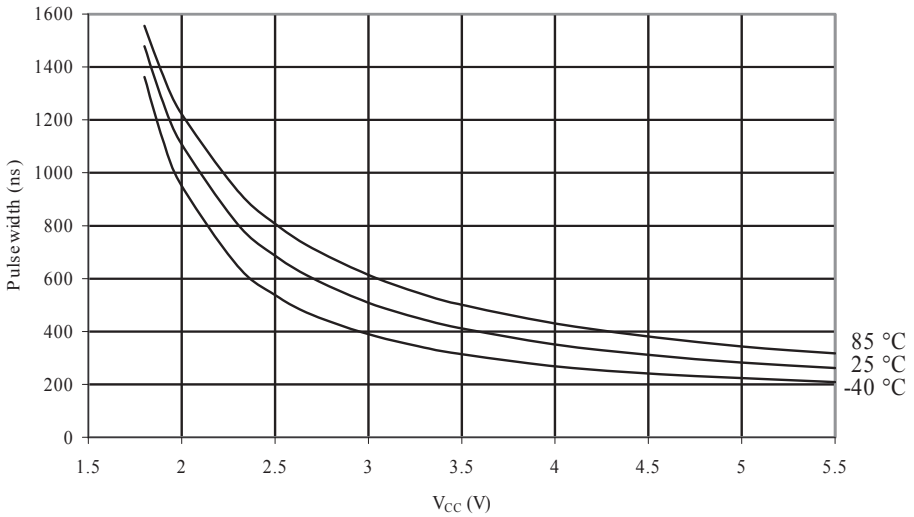



**Figure 34-47. ATmega48A: Minimum Reset Pulse Width vs. $V_{cc}$**



## 34.2. ATmega88A: Typical Characteristics

### 34.2.1. Active Supply Current

**Figure 34-48. ATmega88A: Active Supply Current vs. Low Frequency (0.1-1.0MHz)**

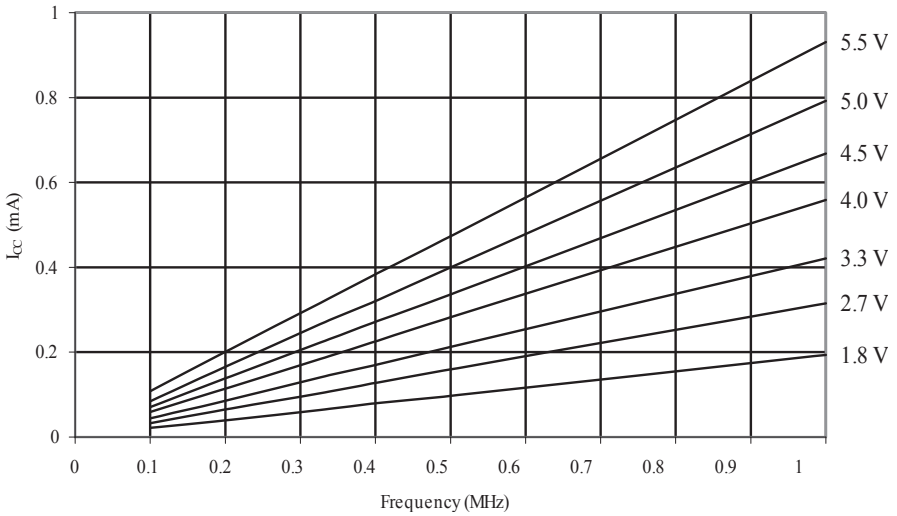



**Figure 34-49. ATmega88A: Active Supply Current vs. Frequency (1 - 20MHz)**

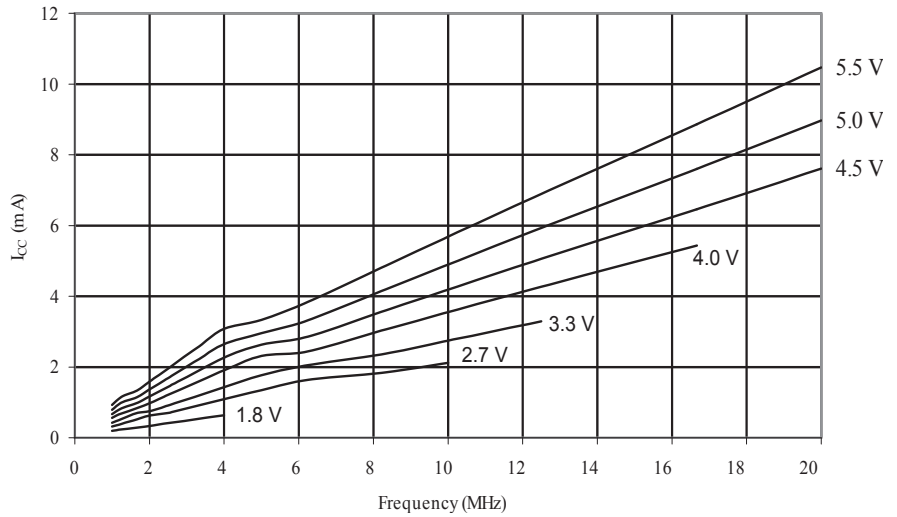


**Figure 34-50. ATmega88A: : Active Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 128kHz)**

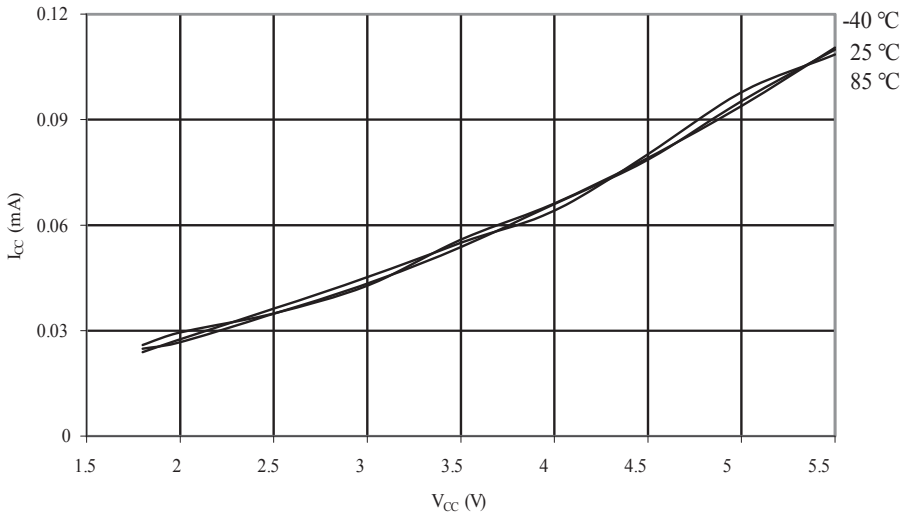



**Figure 34-51. ATmega88A: : Active Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 1MHz)**

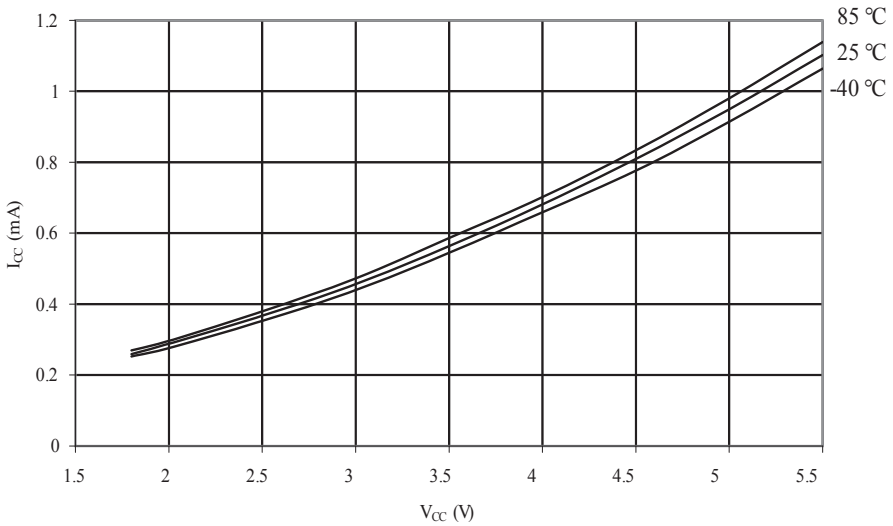
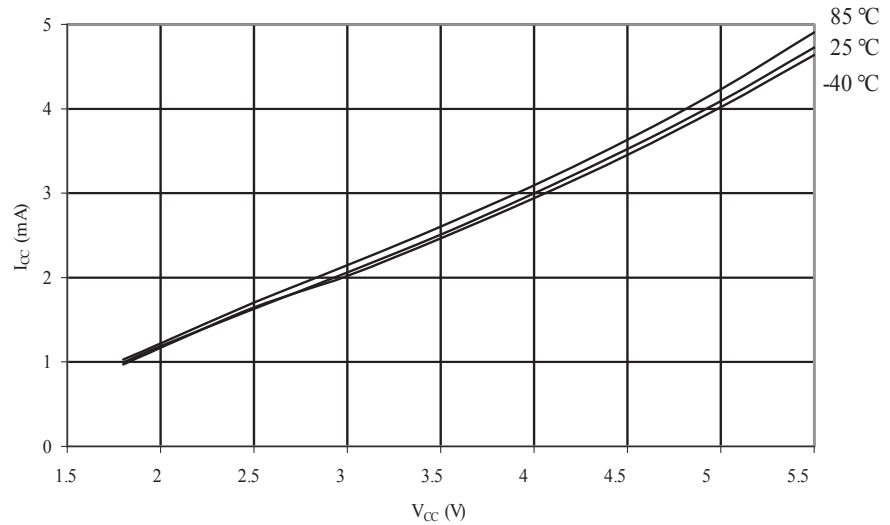


**Figure 34-52. ATmega88A: : Active Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 8MHz)**




### 34.2.2. Idle Supply Current

**Figure 34-53. ATmega88A: Idle Supply Current vs. Low Frequency (0.1-1.0MHz)**

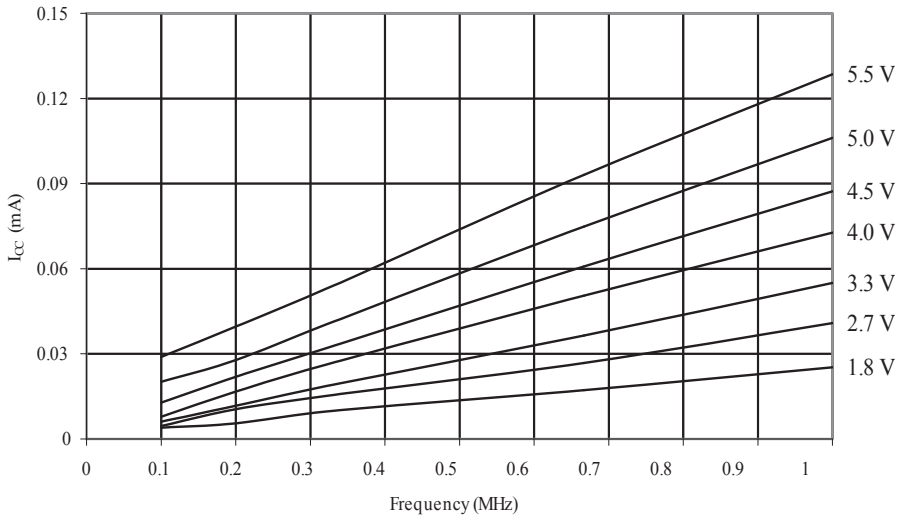


**Figure 34-54. ATmega88A: Idle Supply Current vs. Frequency (1-20MHz)**

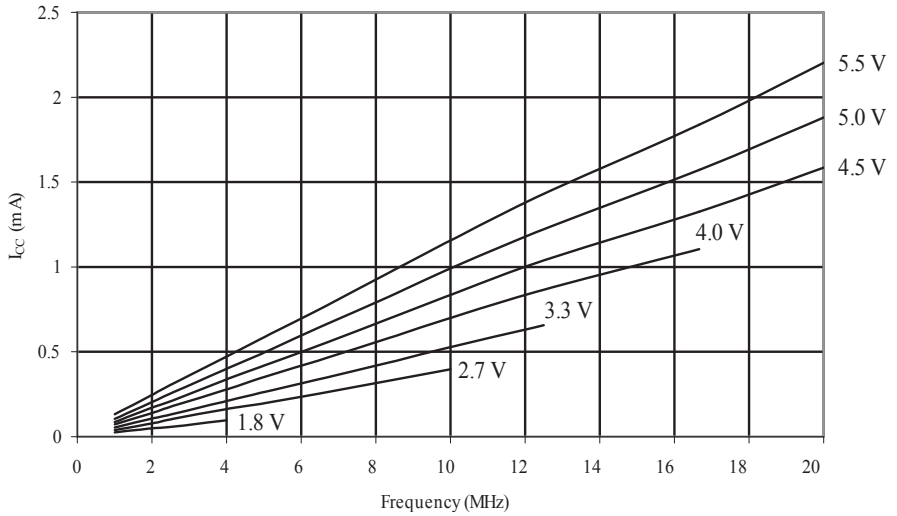



**Figure 34-55. ATmega88A: Idle Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 128kHz)**

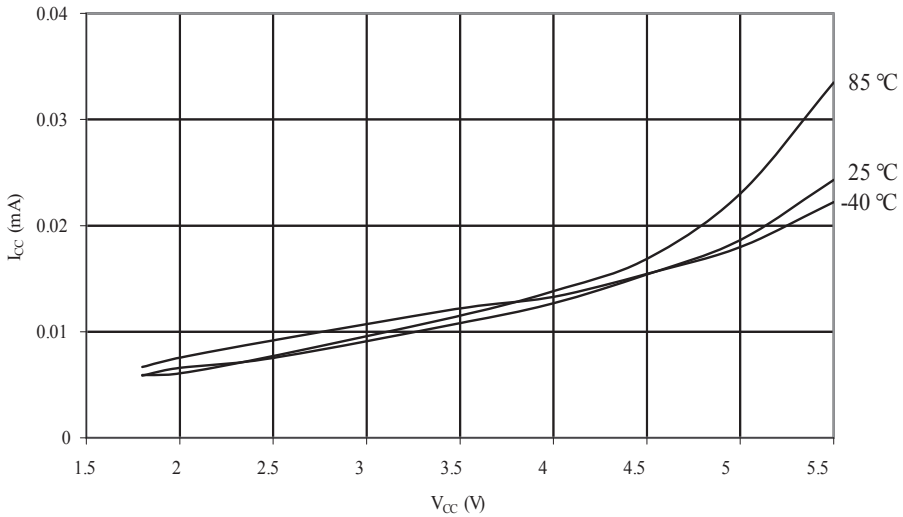


**Figure 34-56. ATmega88A: Idle Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 1MHz)**

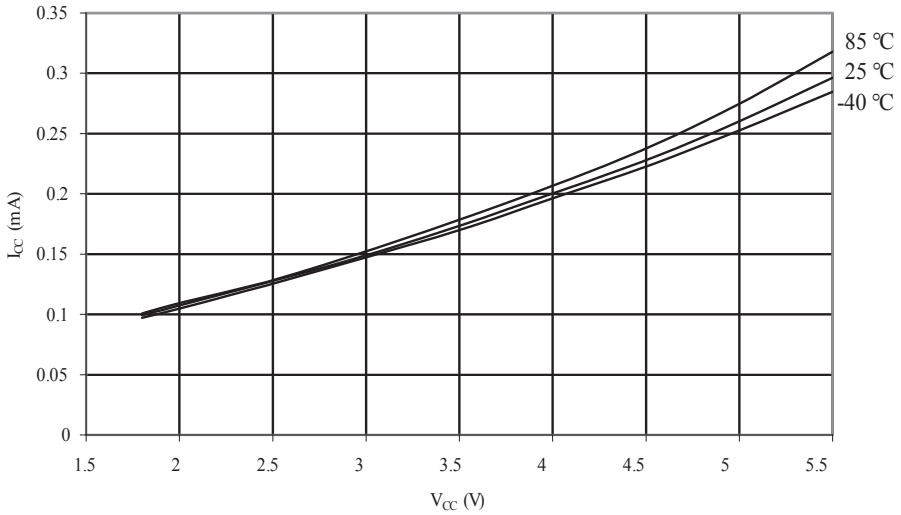
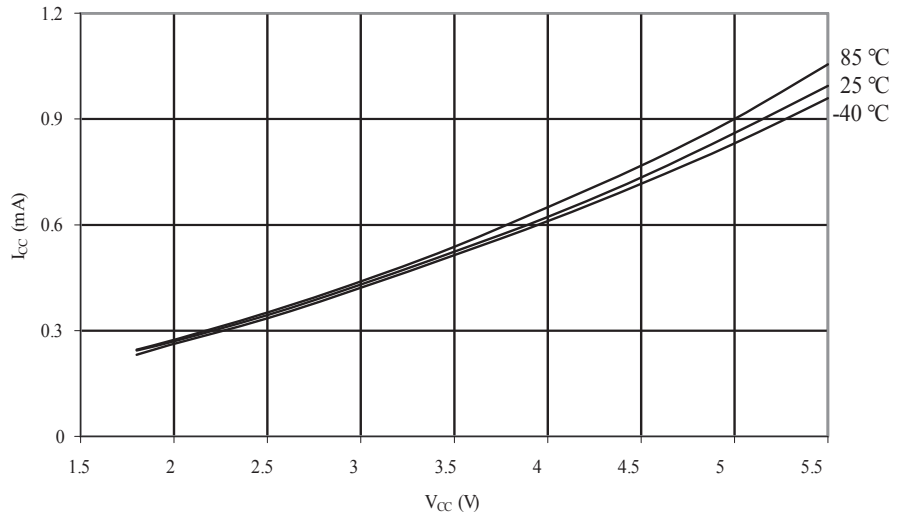



**Figure 34-57. ATmega88A: Idle Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 8MHz)**




### 34.2.3. Supply Current of IO Modules

The tables and formulas below can be used to calculate the additional current consumption for the different I/O modules in Active and Idle mode. The enabling or disabling of the I/O modules are controlled by the Power Reduction Register. See ”Power Reduction Register” for details.

**Table 34-3. Additional Current Consumption for the different I/O modules (absolute values)**

| PRR bit | Typical numbers | | |
|---|---|---|---|
| | $V_{CC}$ = 2V, F = 1MHz | $V_{CC}$ = 3V, F = 4MHz | $V_{CC}$ = 5V, F = 8MHz |
| PRUSART0 | 3.0µA | 21.3µA | 97.9µA |
| PRTWI | 6.1µA | 45.4µA | 219.0µA |
| PRTIM2 | 5.2µA | 35.2µA | 149.5µA |
| PRTIM1 | 3.8µA | 25.6µA | 110.0µA |

| PRR bit | Typical numbers | | |
|---|---|---|---|
| | $V_{CC}$ = 2V, F = 1MHz | $V_{CC}$ = 3V, F = 4MHz | $V_{CC}$ = 5V, F = 8MHz |
| PRTIM0 | 1.5µA | 9.8µA | 39.6µA |
| PRSPI | 5.2µA | 40.0µA | 199.6µA |
| PRADC | 6.3µA | 48.7µA | 247.0µA |

**Table 34-4. Additional Current Consumption (percentage) in Active and Idle mode**

| PRR bit | Additional Current consumption compared to Active with external clock (see Figure 34-48 and Figure 34-49) | Additional Current consumption compared to Idle with external clock (see Figure 34-53 and Figure 34-54) |
|---|---|---|
| PRUSART0 | 1.8% | 11.4% |
| PRTWI | 3.9% | 24.4% |
| PRTIM2 | 2.9% | 18.6% |
| PRTIM1 | 2.1% | 13.6% |
| PRTIM0 | 0.8% | 5.2% |
| PRSPI | 3.5% | 21.5% |
| PRADC | 4.2% | 26.3% |

It is possible to calculate the typical current consumption based on the numbers from the above table for other $V_{CC}$ and frequency settings.

**Related Links**

PRR on page 73

#### 34.2.3.1. Example

Calculate the expected current consumption in idle mode with TIMER1, ADC, and SPI enabled at $V_{CC}$ = 2.0V and F = 1MHz. From Table *Additional Current Consumption (percentage) in Active and Idle mode* in the previous section, third column, we see that we need to add 13.6% for the TIMER1, 26.3% for the ADC, and 21.5% for the SPI module. Reading from Figure *Idle Supply Current vs. Low Frequency (0.1-1.0MHz)*, we find that the idle current consumption is ~0.027mA at $V_{CC}$ = 2.0V and F = 1MHz. The total current consumption in idle mode with TIMER1, ADC, and SPI enabled, gives:

$$\mathrm{Icctotal} \simeq 0.027\ \mathrm{mA} \cdot (1 + 0.136 + 0.263 + 0.215) \simeq 0.043\ \mathrm{mA}$$

**Related Links**

PRR on page 73

### 34.2.4. Power-down Supply Current

**Figure 34-58. ATmega88A: Power-Down Supply Current vs. $V_{CC}$ (Watchdog Timer Disabled)**

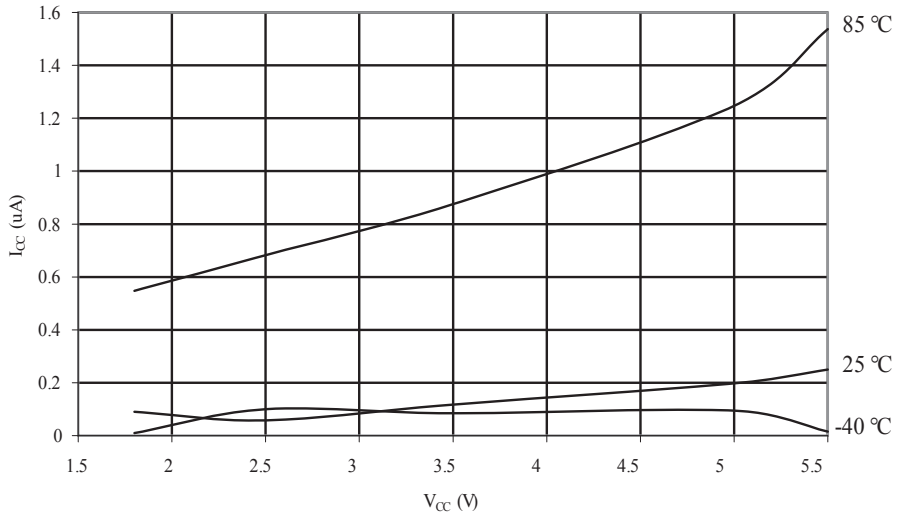
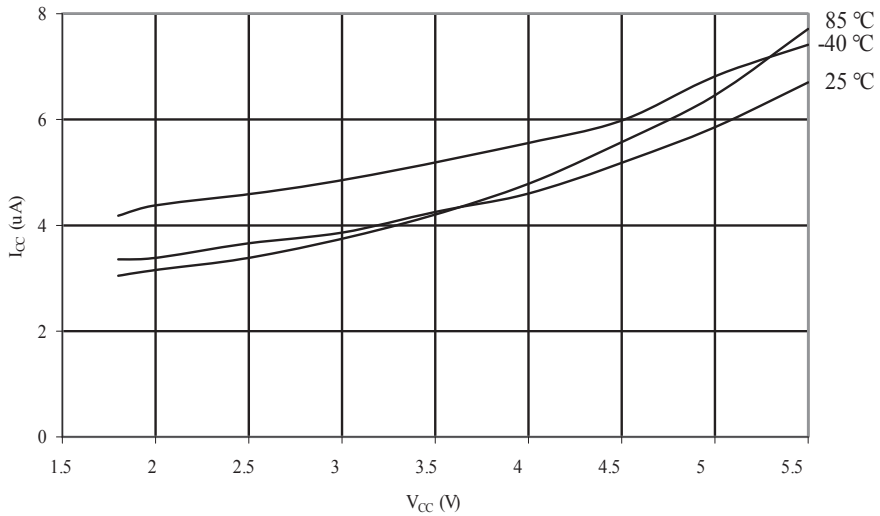
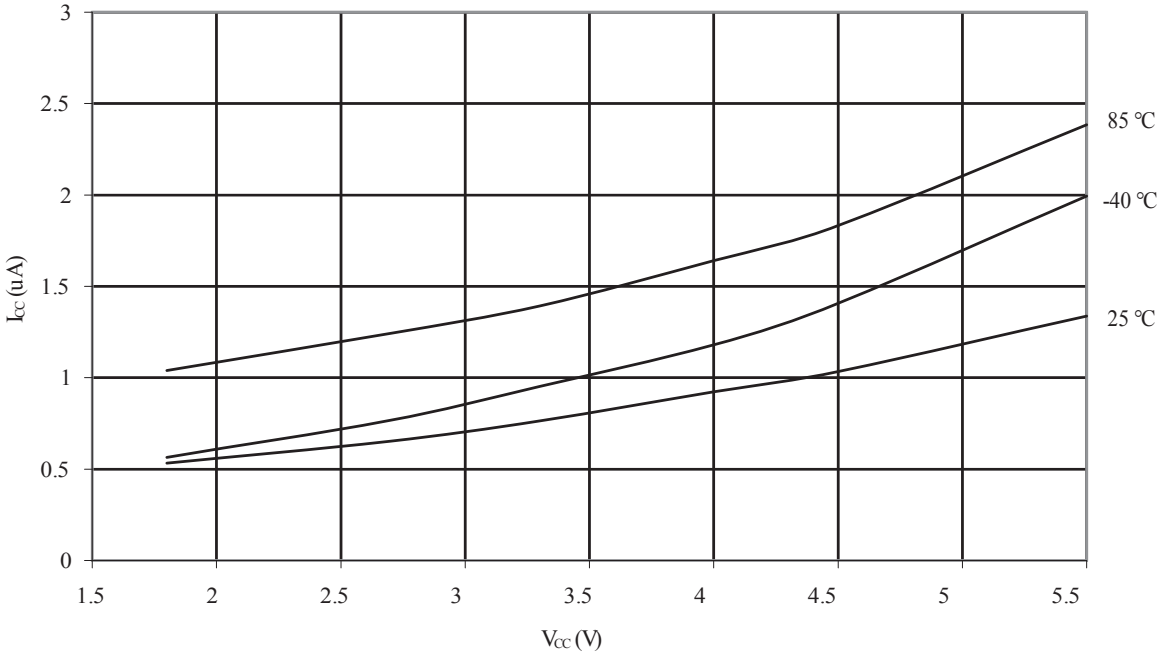



**Figure 34-59. ATmega88A: Power-Down Supply Current vs. $V_{CC}$ (Watchdog Timer Enabled)**



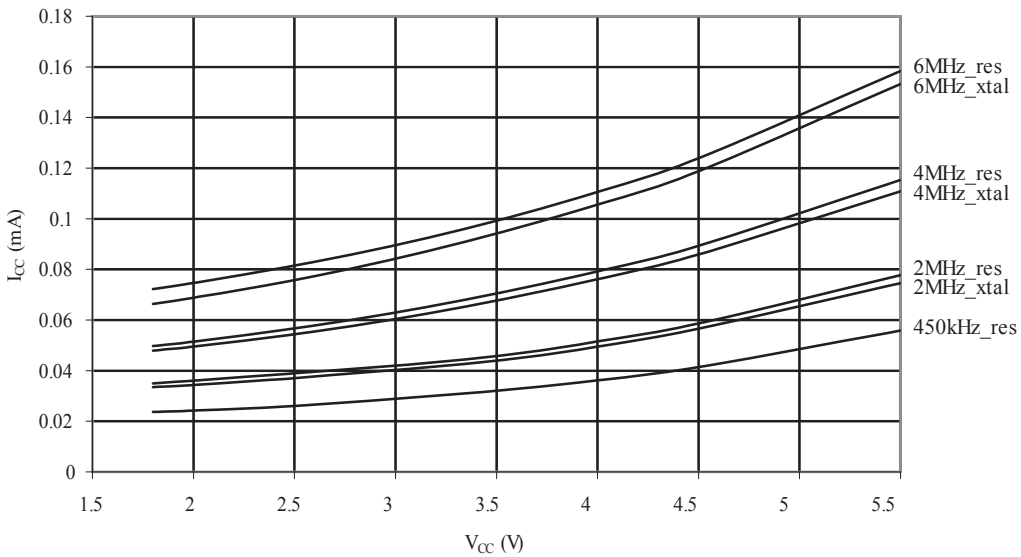
### 34.2.5. Power-save Supply Current

**Figure 34-60. ATmega88A: Power-Save Supply Current vs. $V_{CC}$ (Watchdog Timer Disabled and 32kHz Crystal Oscillator Running)**




### 34.2.6. Standby Supply Current

**Figure 34-61. ATmega88A: Standby Supply Current vs. $V_{CC}$ (Watchdog Timer Disabled)**



### 34.2.7. Pin Pull-Up

**Figure 34-62. ATmega88A: I/O Pin Pull-up Resistor Current vs. Input Voltage ($V_{CC}$ = 1.8 V)**

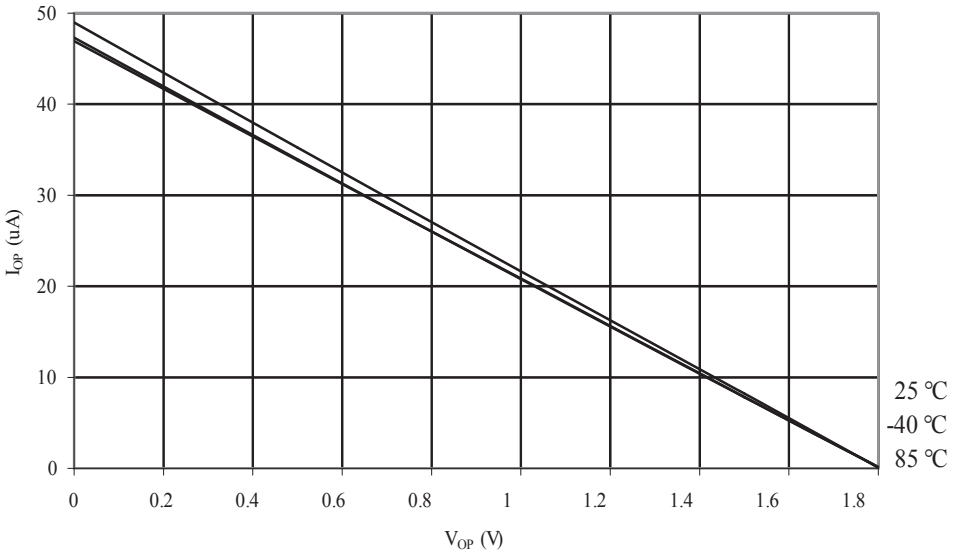



**Figure 34-63. ATmega88A: I/O Pin Pull-up Resistor Current vs. Input Voltage ($V_{CC}$ = 2.7 V)**

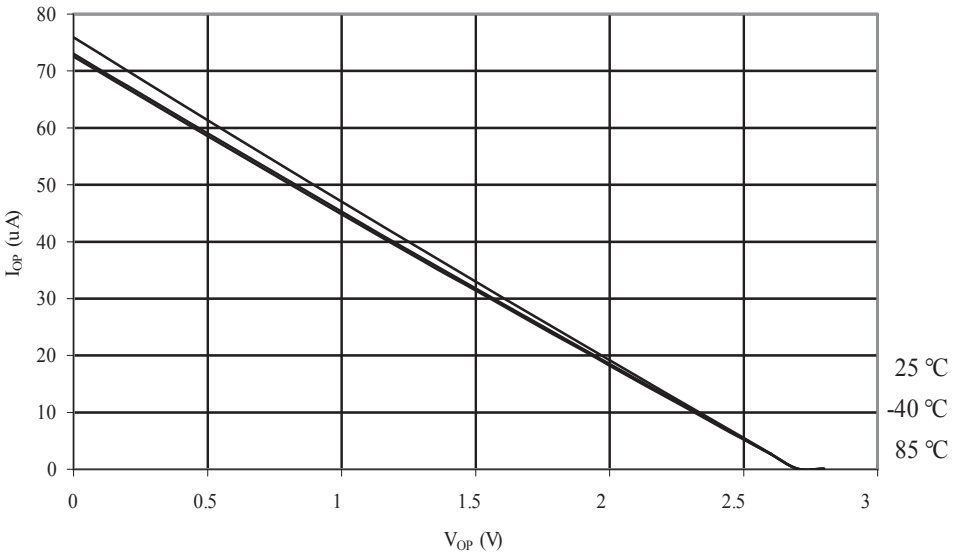


**Figure 34-64. ATmega88A: I/O Pin Pull-up Resistor Current vs. Input Voltage ($V_{CC}$ = 5 V)**

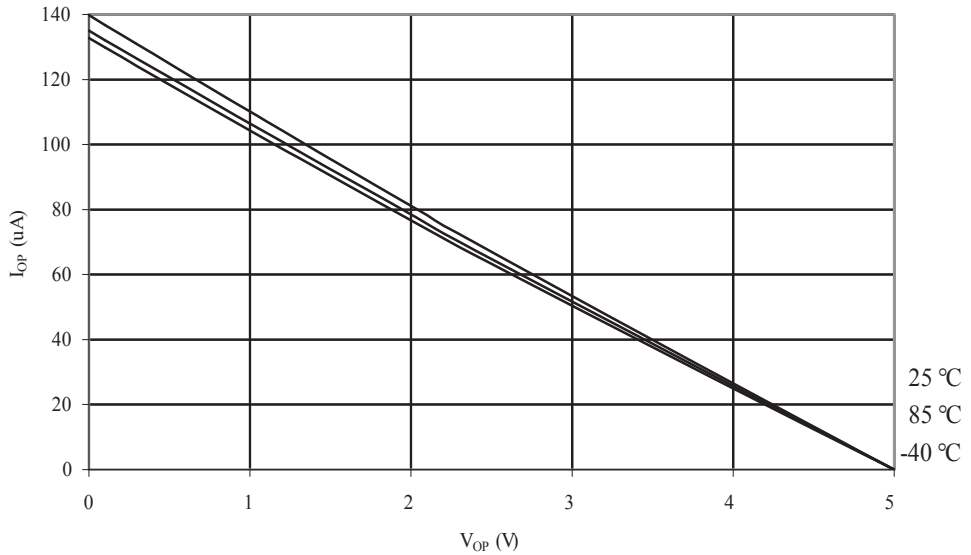



**Figure 34-65. ATmega88A: Reset Pull-up Resistor Current vs. Reset Pin Voltage ($V_{CC}$ = 1.8 V)**

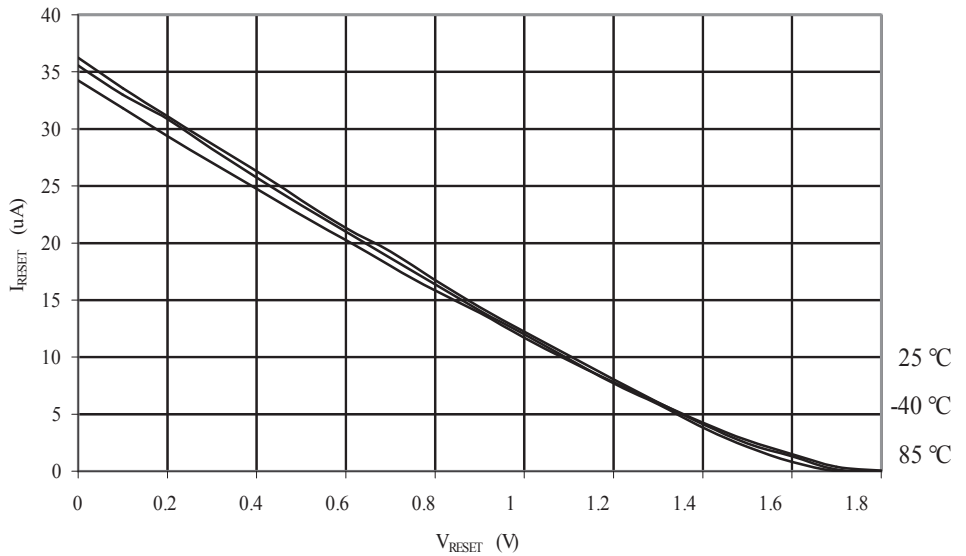


**Figure 34-66. ATmega88A: Reset Pull-up Resistor Current vs. Reset Pin Voltage ($V_{CC}$ = 2.7 V)**

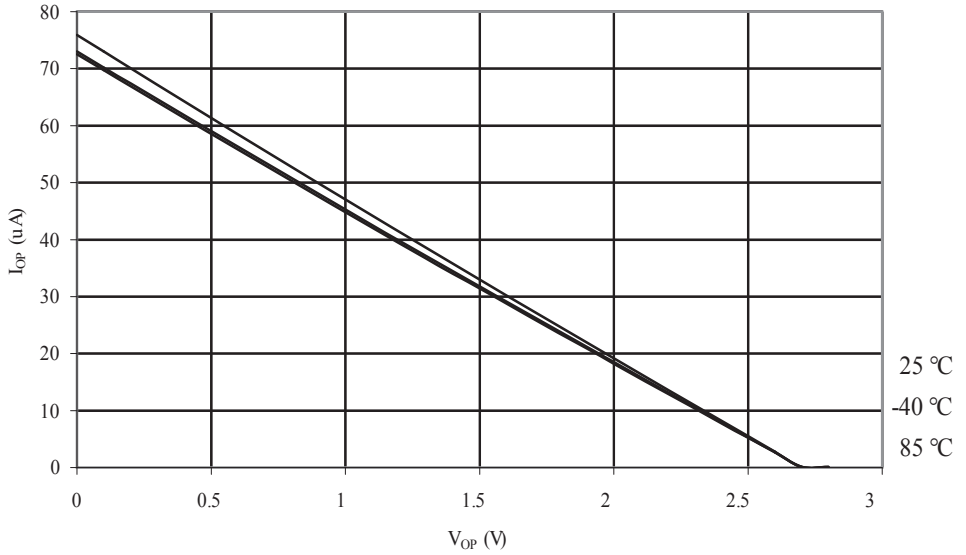
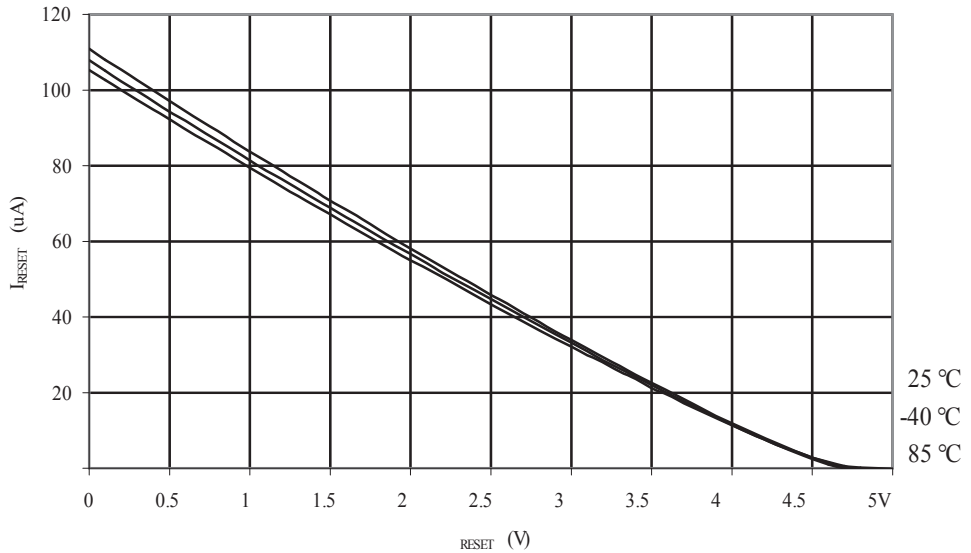



**Figure 34-67. ATmega88A: Reset Pull-up Resistor Current vs. Reset Pin Voltage ($V_{CC}$ = 5 V)**



### 34.2.8. Pin Driver Strength

**Figure 34-68. ATmega88A: I/O Pin Output Voltage vs. Sink Current ($V_{CC}$ = 3 V)**

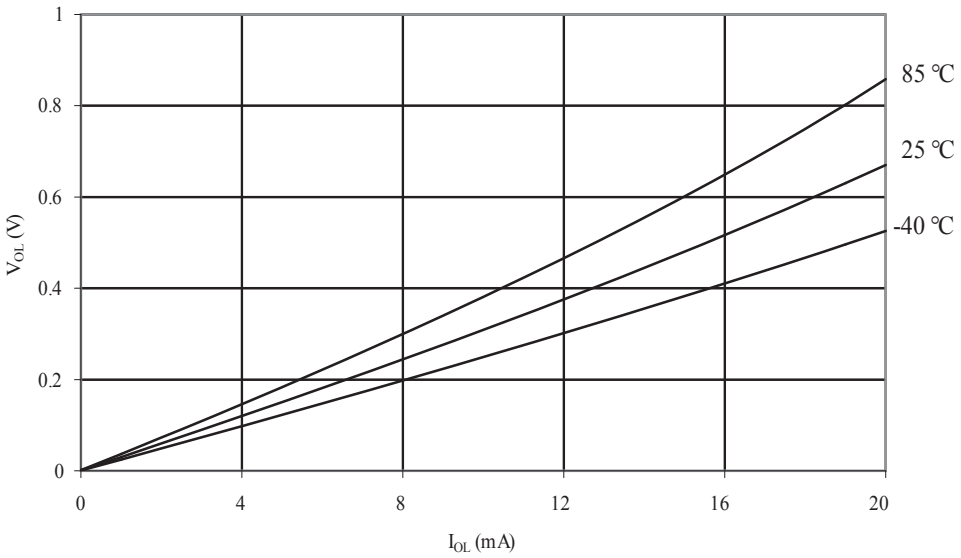



**Figure 34-69. ATmega88A: I/O Pin Output Voltage vs. Sink Current ($V_{CC}$ = 5 V)**

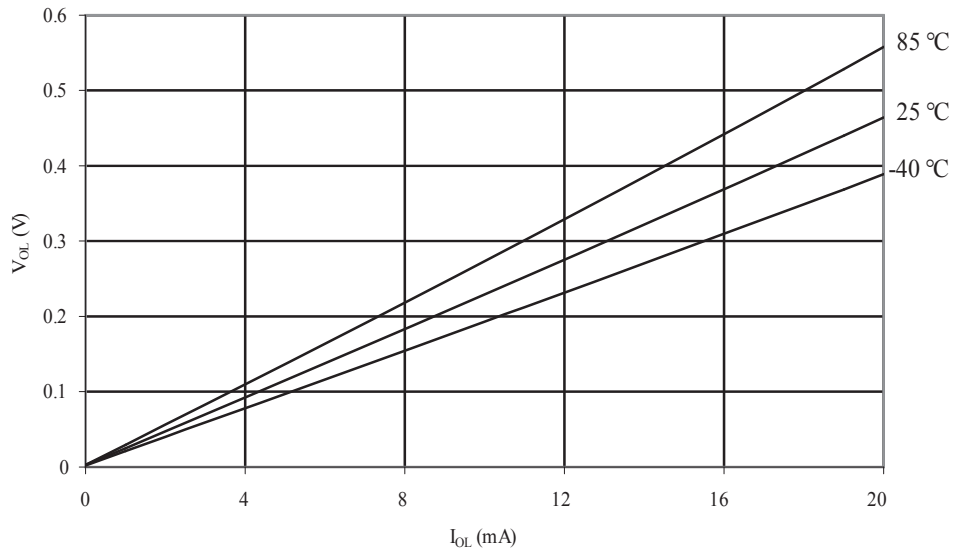


**Figure 34-70. ATmega88A: I/O Pin Output Voltage vs. Source Current ($V_{CC}$ = 3 V)**

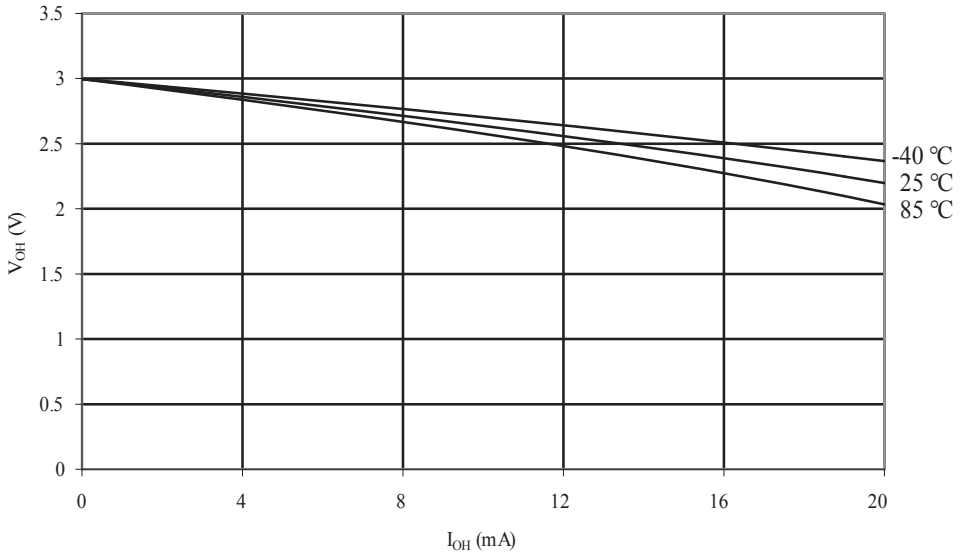
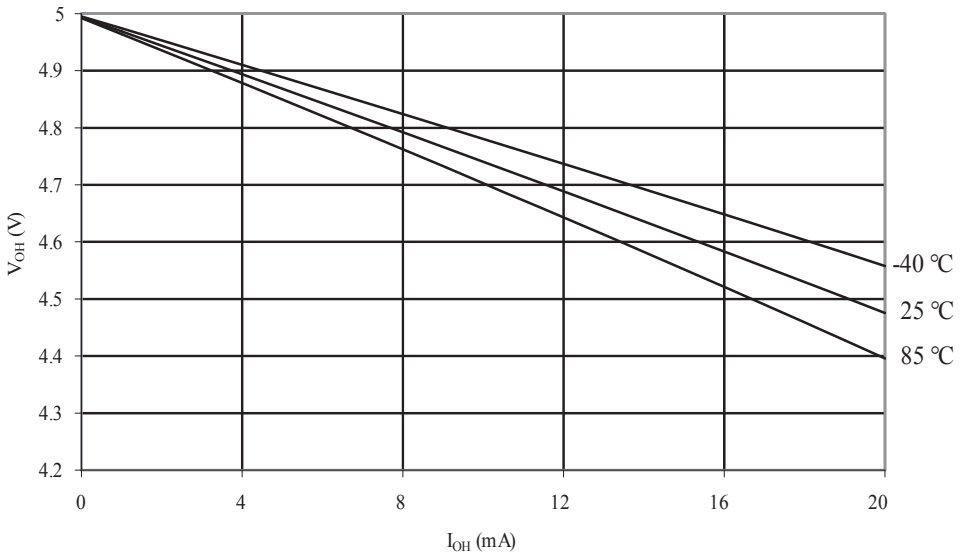



**Figure 34-71. ATmega88A: I/O Pin Output Voltage vs. Source Current ($V_{CC}$ = 5 V)**



### 34.2.9. Pin Threshold and Hysteresis

**Figure 34-72. ATmega88A: I/O Pin Input Threshold Voltage vs. $V_{CC}$ ($V_{IH}$, I/O Pin read as '1')**

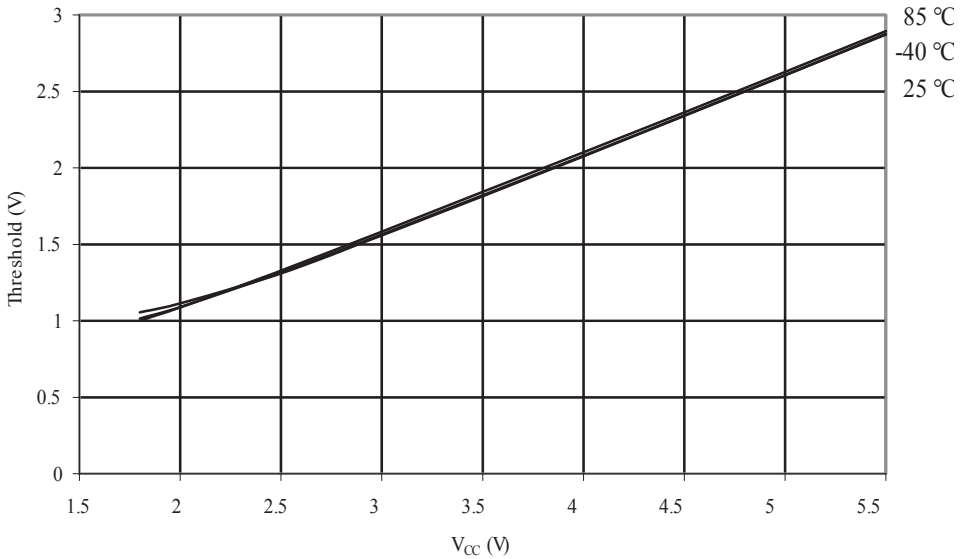



**Figure 34-73. ATmega88A: I/O Pin Input Threshold Voltage vs. $V_{CC}$ ($V_{IL}$, I/O Pin read as '0'**

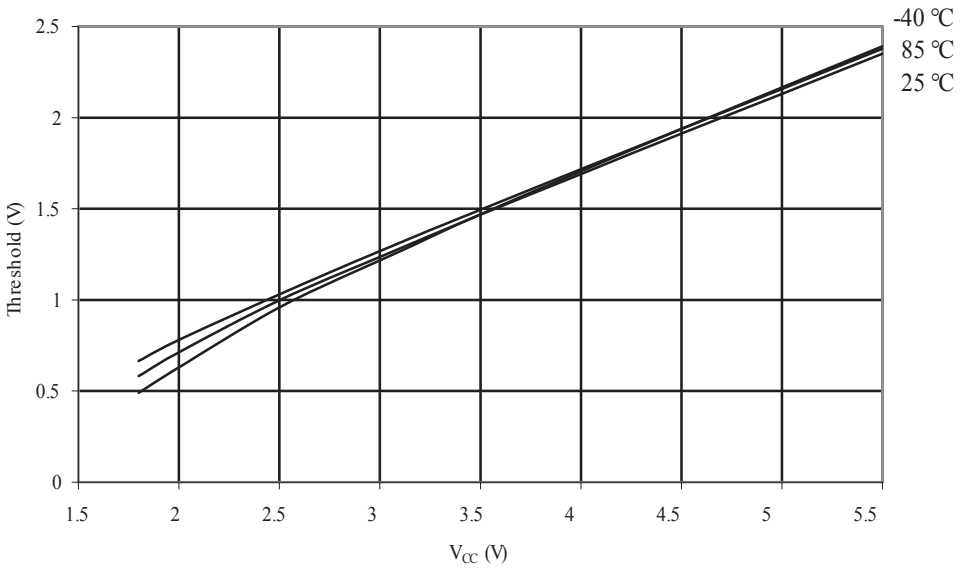


**Figure 34-74. ATmega88A: I/O Pin Input Hysteresis vs. $V_{CC}$**

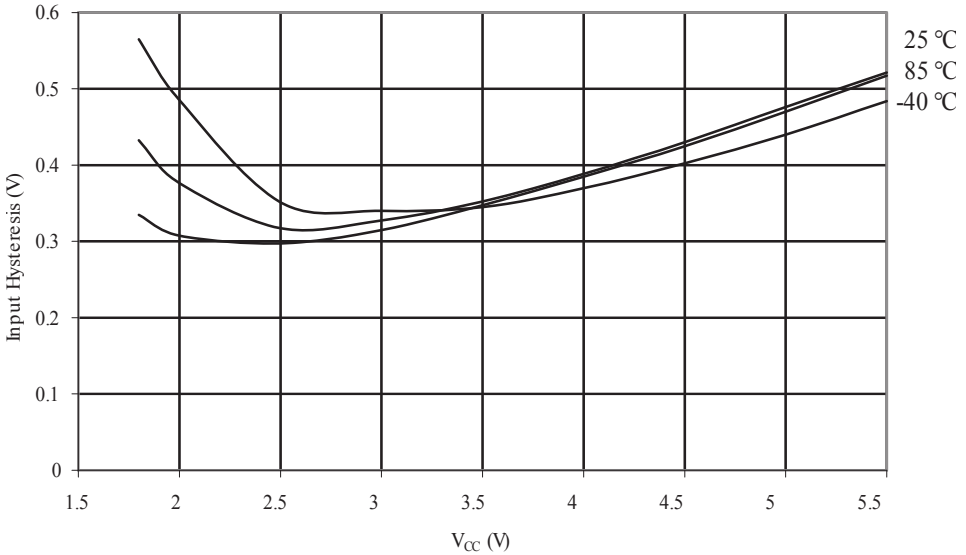



**Figure 34-75. ATmega88A: Reset Input Threshold Voltage vs. $V_{CC}$ ($V_{IH}$, I/O Pin read as '1')**

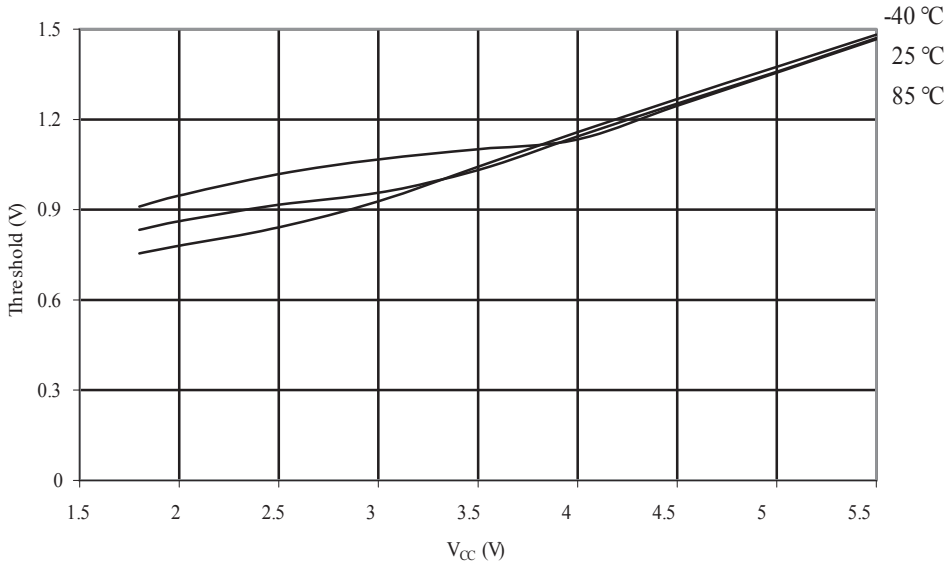


**Figure 34-76. ATmega88A: Reset Input Threshold Voltage vs. $V_{CC}$ ($V_{IL}$, I/O Pin read as '0')**

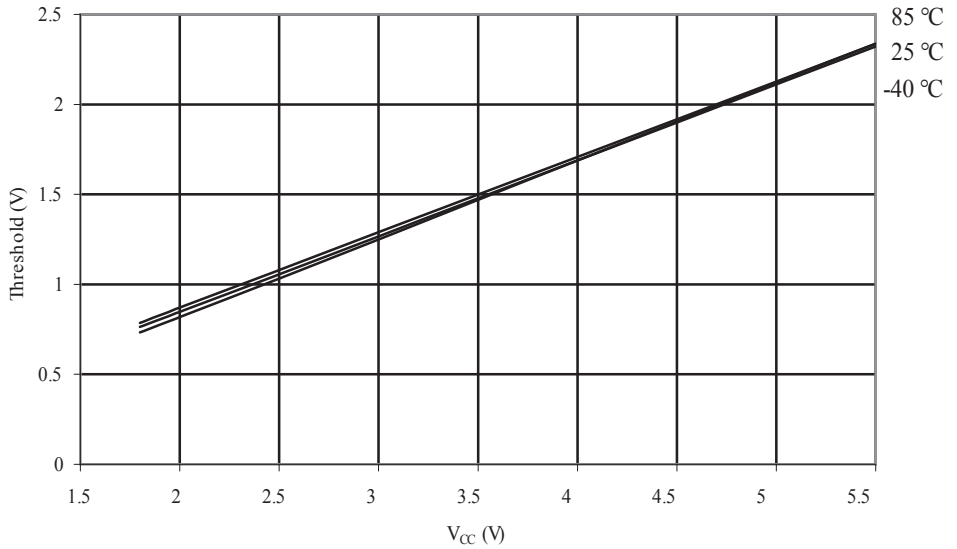
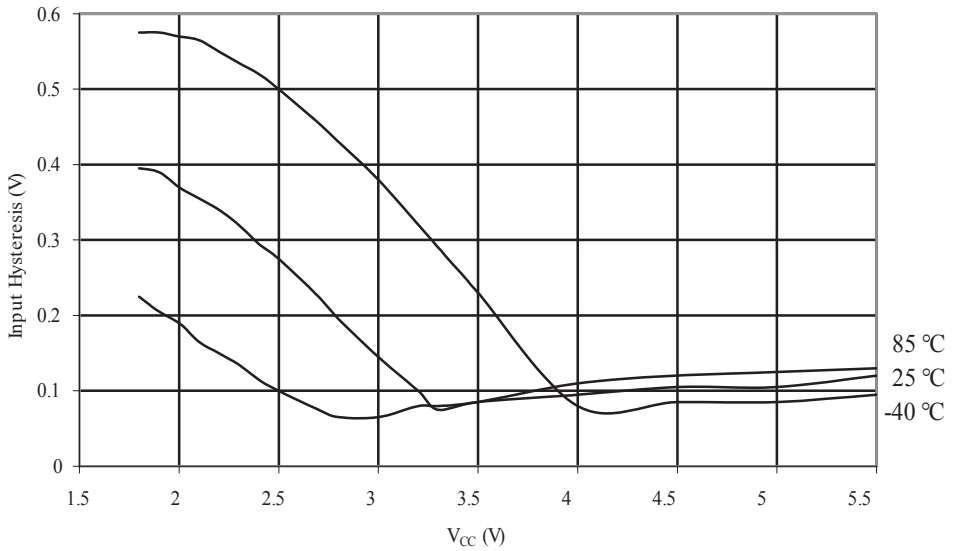



**Figure 34-77. ATmega88A: Reset Pin Input Hysteresis vs. $V_{CC}$**



### 34.2.10. BOD Threshold

**Figure 34-78. ATmega88A: : BOD Thresholds vs. Temperature (BODLEVEL is 1.8 V)**

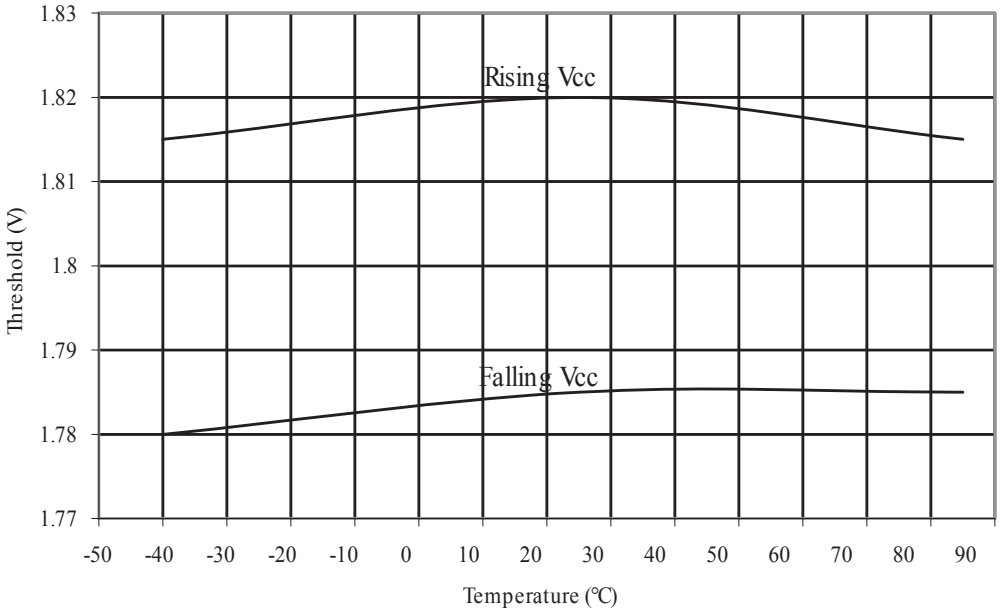



**Figure 34-79. ATmega88A: BOD Thresholds vs. Temperature (BODLEVEL is 2.7 V)**

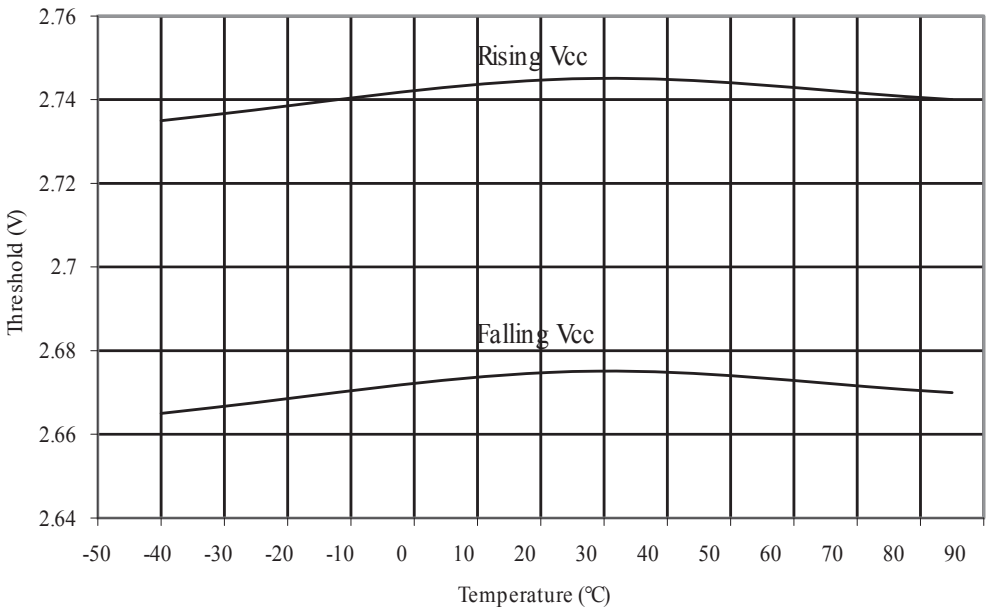


**Figure 34-80. ATmega88A: BOD Thresholds vs. Temperature (BODLEVEL is 4.3 V)**

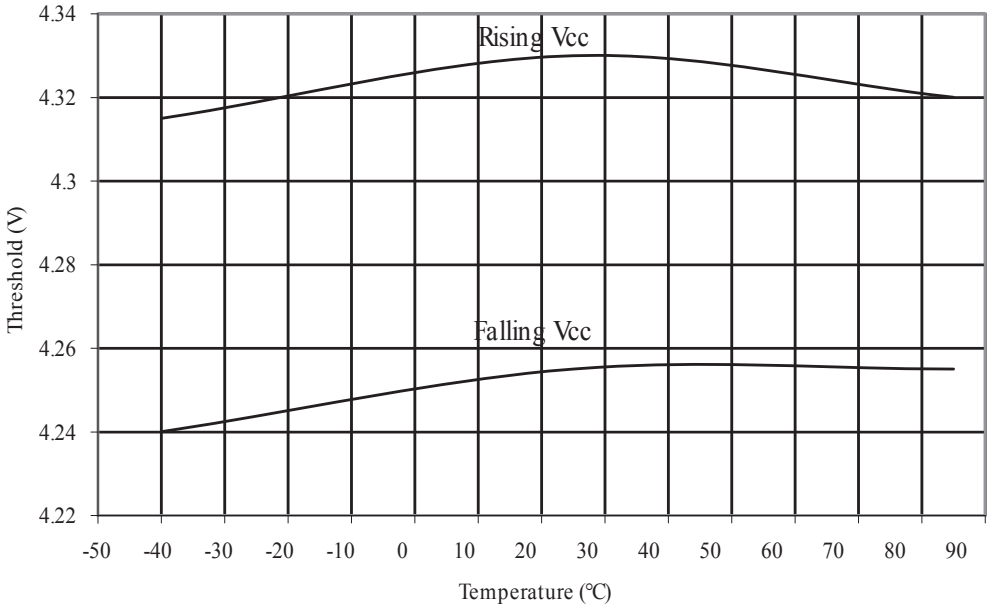
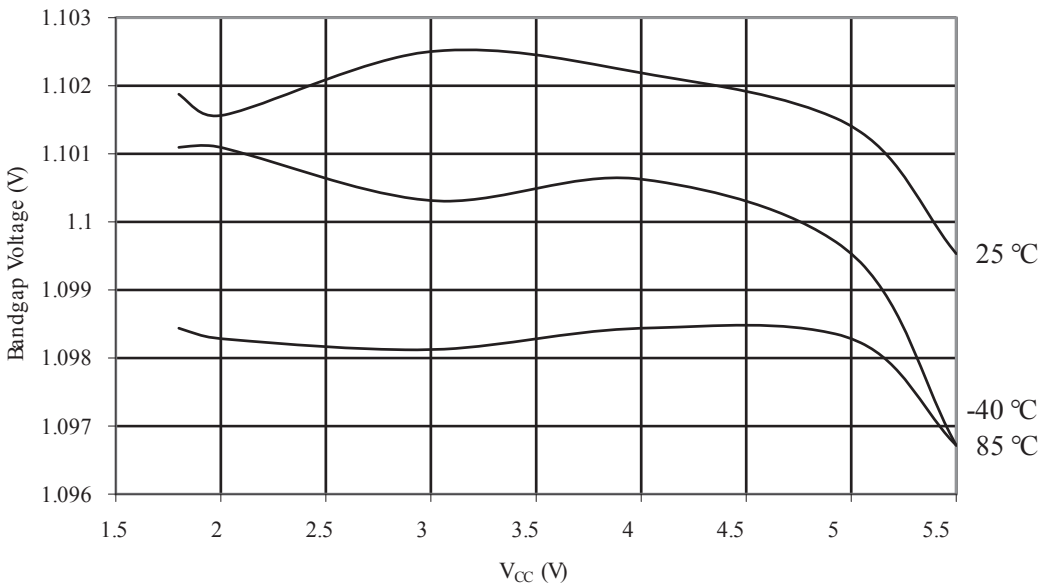



**Figure 34-81. ATmega88A: Bandgap Voltage vs. $V_{CC}$**



### 34.2.11. Internal Oscillator Speed

**Figure 34-82. ATmega88A: Watchdog Oscillator Frequency vs. Temperature**

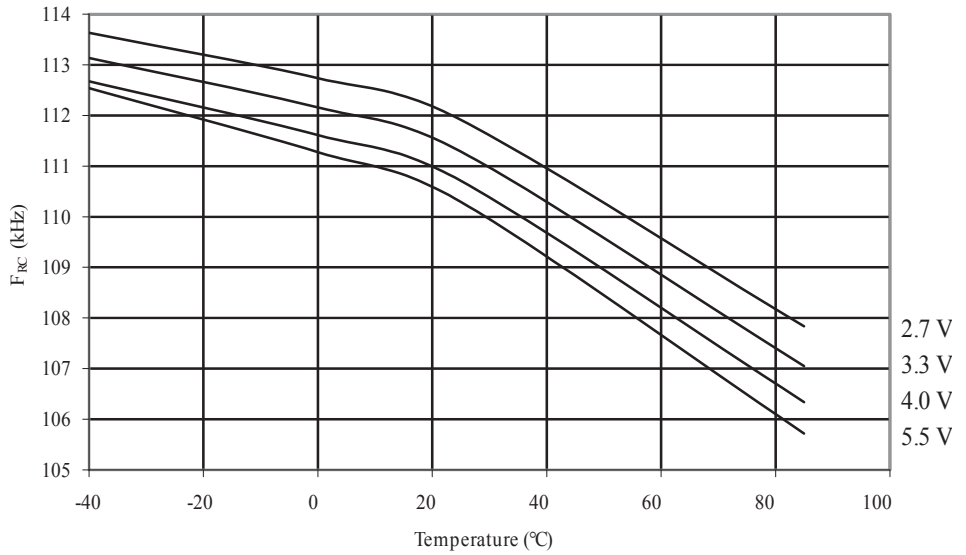



**Figure 34-83. ATmega88A: Watchdog Oscillator Frequency vs. $V_{CC}$**

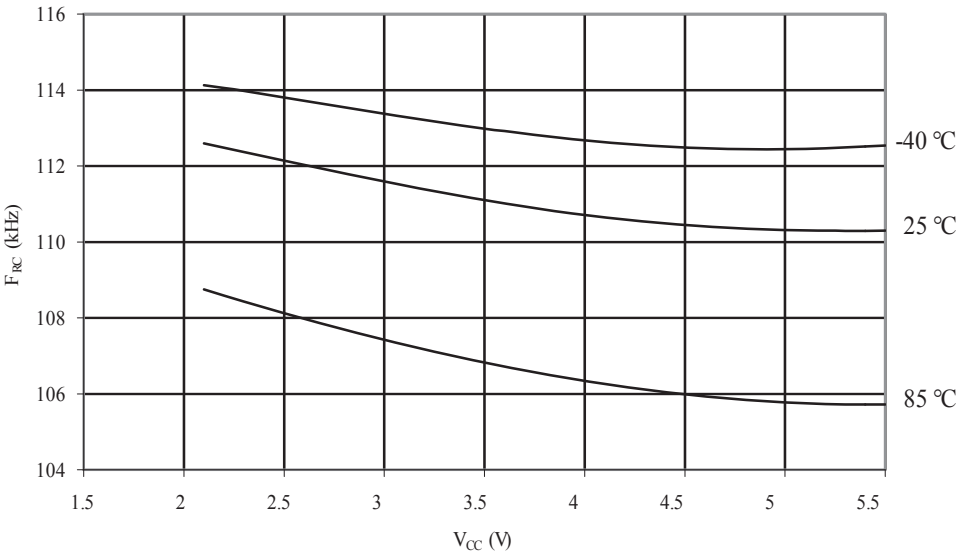


**Figure 34-84. ATmega88A: Calibrated 8MHz RC Oscillator Frequency vs. $V_{CC}$**

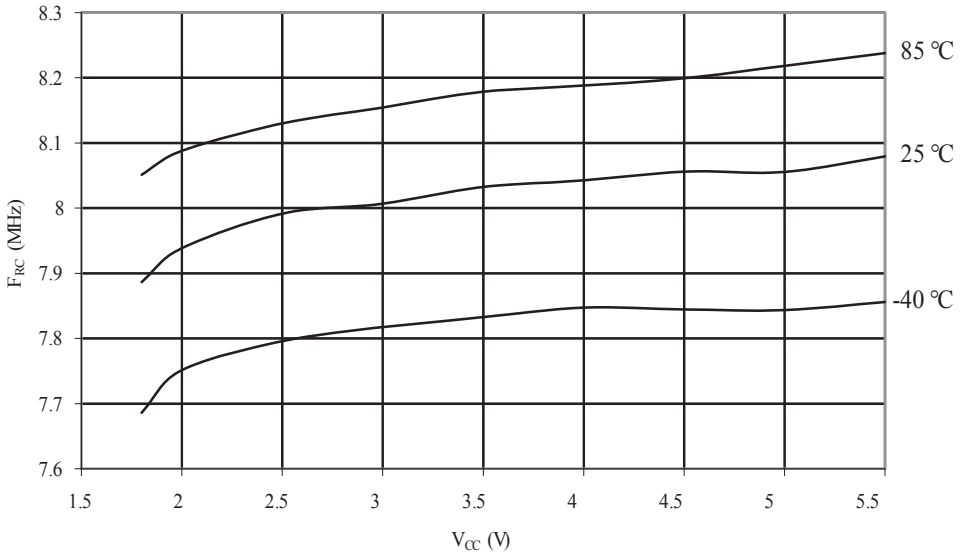



**Figure 34-85. ATmega88A: Calibrated 8MHz RC Oscillator Frequency vs. Temperature**

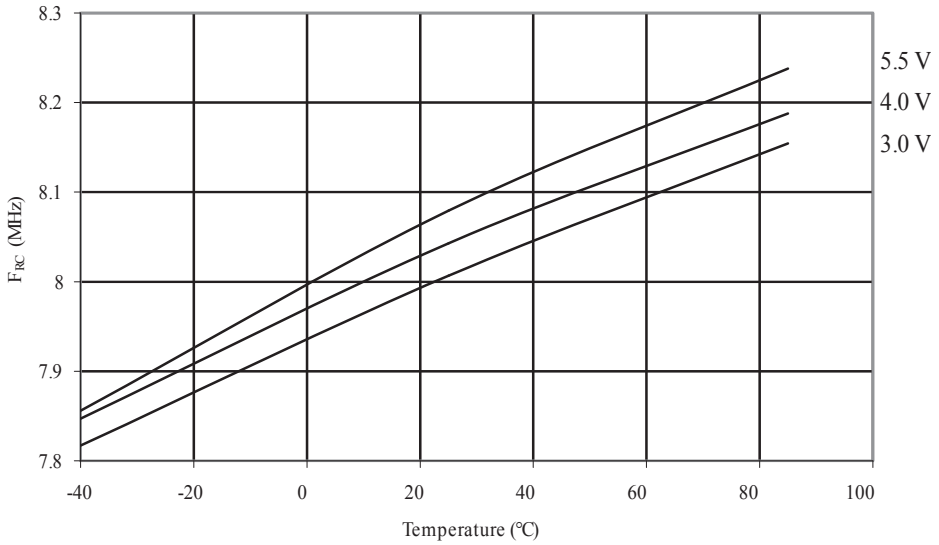
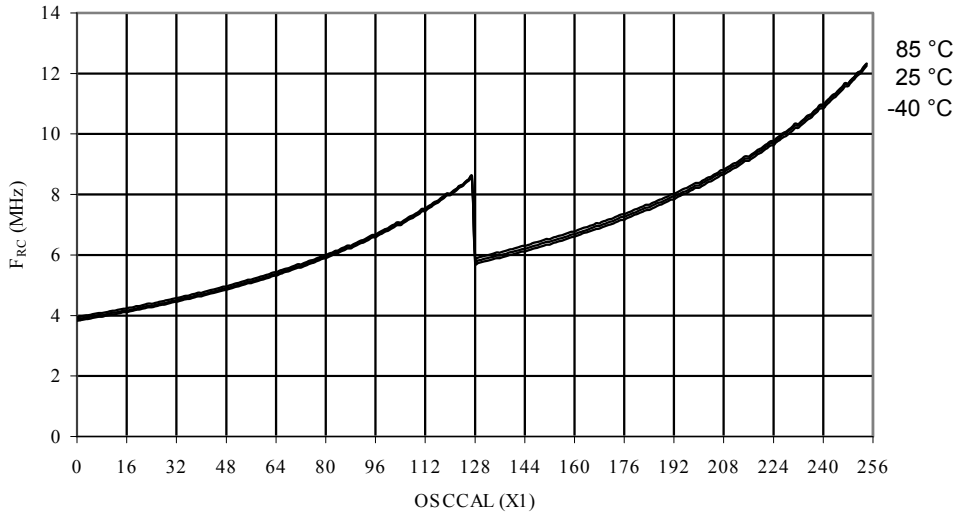


**Figure 34-86. ATmega88A: Calibrated 8MHz RC Oscillator Frequency vs. OSCCAL Value**




### 34.2.12. Current Consumption of Peripheral Units

**Figure 34-87. ATmega88A: ADC Current vs. $V_{CC}$ (AREF = $AV_{CC}$)**

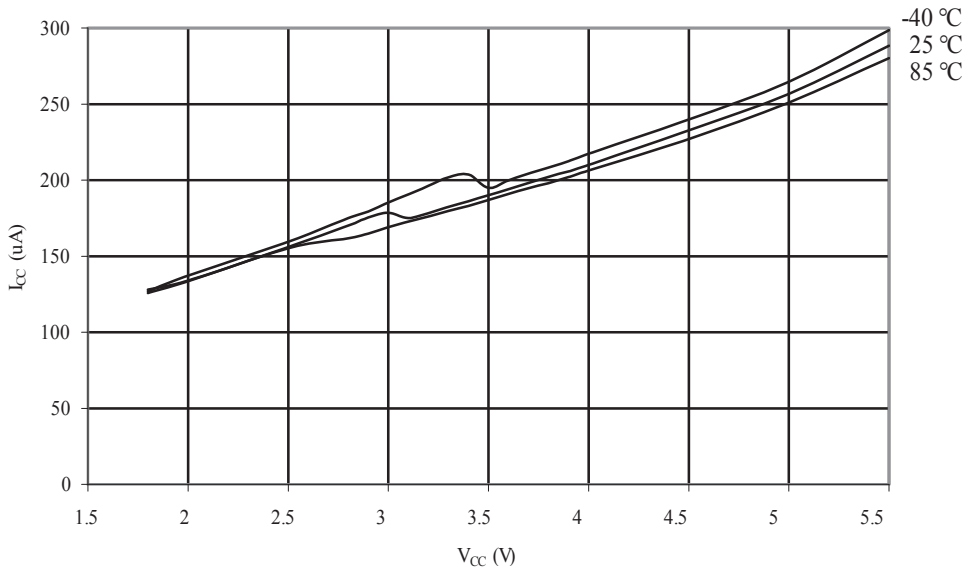


**Figure 34-88. ATmega88A: Analog Comparator Current vs. $V_{CC}$**

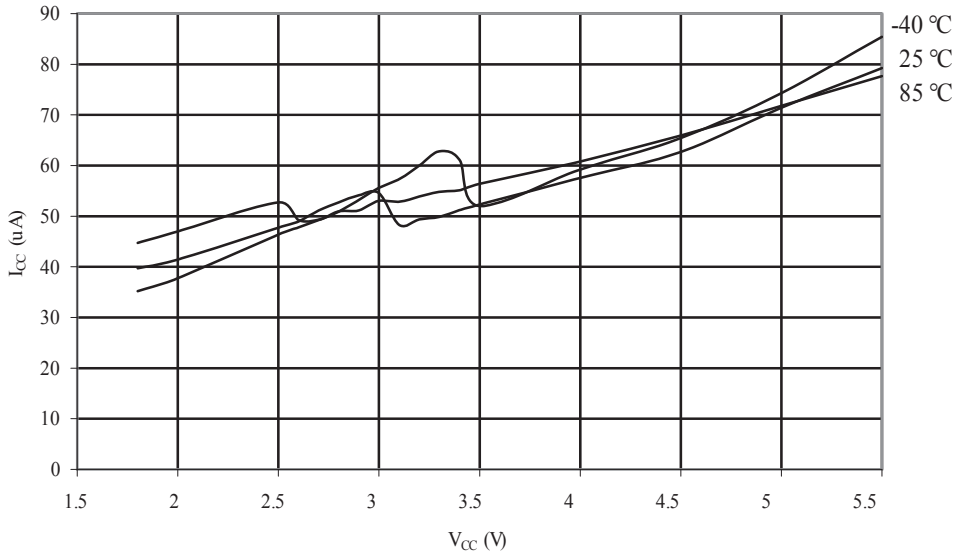



**Figure 34-89. ATmega88A: AREF External Reference Current vs. $V_{CC}$**

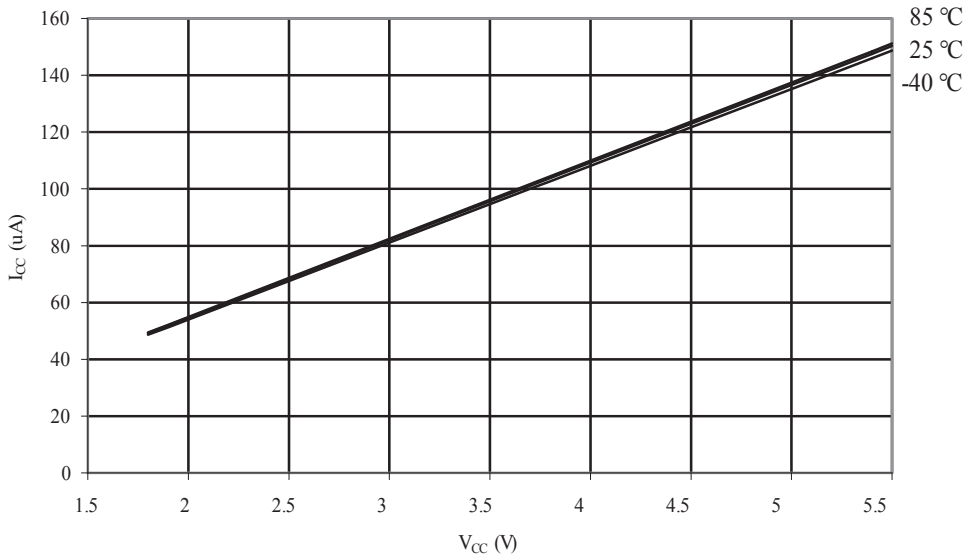


**Figure 34-90. ATmega88A: Brownout Detector Current vs. $V_{CC}$**

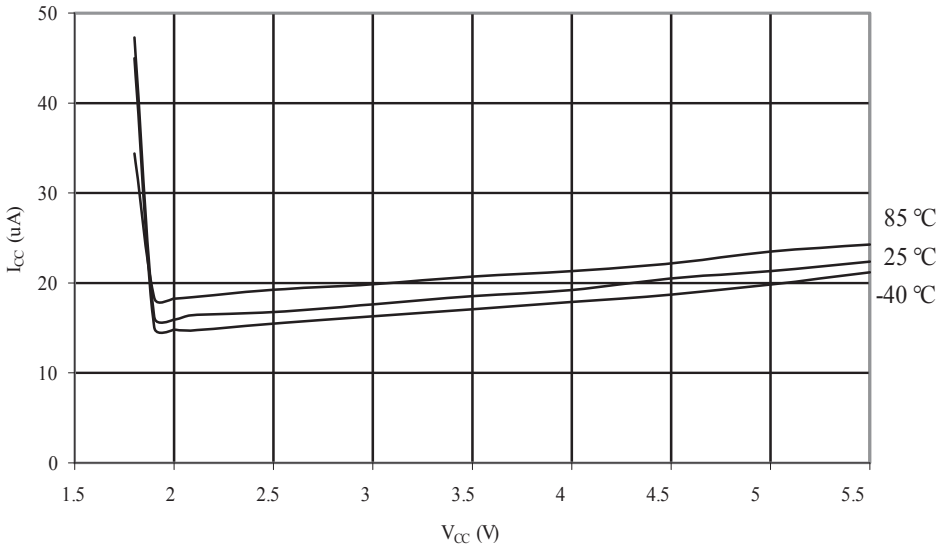
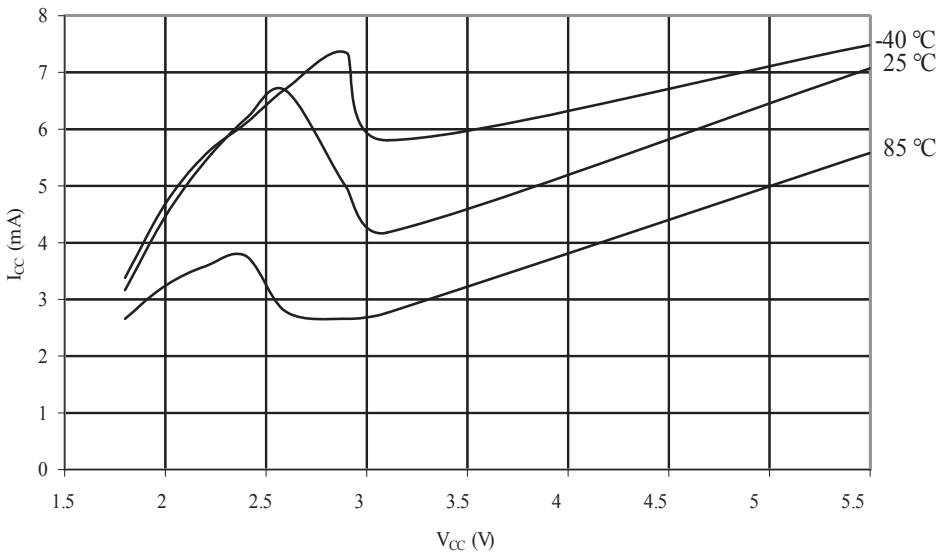



**Figure 34-91. ATmega88A: Programming Current vs. $V_{CC}$**



### 34.2.13. Current Consumption in Reset and Reset Pulsewidth

**Figure 34-92. ATmega88A: Reset Supply Current vs. Low Frequency (0.1 - 1.0MHz)**

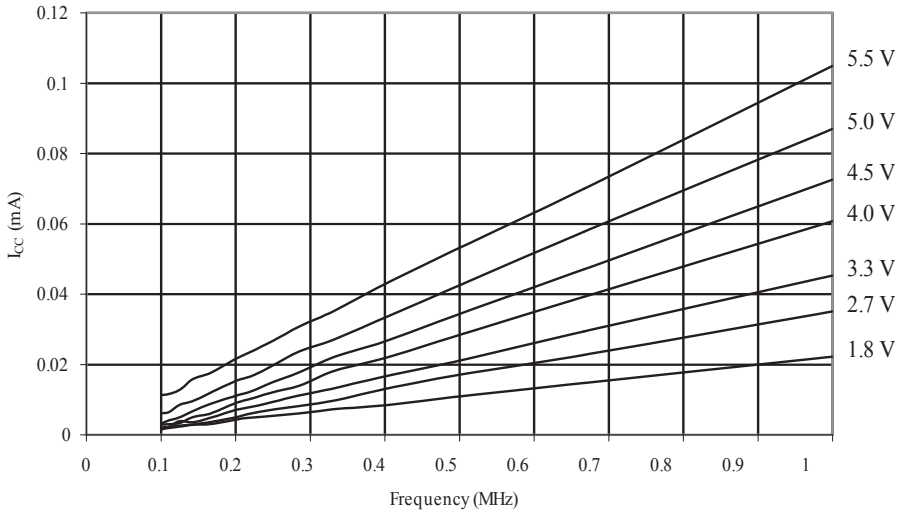



**Figure 34-93. ATmega88A: Reset Supply Current vs. Frequency (1 - 20MHz)**

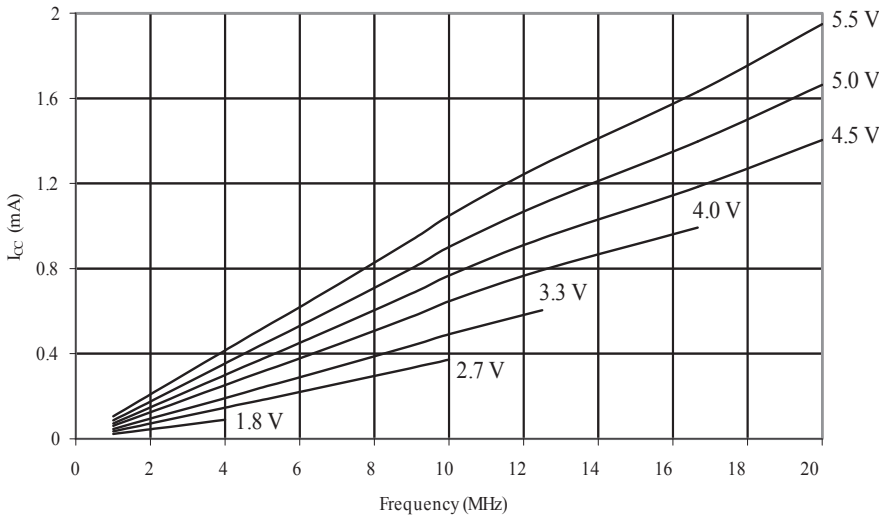
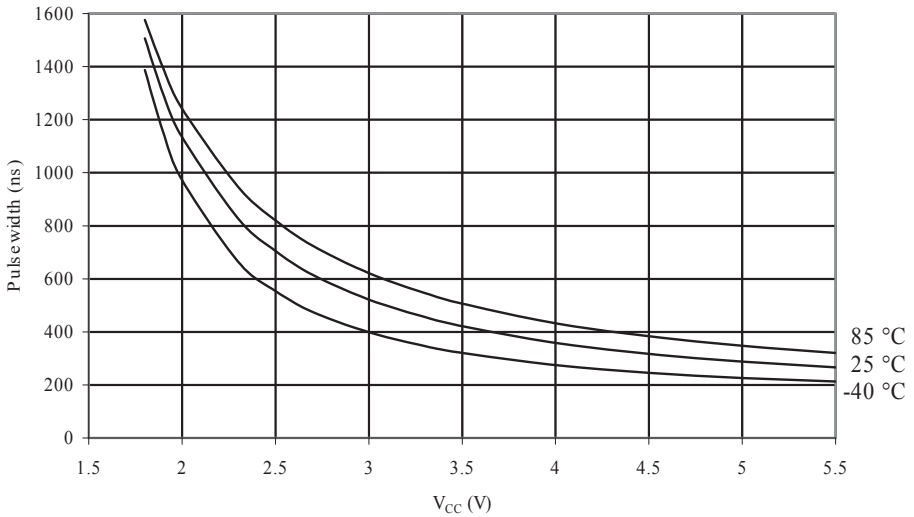


**Figure 34-94. ATmega88A: : Minimum Reset Pulse width vs. $V_{CC}$**




## 34.3. ATmega168A Typical Characteristics

### 34.3.1. Active Supply Current

**Figure 34-95. ATmega168A: Active Supply Current vs. Low Frequency (0.1MHz - 1.0MHz)**

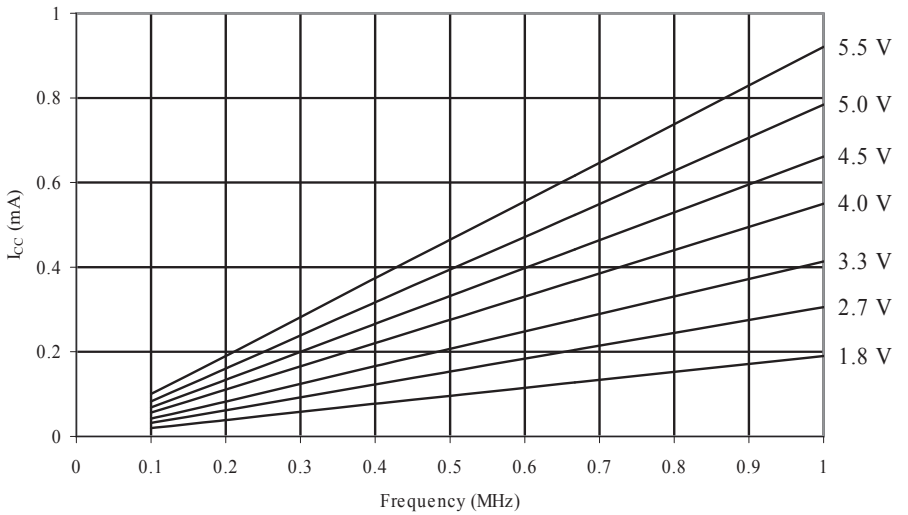


**Figure 34-96. ATmega168A: Active Supply Current vs. Frequency (1MHz - 20MHz)**

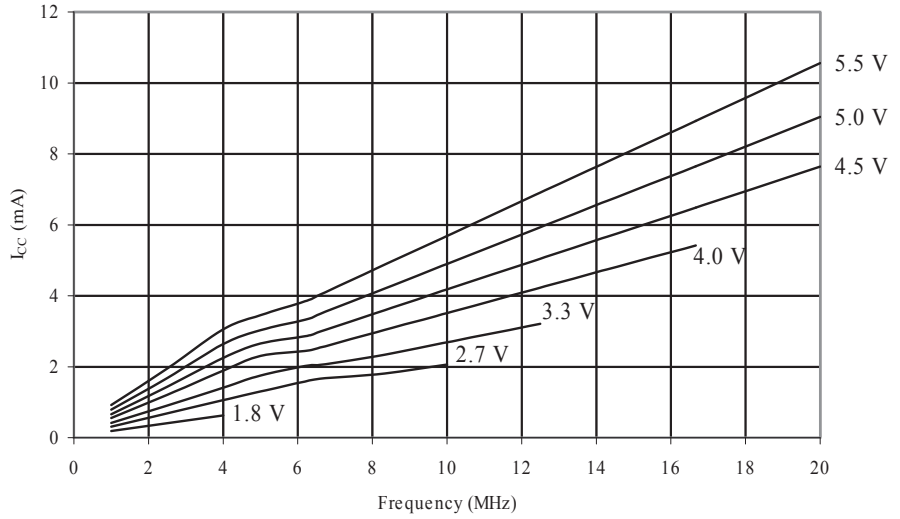



**Figure 34-97. ATmega168A: Active Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 128kHz)**

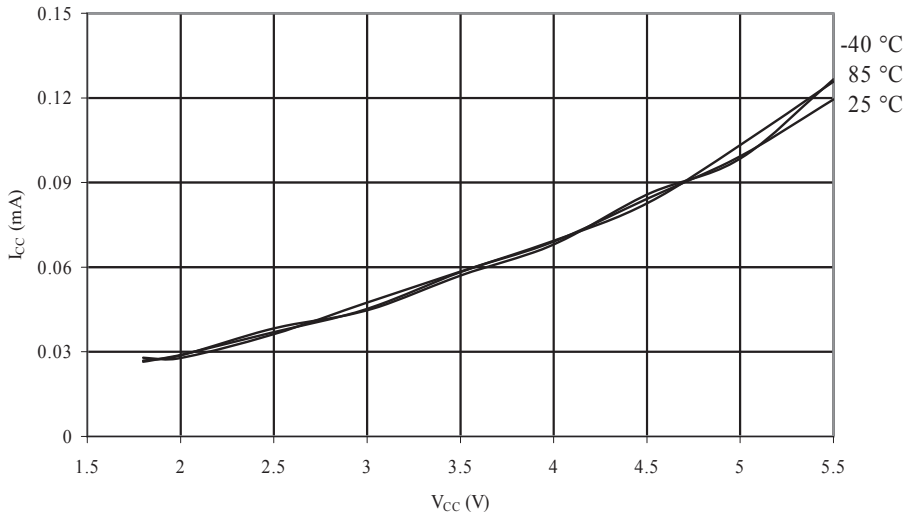


**Figure 34-98. ATmega168A: Active Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 1MHz)**

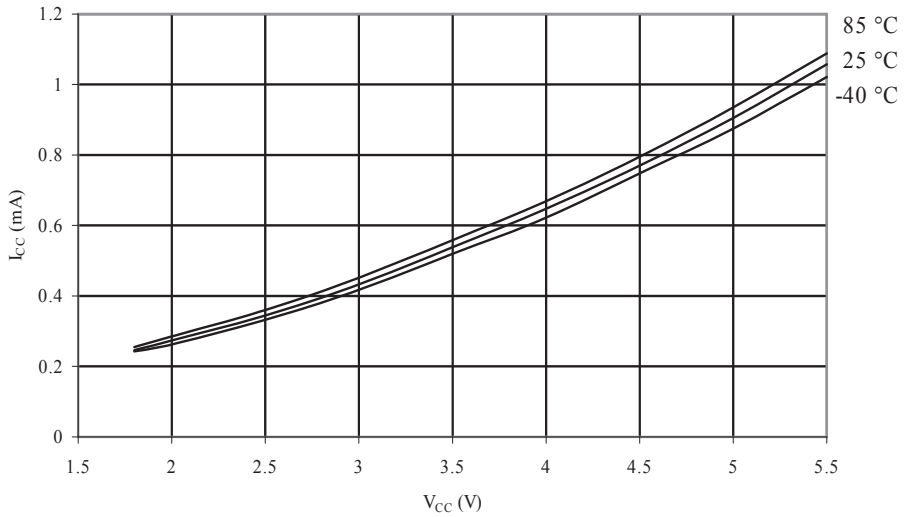
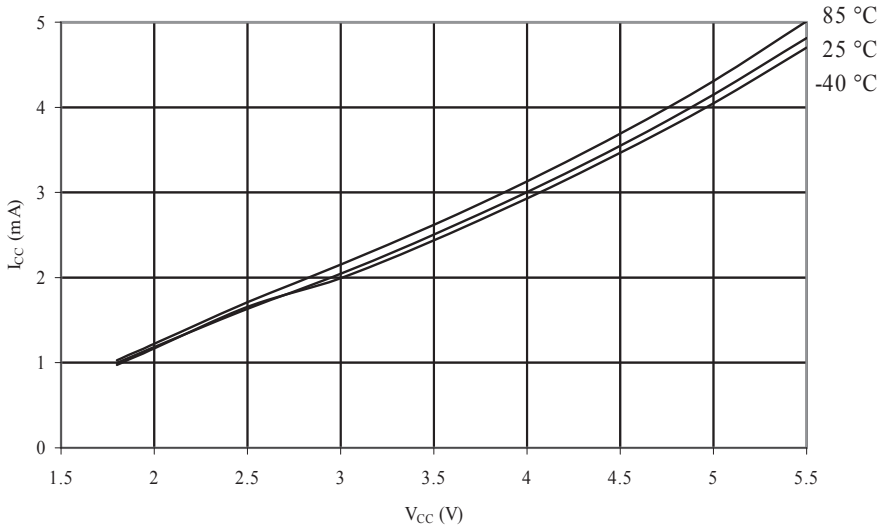



**Figure 34-99. ATmega168A: Active Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 8MHz)**



### 34.3.2. Idle Supply Current

**Figure 34-100. ATmega168A: Idle Supply Current vs. Low Frequency (0.1MHz - 1.0MHz)**

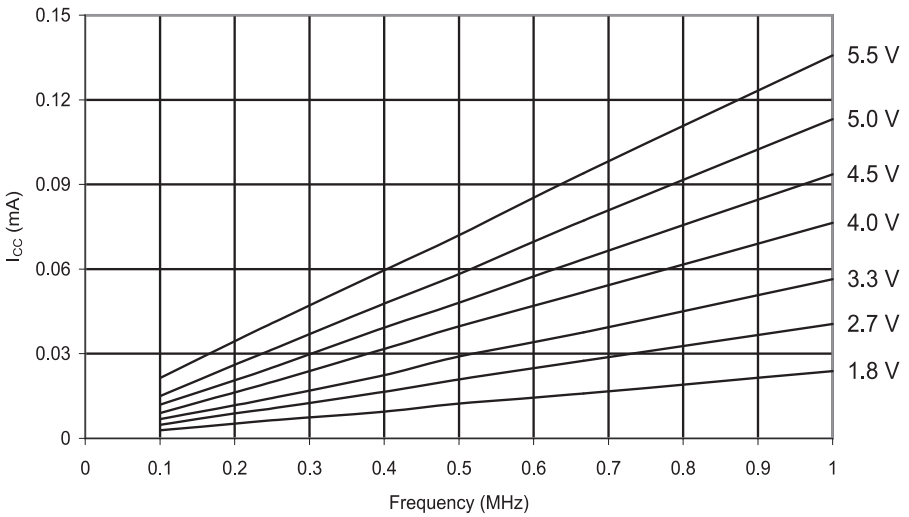



**Figure 34-101. ATmega168A: Idle Supply Current vs. Frequency (1MHz - 20MHz)**

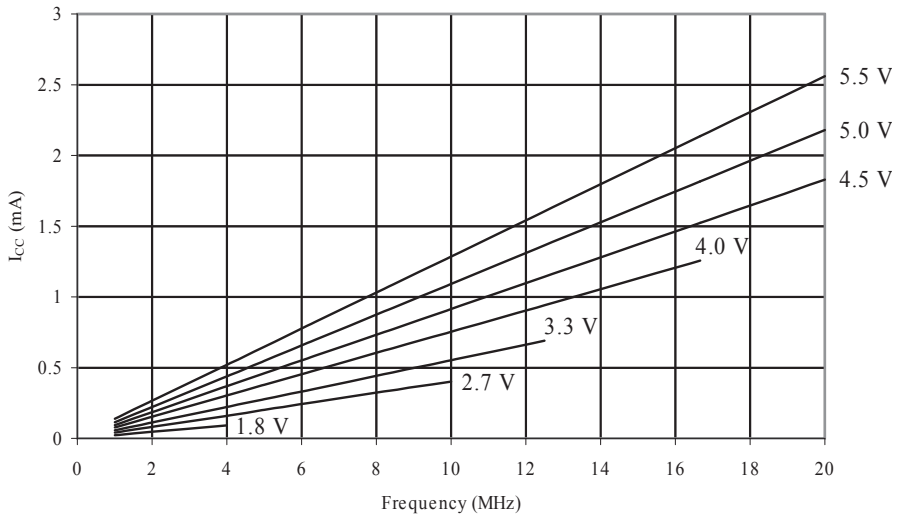


**Figure 34-102. ATmega168A: Idle Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 128kHz)**

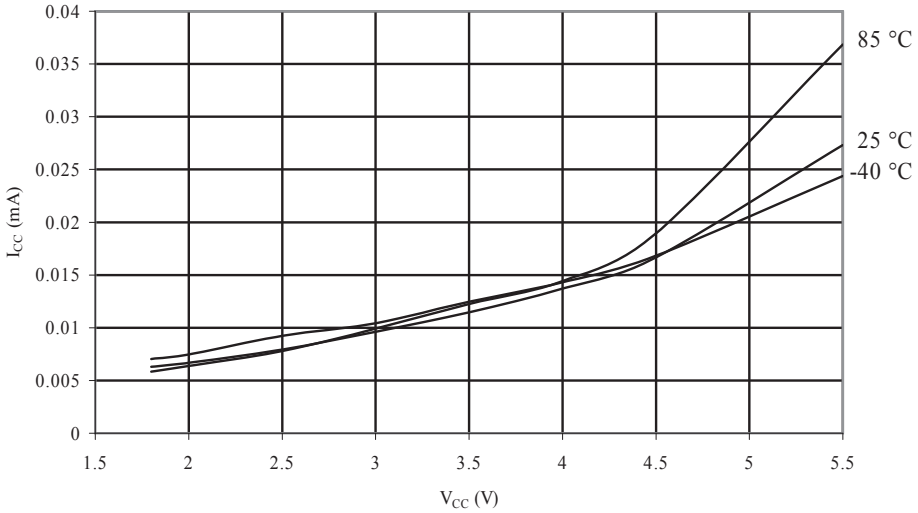



**Figure 34-103. ATmega168A: Idle Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 1MHz)**

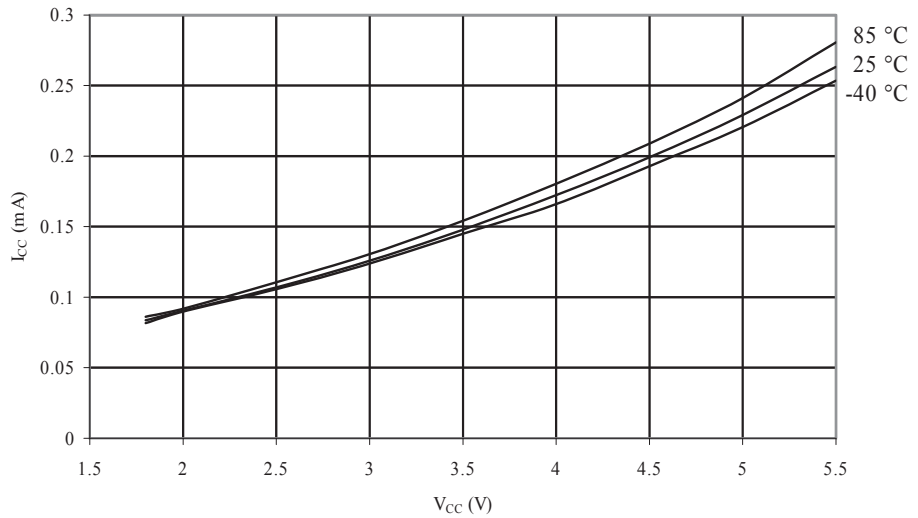
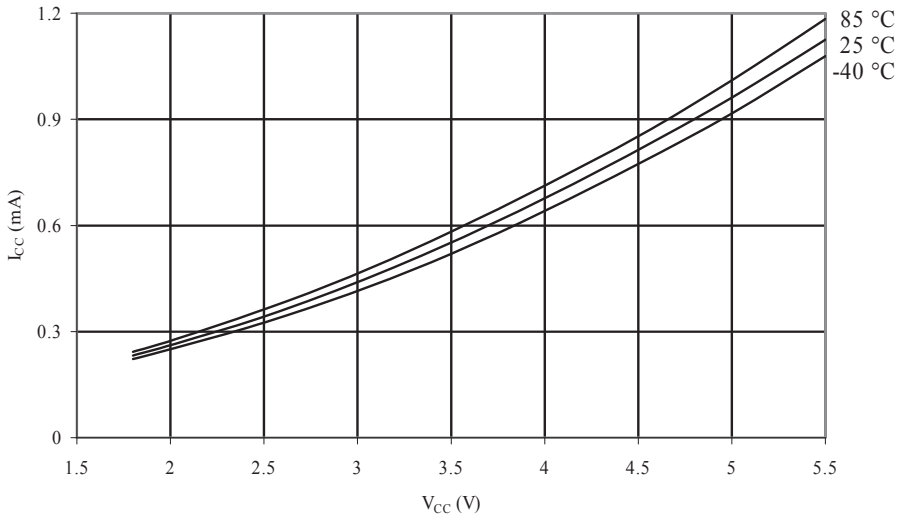


**Figure 34-104. ATmega168A: Idle Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 8MHz)**




### 34.3.3. Supply Current of IO Modules

The tables and formulas below can be used to calculate the additional current consumption for the different I/O modules in Active and Idle mode. The enabling or disabling of the I/O modules are controlled by the Power Reduction Register. See “Power Reduction Register” for details.

**Table 34-5. ATmega168A: Additional Current Consumption for the different I/O modules (absolute values)**

| PRR bit | Typical numbers (µA) | | |
|---|---|---|---|
| | $V_{CC}$ = 2V, F = 1MHz | $V_{CC}$ = 3V, F = 4MHz | $V_{CC}$ = 5V, F = 8MHz |
| PRUSART0 | 2.86 | 20.3 | 52.2 |
| PRTWI | 6.00 | 44.1 | 122.0 |
| PRTIM2 | 4.97 | 33.2 | 79.8 |
| PRTIM1 | 3.50 | 23.0 | 55.3 |
| PRTIM0 | 1.43 | 9.2 | 21.4 |
| PRSPI | 5.01 | 38.6 | 111.4 |
| PRADC | 6.34 | 45.7 | 123.6 |

**Table 34-6. ATmega168A: Additional Current Consumption (percentage) in Active and Idle mode**

| PRR bit | Additional Current consumption compared to Active with external clock (Figure 34-95 and Figure 34-96) | Additional Current consumption compared to Idle with external clock (see Figure 34-100 and Figure 34-101) |
|---|---|---|
| PRUSART0 | 1.5% | 8.9% |
| PRTWI | 3.2% | 19.5% |
| PRTIM2 | 2.4% | 14.8% |
| PRTIM1 | 1.7% | 10.3% |
| PRTIM0 | 0.7% | 4.1% |

| PRR bit | Additional Current consumption compared to Active with external clock (Figure 34-95 and Figure 34-96) | Additional Current consumption compared to Idle with external clock (see Figure 34-100 and Figure 34-101) |
|---|---|---|
| PRSPI | 2.9% | 17.1% |
| PRADC | 3.4% | 20.3% |

It is possible to calculate the typical current consumption based on the numbers from the above table for other $V_{CC}$ and frequency settings.

**Related Links**

PRR on page 73

#### 34.3.3.1. Example

Calculate the expected current consumption in idle mode with TIMER1, ADC, and SPI enabled at $V_{CC}$ = 2.0V and F = 1MHz. From Table *Additional Current Consumption (percentage) in Active and Idle mode* in the previous section, third column, we see that we need to add 10.3% for the TIMER1, 20.3% for the ADC, and 17.1% for the SPI module. Reading from Figure *Idle Supply Current vs. Low Frequency (0.1-1.0MHz)*, we find that the idle current consumption is ~0.027mA at $V_{CC}$ = 2.0V and F = 1MHz. The total current consumption in idle mode with TIMER1, ADC, and SPI enabled, gives:

$$I_{CCtotal} \simeq 0.027\ \text{mA} \cdot (1 + 0.103 + 0.203 + 0.171) \simeq 0.040\ \text{mA}$$

### 34.3.4. Power-down Supply Current

**Figure 34-105. ATmega168A: Power-Down Supply Current vs. $V_{CC}$ (Watchdog Timer Disabled)**

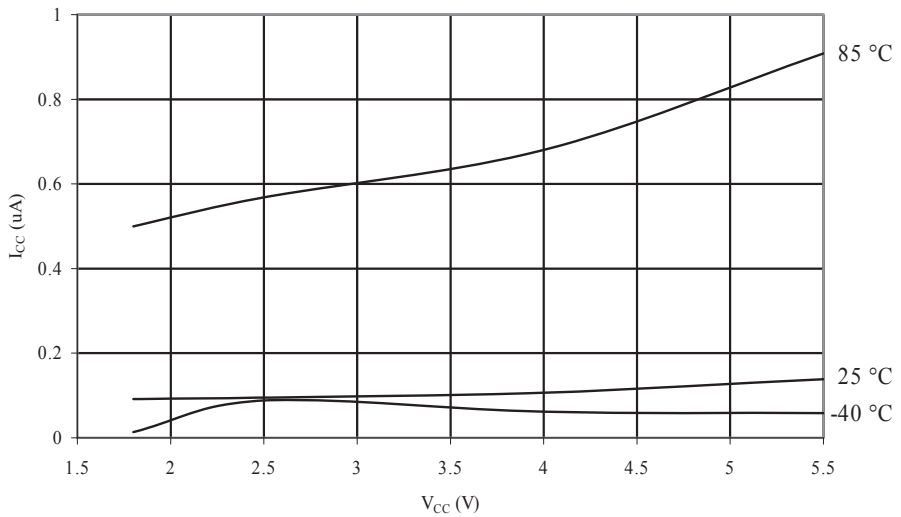
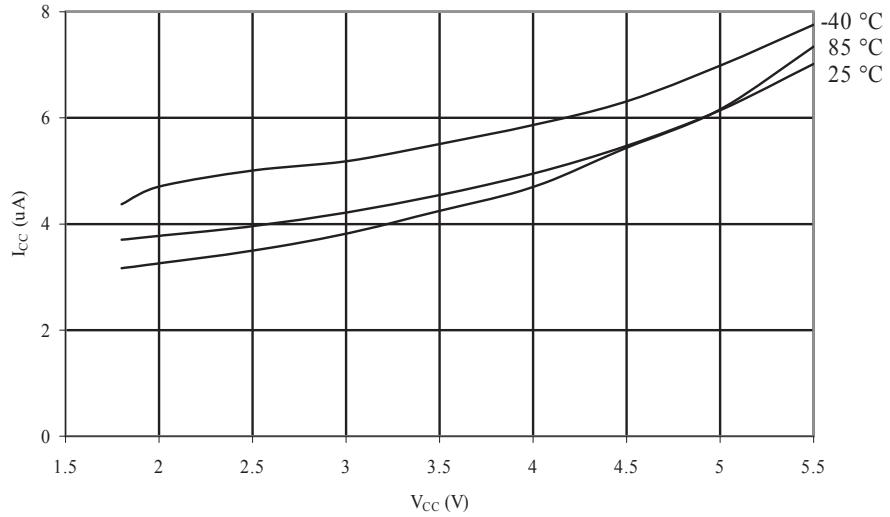


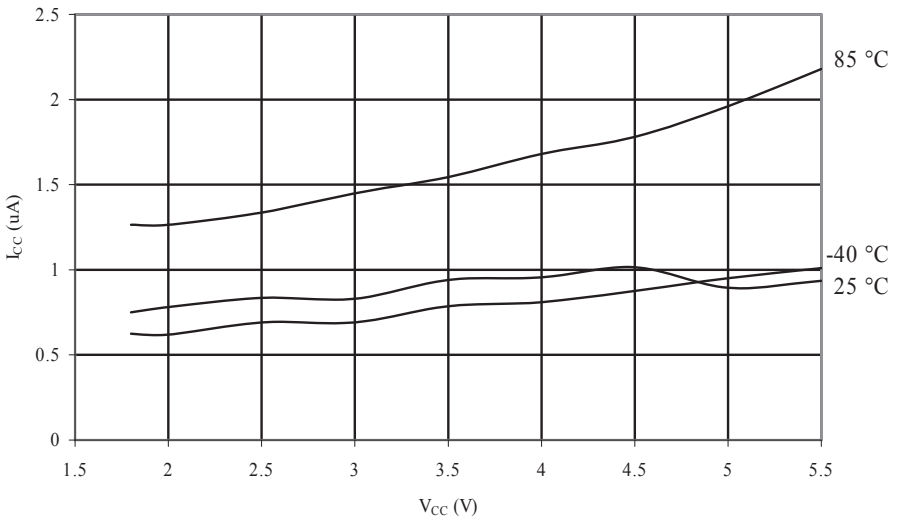
**Figure 34-106. ATmega168A: Power-Down Supply Current vs. $V_{CC}$ (Watchdog Timer Enabled)**

cc (Watchdog Timer Enabled)



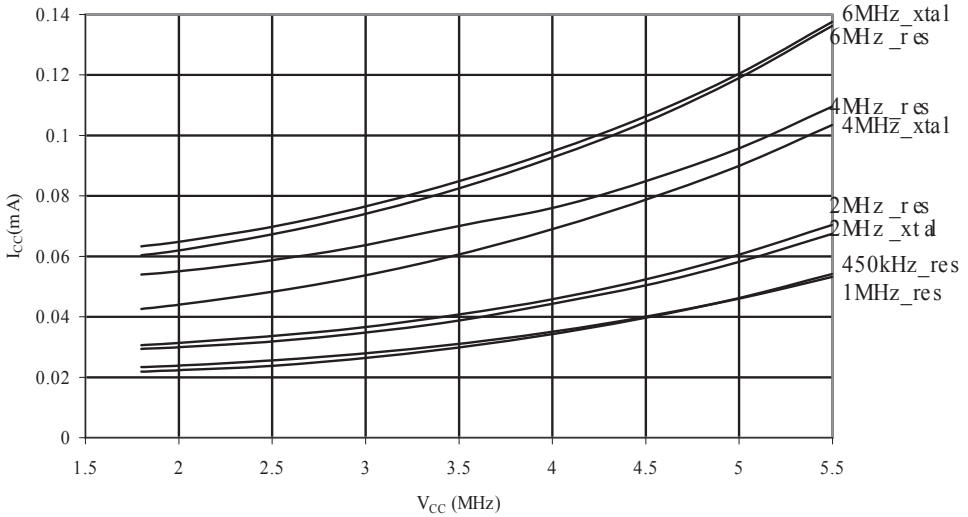

### 34.3.5. Power-save Supply Current

**Figure 34-107. ATmega168A: Power-Save Supply Current vs. $V_{CC}$ (Watchdog Timer Disabled and 32kHz Crystal Oscillator Running)**



### 34.3.6. Standby Supply Current

**Figure 34-108. ATmega168A: Standby Supply Current vs. $V_{cc}$ (Watchdog Timer Disabled)**




### 34.3.7. Pin Pull-Up

**Figure 34-109. ATmega168A: I/O Pin Pull-up Resistor Current vs. Input Voltage ($V_{CC}$ = 1.8V)**

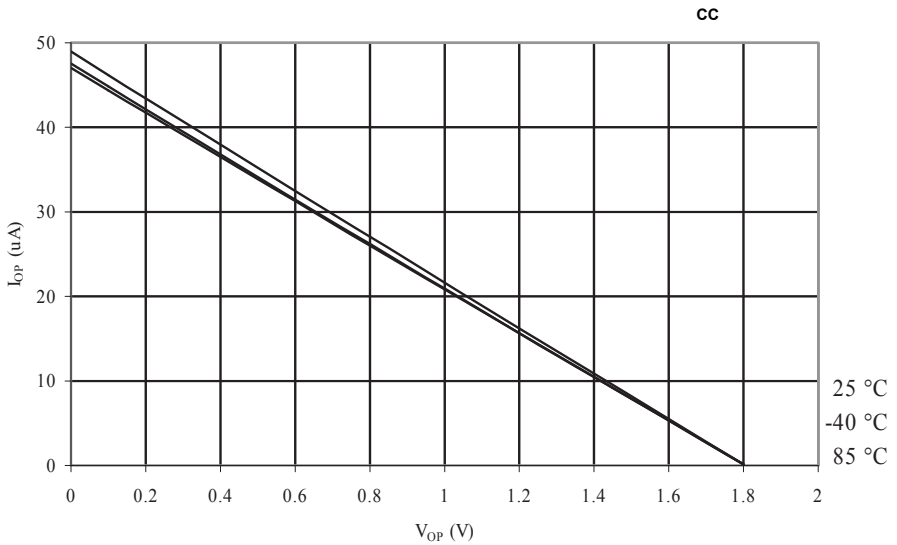


**Figure 34-110. ATmega168A: I/O Pin Pull-up Resistor Current vs. Input Voltage ($V_{CC}$ = 2.7V)**

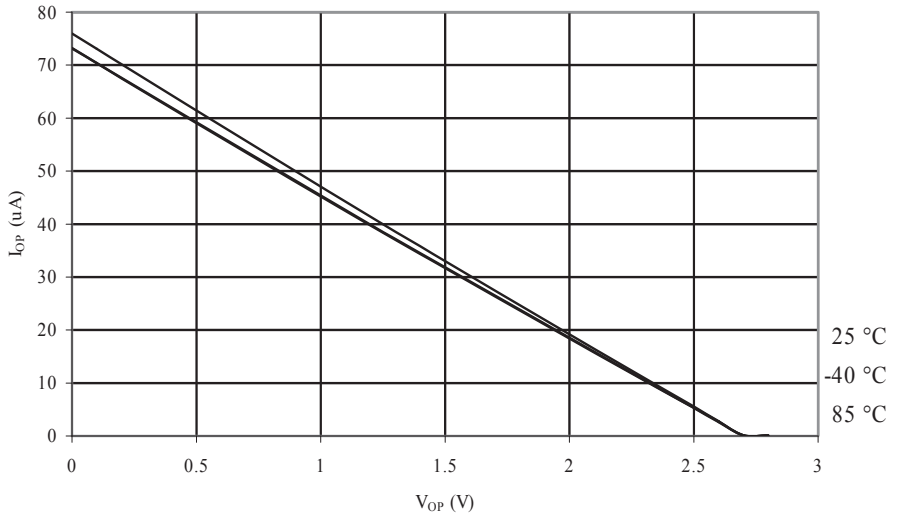



**Figure 34-111. ATmega168A: I/O Pin Pull-up Resistor Current vs. Input Voltage ($V_{CC}$ = 5V)**

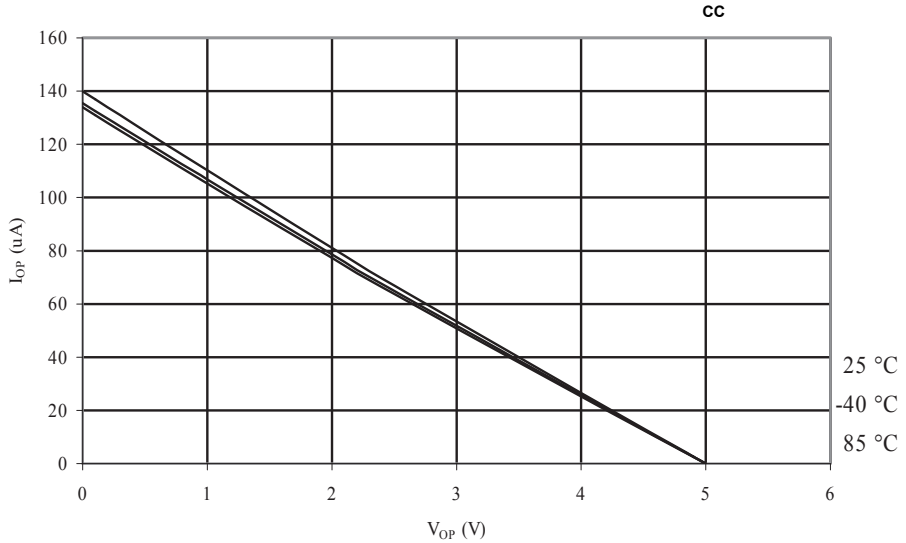



**Figure 34-112. ATmega168A: Reset Pull-up Resistor Current vs. Reset Pin Voltage ($V_{CC}$ = 1.8V)**

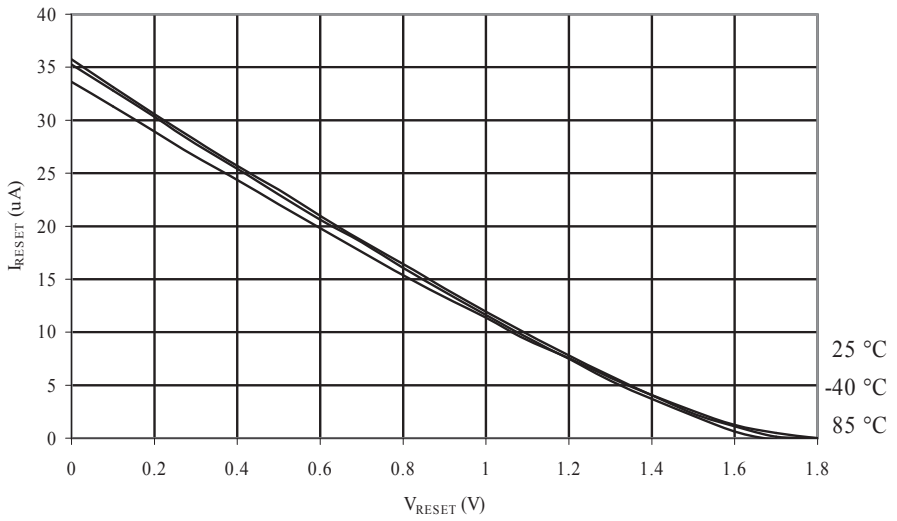


**Figure 34-113. ATmega168A: Reset Pull-up Resistor Current vs. Reset Pin Voltage ($V_{CC}$ = 2.7V)**

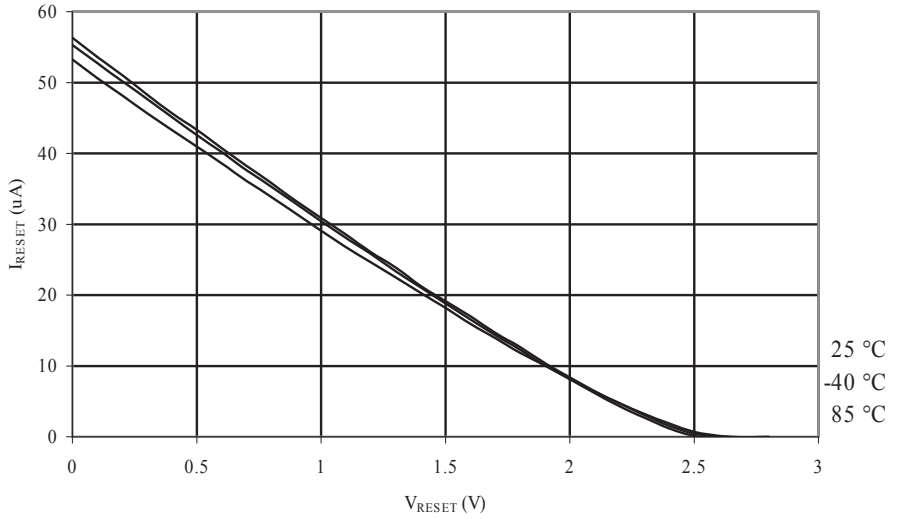
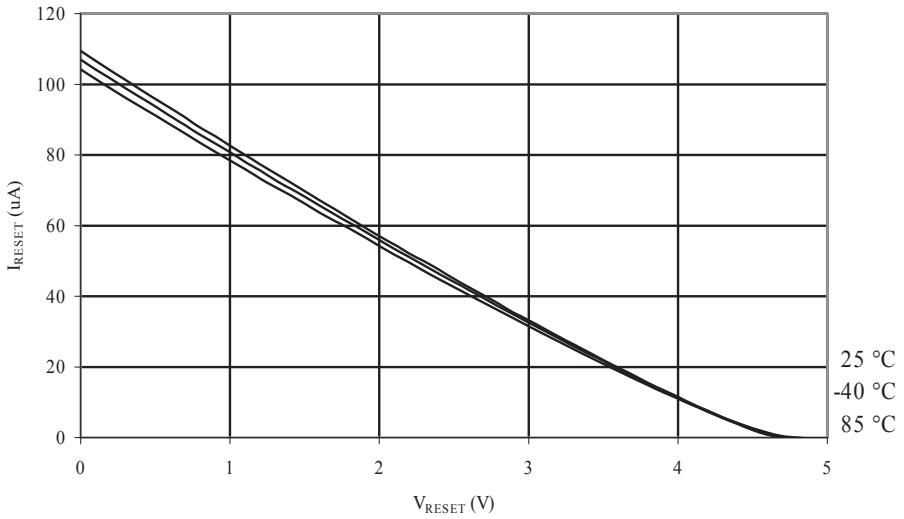



**Figure 34-114. ATmega168A: Reset Pull-up Resistor Current vs. Reset Pin Voltage ($V_{CC}$ = 5V)**



### 34.3.8. Pin Driver Strength

**Figure 34-115. ATmega168A: I/O Pin Output Voltage vs. Sink Current ($V_{CC}$ = 3V)**

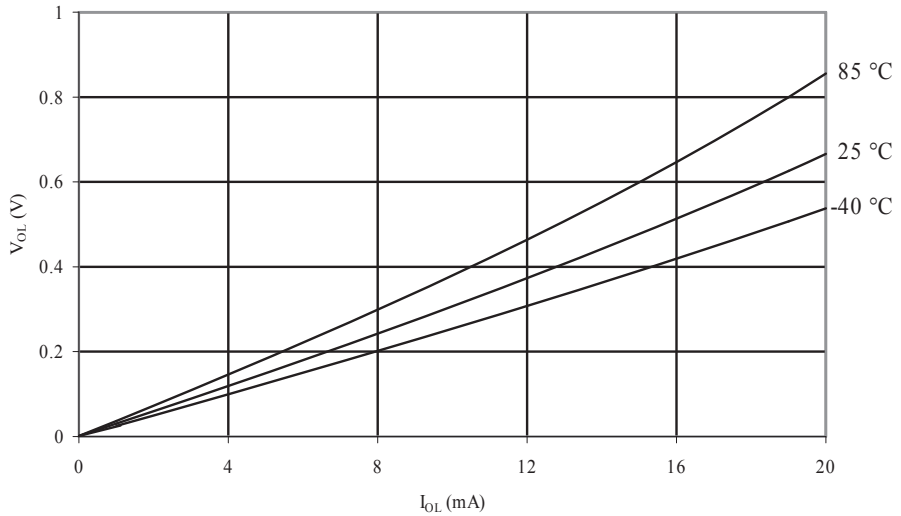



**Figure 34-116. ATmega168A: I/O Pin Output Voltage vs. Sink Current ($V_{CC}$ = 5V)**

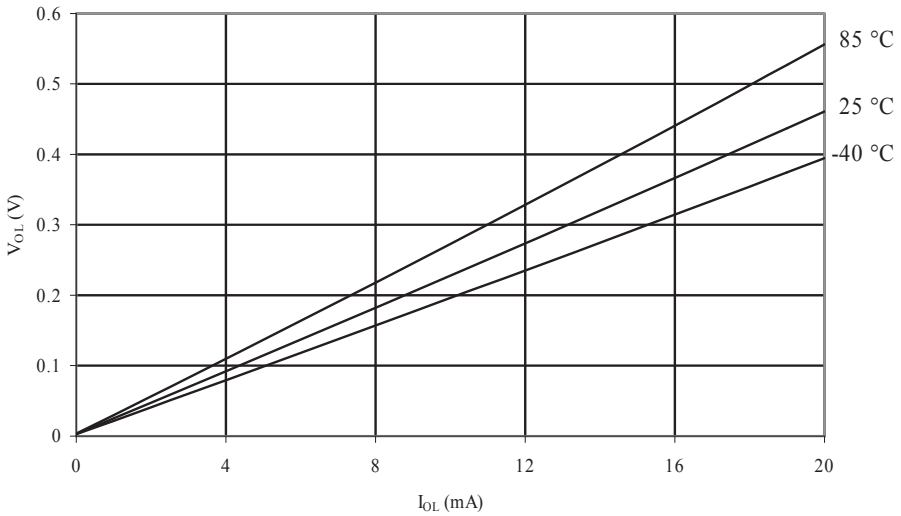


**Figure 34-117. ATmega168A: I/O Pin Output Voltage vs. Source Current ($V_{cc}$ = 3V)**

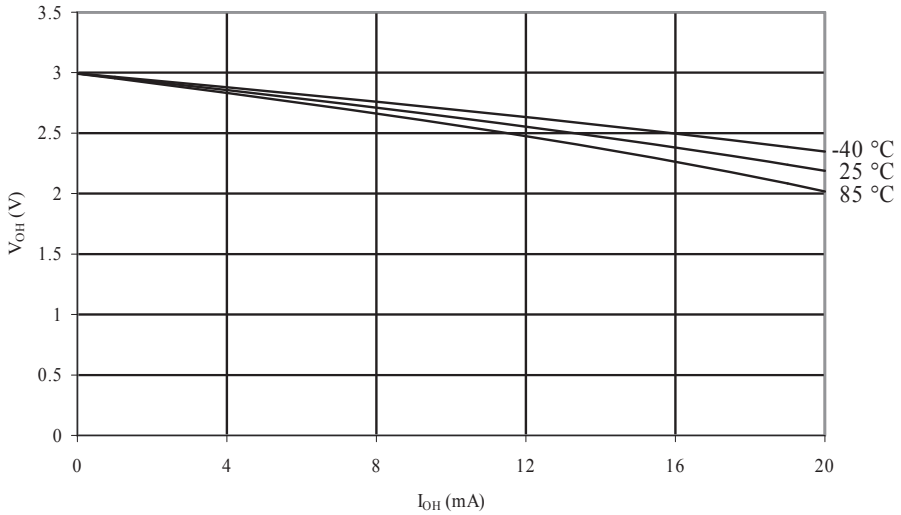
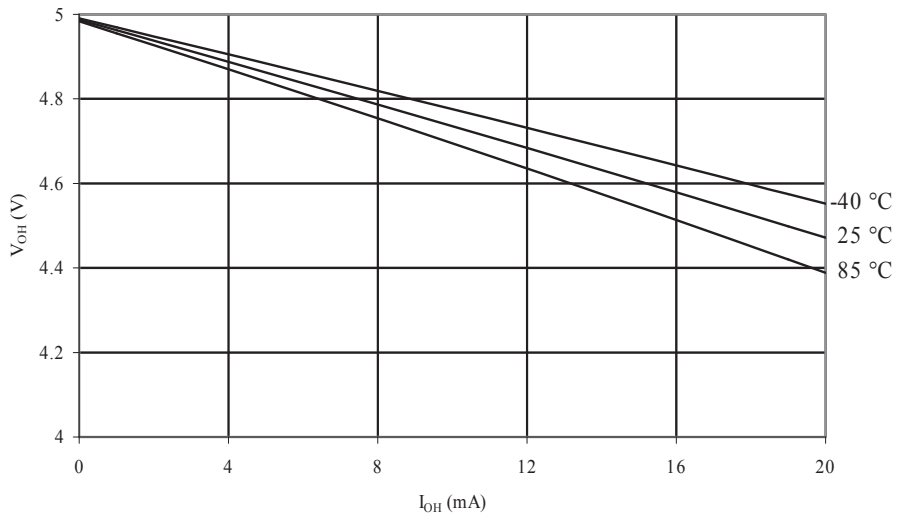



**Figure 34-118. ATmega168A: I/O Pin Output Voltage vs. Source Current ($V_{CC}$ = 5V)**



### 34.3.9. Pin Threshold and Hysteresis

**Figure 34-119. ATmega168A: I/O Pin Input Threshold Voltage vs. $V_{CC}$ ($V_{IH}$, I/O Pin read as '1')**

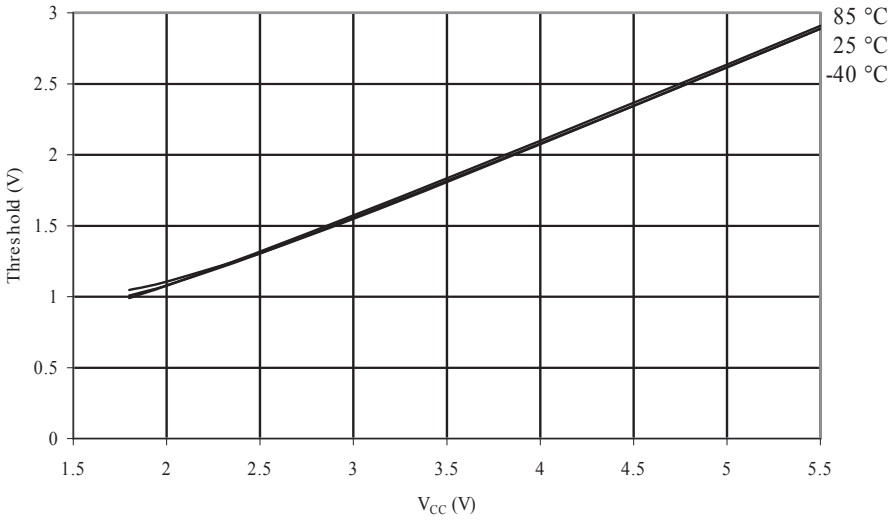



**Figure 34-120. ATmega168A: I/O Pin Input Threshold Voltage vs. $V_{CC}$ ($V_{IL}$, I/O Pin read as '0')**

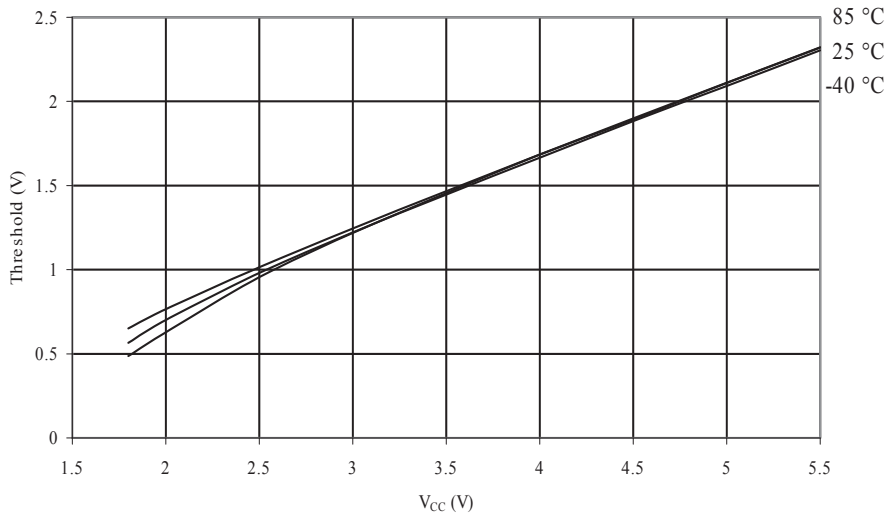


**Figure 34-121. ATmega168A: I/O Pin Input Hysteresis vs. $V_{CC}$**

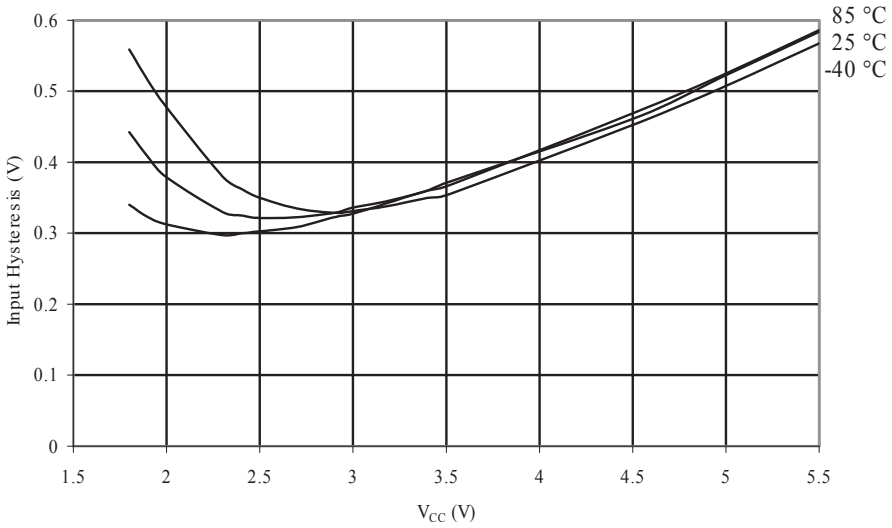



**Figure 34-122. ATmega168A: Reset Input Threshold Voltage vs. $V_{CC}$ ($V_{IH}$, I/O Pin read as '1')**

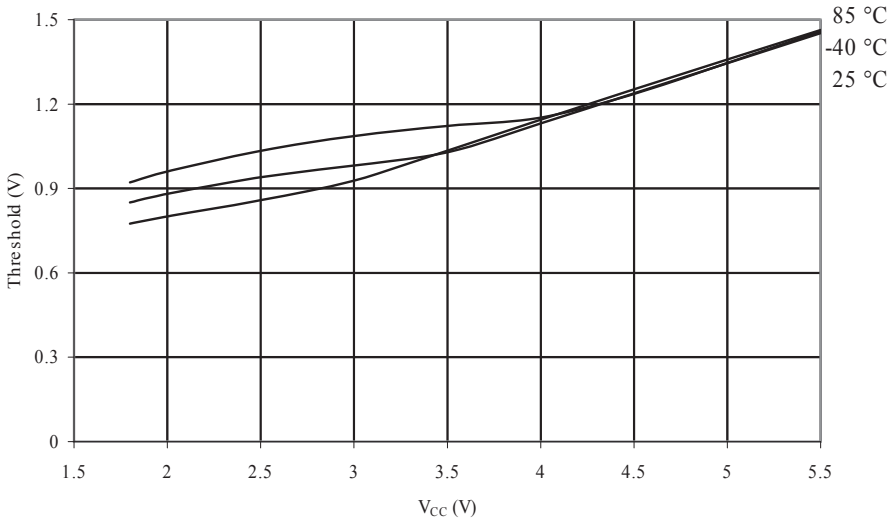



**Figure 34-123. ATmega168A: Reset Input Threshold Voltage vs. $V_{CC}$ ($V_{IL}$, I/O Pin read as '0')**

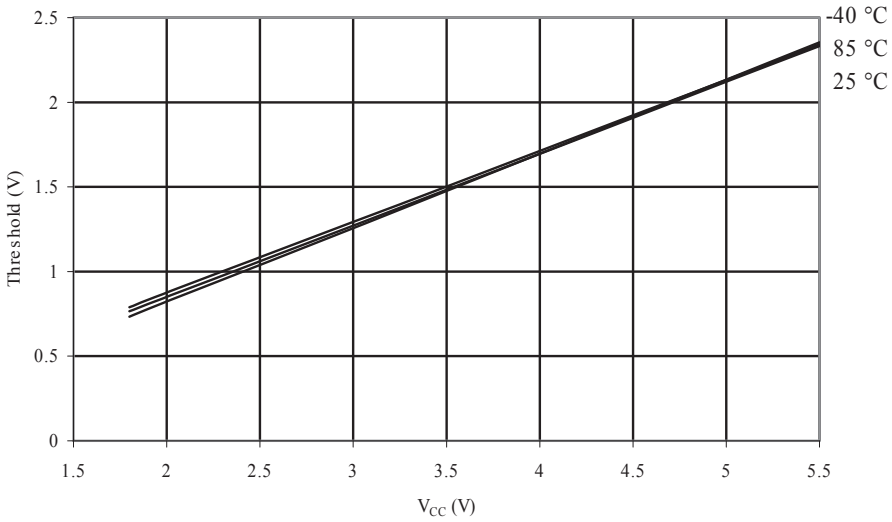
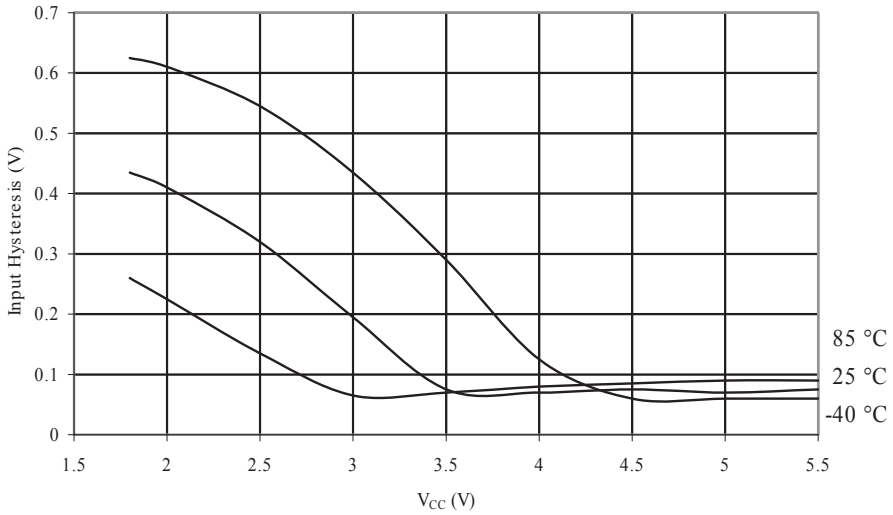


**Figure 34-124. ATmega168A: Reset Pin Input Hysteresis vs. $V_{CC}$**




### 34.3.10. BOD Threshold

**Figure 34-125. ATmega168A: BOD Thresholds vs. Temperature (BODLEVEL is 1.8V)**

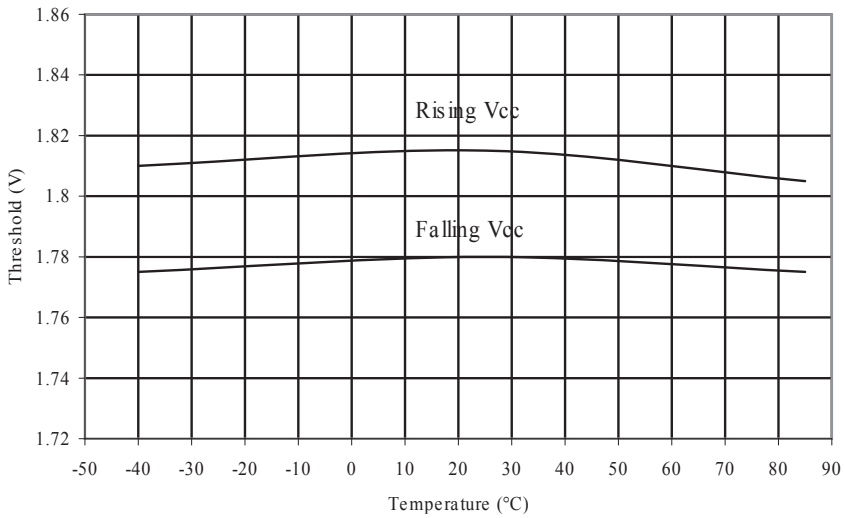


**Figure 34-126. ATmega168A: BOD Thresholds vs. Temperature (BODLEVEL is 2.7V)**

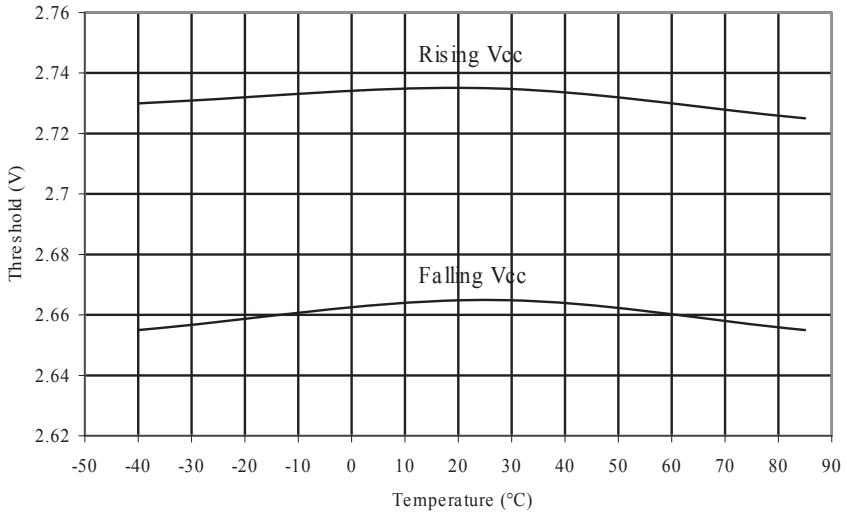



**Figure 34-127. ATmega168A: BOD Thresholds vs. Temperature (BODLEVEL is 4.3V)**

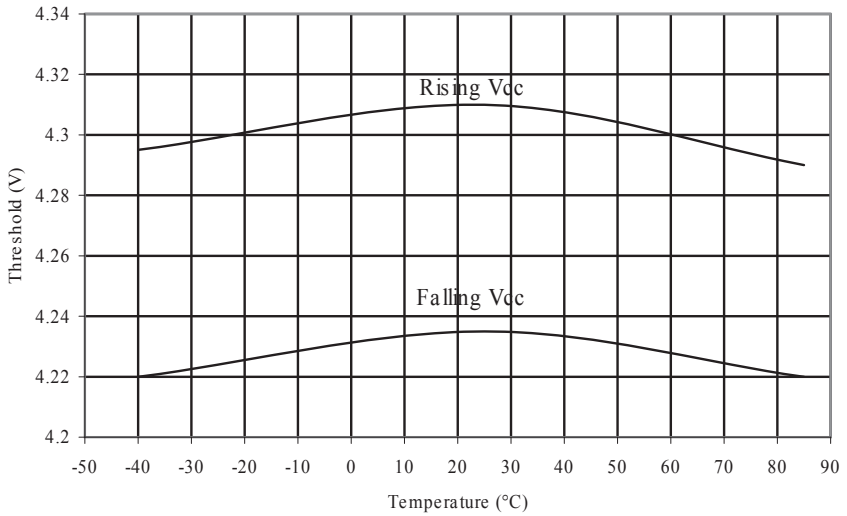
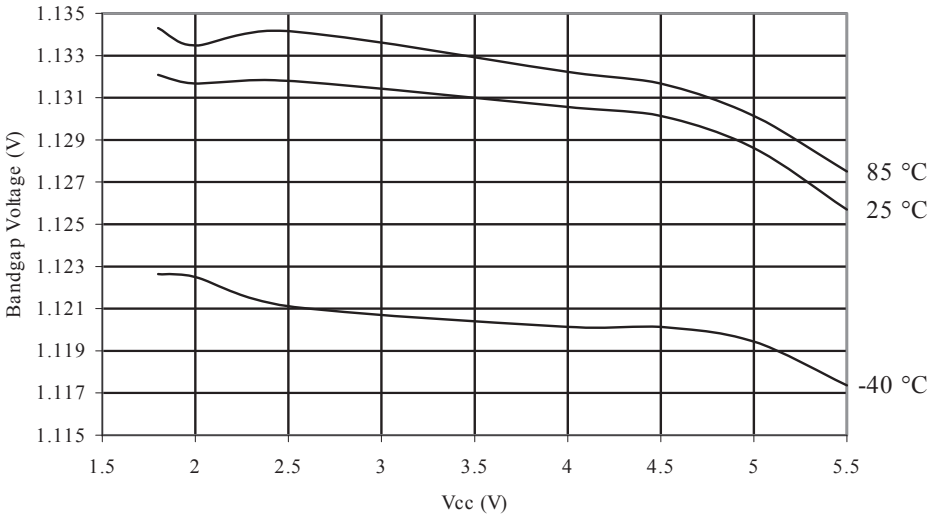


**Figure 34-128. ATmega168A: Bandgap Voltage vs. $V_{cc}$**




### 34.3.11. Internal Oscillator Speed

**Figure 34-129. ATmega168A: Watchdog Oscillator Frequency vs. Temperature**

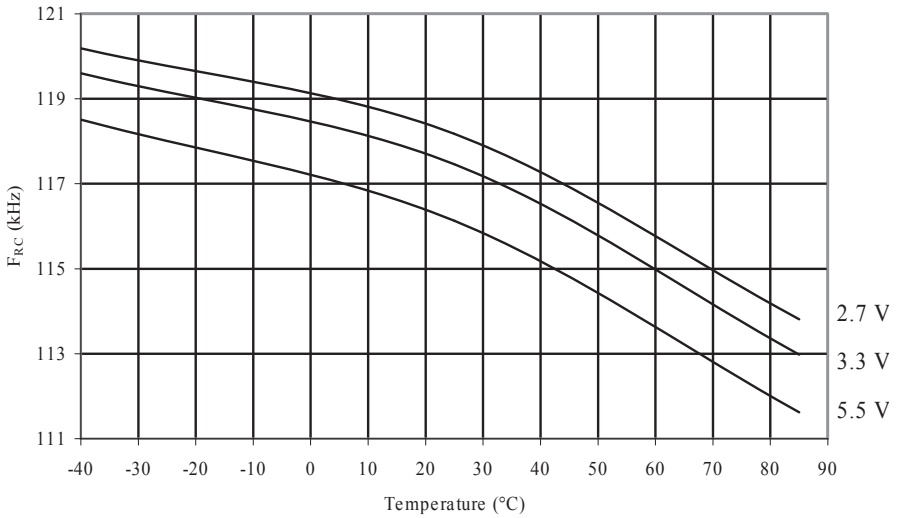


**Figure 34-130. ATmega168A: Watchdog Oscillator Frequency vs. $V_{CC}$**

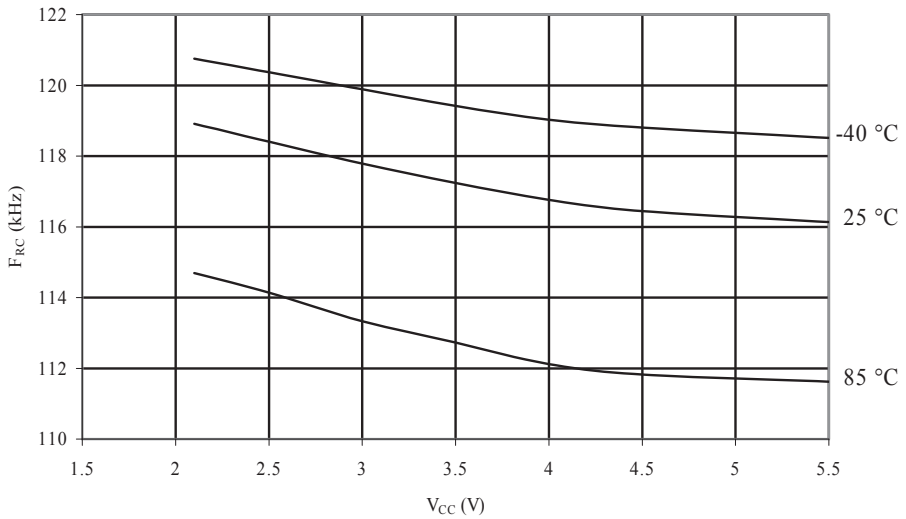



**Figure 34-131. ATmega168A: Calibrated 8 MHz RC Oscillator Frequency vs. $V_{CC}$**

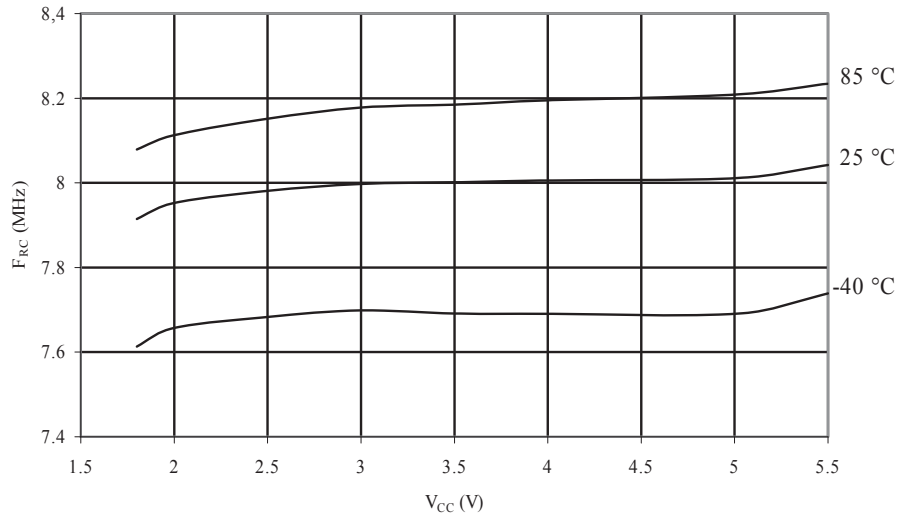



**Figure 34-132. ATmega168A: Calibrated 8MHz RC Oscillator Frequency vs. Temperature**

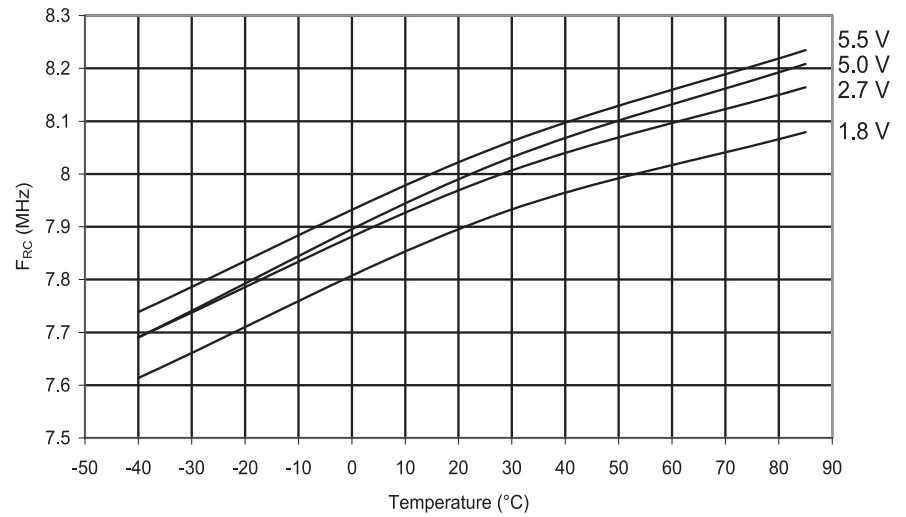
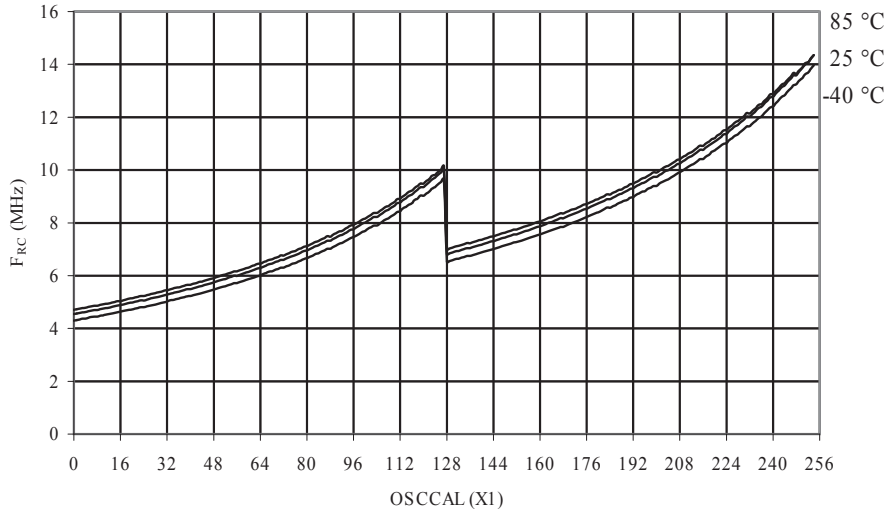


**Figure 34-133. ATmega168A: Calibrated 8MHz RC Oscillator Frequency vs. OSCCAL Value**




### 34.3.12. Current Consumption of Peripheral Units

**Figure 34-134. ATmega168A: ADC Current vs. $V_{CC}$ ($AREF = AV_{CC}$)**

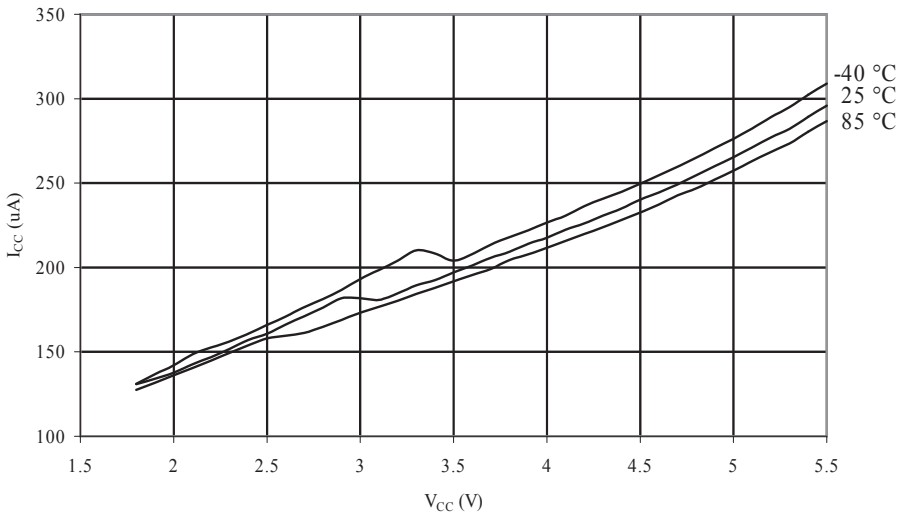


**Figure 34-135. ATmega168A: Analog Comparator Current vs. $V_{CC}$**

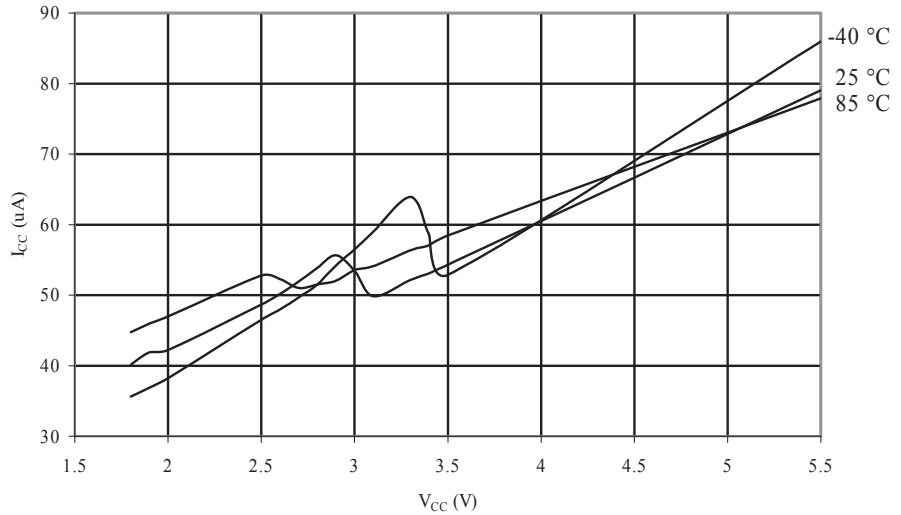



**Figure 34-136. ATmega168A: AREF External Reference Current vs. $V_{CC}$**

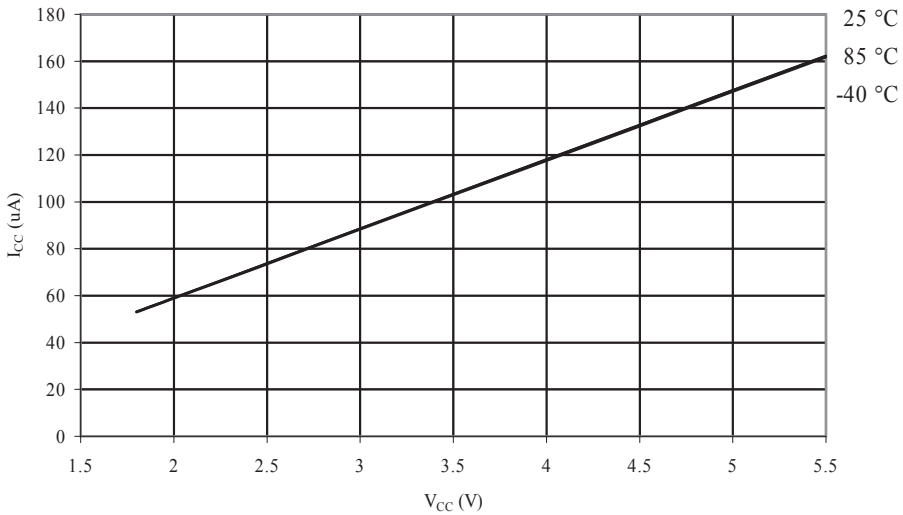



**Figure 34-137. ATmega168A: Brownout Detector Current vs. $V_{CC}$**

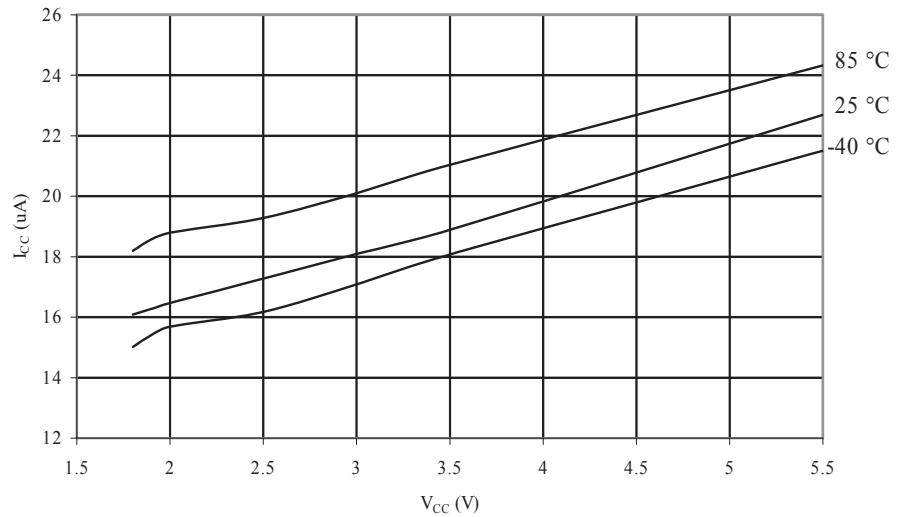
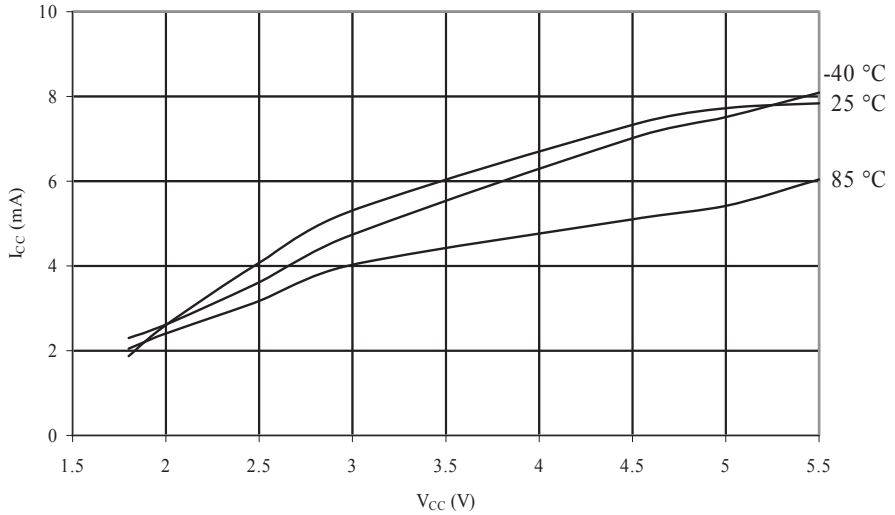


**Figure 34-138. ATmega168A: Programming Current vs. $V_{CC}$**




### 34.3.13. Current Consumption in Reset and Reset Pulsewidth

**Figure 34-139. ATmega168A: Reset Supply Current vs. Low Frequency (0.1MHz - 1.0MHz)**

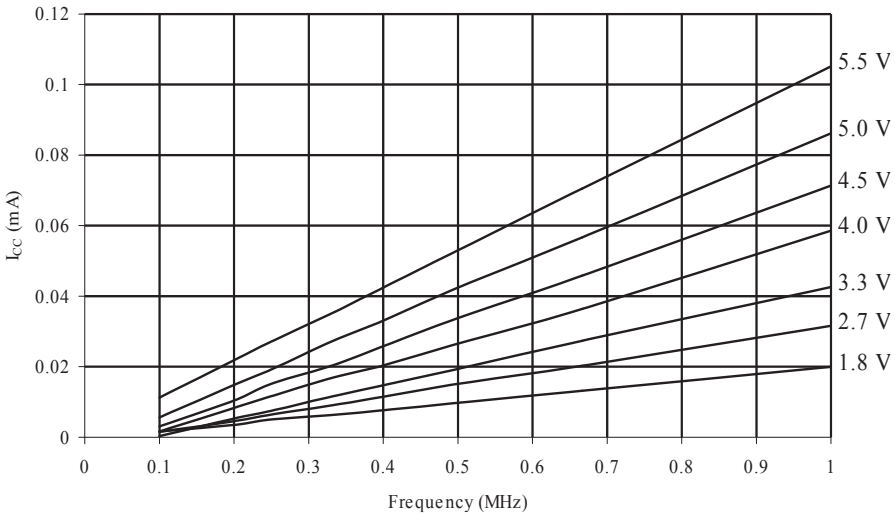


**Figure 34-140. ATmega168A: Reset Supply Current vs. Frequency (1MHz - 20MHz)**

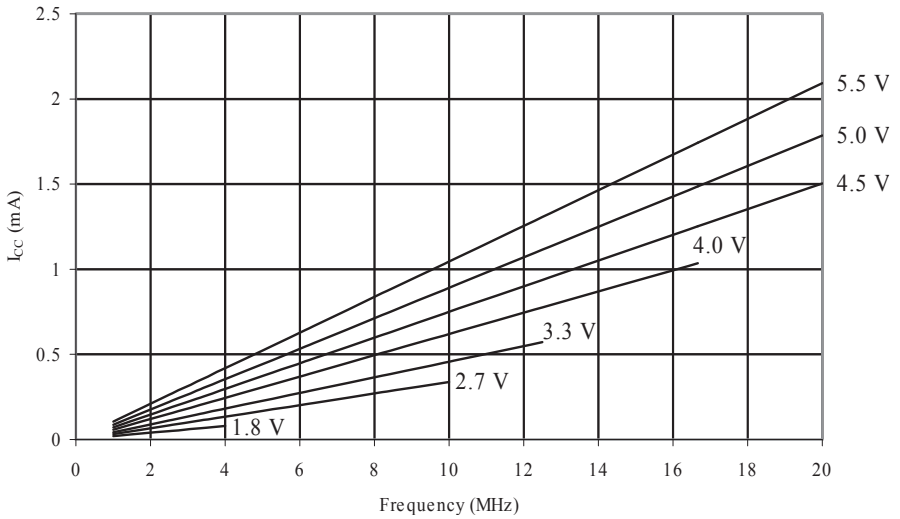
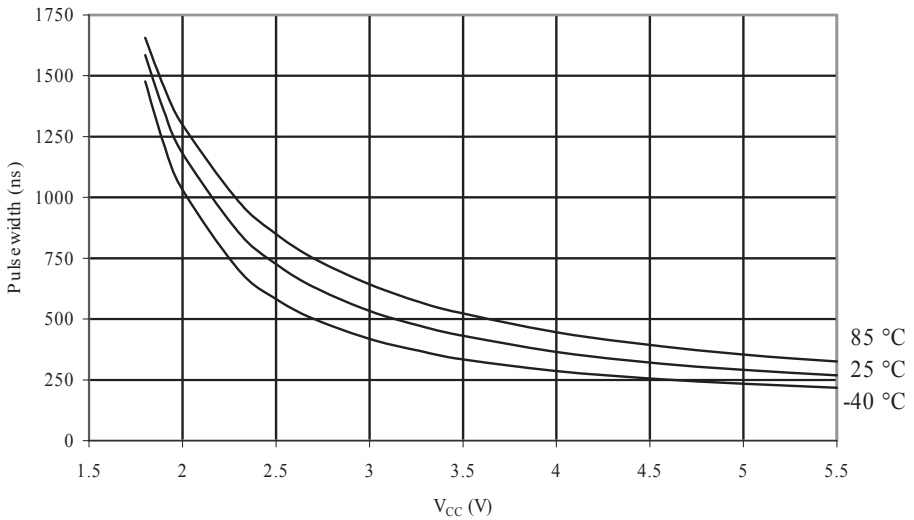



**Figure 34-141. ATmega168A: Minimum Reset Pulse Width vs. $V_{cc}$**



# 35. Register Summary

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x23 | PINB | 7:0 | PINB7 | PINB6 | PINB5 | PINB4 | PINB3 | PINB2 | PINB1 | PINB0 |
| 0x24 | DDRB | 7:0 | DDRB7 | DDRB6 | DDRB5 | DDRB4 | DDRB3 | DDRB2 | DDRB1 | DDRB0 |
| 0x25 | PORTB | 7:0 | PORTB7 | PORTB6 | PORTB5 | PORTB4 | PORTB3 | PORTB2 | PORTB1 | PORTB0 |
| 0x26 | PINC | 7:0 | | PINC6 | PINC5 | PINC4 | PINC3 | PINC2 | PINC1 | PINC0 |
| 0x27 | DDRC | 7:0 | | DDRC6 | DDRC5 | DDRC4 | DDRC3 | DDRC2 | DDRC1 | DDRC0 |
| 0x28 | PORTC | 7:0 | | PORTC6 | PORTC5 | PORTC4 | PORTC3 | PORTC2 | PORTC1 | PORTC0 |
| 0x29 | PIND | 7:0 | PIND7 | PIND6 | PIND5 | PIND4 | PIND3 | PIND2 | PIND1 | PIND0 |
| 0x2A | DDRD | 7:0 | DDRD7 | DDRD6 | DDRD5 | DDRD4 | DDRD3 | DDRD2 | DDRD1 | DDRD0 |
| 0x2B | PORTD | 7:0 | PORTD7 | PORTD6 | PORTD5 | PORTD4 | PORTD3 | PORTD2 | PORTD1 | PORTD0 |
| 0x2C ... 0x34 | Reserved | | | | | | | | | |
| 0x35 | TIFR0 | 7:0 | | | | | | OCFB | OCFA | TOV |
| 0x36 | TIFR1 | 7:0 | | | ICF | | | OCFB | OCFA | TOV |
| 0x37 | TIFR2 | 7:0 | | | | | | OCFB | OCFA | TOV |
| 0x38 ... 0x3A | Reserved | | | | | | | | | |
| 0x3B | PCIFR | 7:0 | | | | | | PCIF2 | PCIF1 | PCIF0 |
| 0x3C | EIFR | 7:0 | | | | | | | INTF1 | INTF0 |
| 0x3D | EIMSK | 7:0 | | | | | | | INT1 | INT0 |
| 0x3E | GPIOR0 | 7:0 | GPIOR0[7:0] | | | | | | | |
| 0x3F | EECR | 7:0 | | | EEPM1 | EEPM0 | EERIE | EEMPE | EEPE | EERE |
| 0x40 | EEDR | 7:0 | EEDR[7:0] | | | | | | | |
| 0x41 | EEARL | 7:0 | EEAR7 | EEAR6 | EEAR5 | EEAR4 | EEAR3 | EEAR2 | EEAR1 | EEAR0 |
| 0x42 | EEARH | 7:0 | | | | | | | EEAR9 | EEAR8 |
| 0x43 | GTCCR | 7:0 | TSM | | | | | | PSRASY | PSRSYNC |
| 0x44 | TCCR0A | 7:0 | COM0A1 | COM0A0 | COM0B1 | COM0B0 | | | WGM01 | WGM00 |
| 0x45 | TCCR0B | 7:0 | FOC0A | FOC0B | | | WGM02 | CS0[2:0] | | |
| 0x46 | TCNT0 | 7:0 | TCNT0[7:0] | | | | | | | |
| 0x47 | OCR0A | 7:0 | OCR0A[7:0] | | | | | | | |
| 0x48 | OCR0B | 7:0 | OCR0B[7:0] | | | | | | | |
| 0x49 | Reserved | | | | | | | | | |
| 0x4A | GPIOR1 | 7:0 | GPIOR1[7:0] | | | | | | | |
| 0x4B | GPIOR2 | 7:0 | GPIOR2[7:0] | | | | | | | |
| 0x4C | SPCR0 | 7:0 | SPIE0 | SPE0 | DORD0 | MSTR0 | CPOL0 | CPHA0 | SPR01 | SPR00 |
| 0x4D | SPSR0 | 7:0 | SPIF0 | WCOL0 | | | | | | SPI2X0 |
| 0x4E | SPDR0 | 7:0 | SPID[7:0] | | | | | | | |
| 0x4F | Reserved | | | | | | | | | |
| 0x50 | ACSR | 7:0 | ACD | ACBG | ACO | ACI | ACIE | ACIC | ACIS1 | ACIS0 |
| 0x51 | DWDR | 7:0 | DWDR[7:0] | | | | | | | |
| 0x52 | Reserved | | | | | | | | | |
| 0x53 | SMCR | 7:0 | | | | | SM2 | SM1 | SM0 | SE |
| 0x54 | MCUSR | 7:0 | | | | | WDRF | BORF | EXTRF | PORF |

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x55 | MCUCR | 7:0 | | | | PUD | | | IVSEL | IVCE |
| 0x56 | Reserved | | | | | | | | | |
| 0x57 | SPMCSR | 7:0 | SPMIE | RWWSB | SIGRD | RWWSRE | BLBSET | PGWRT | PGERS | SPMEN |
| 0x58 ... 0x5C | Reserved | | | | | | | | | |
| 0x5D | SPL | 7:0 | (SP[7:0]) SPL | | | | | | | |
| 0x5E | SPH | 7:0 | | | | | | (SP[10:8]) SPH | | |
| 0x5F | SREG | 7:0 | I | T | H | S | V | N | Z | C |
| 0x60 | WDTCSR | 7:0 | WDIF | WDIE | WDP[3] | WDCE | WDE | WDP[2:0] | | |
| 0x61 | CLKPR | 7:0 | CLKPCE | | | | CLKPS3 | CLKPS2 | CLKPS1 | CLKPS0 |
| 0x62 ... 0x63 | Reserved | | | | | | | | | |
| 0x64 | PRR | 7:0 | PRTWI0 | PRTIM2 | PRTIM0 | | PRTIM1 | PRSPI0 | PRUSART0 | PRADC |
| 0x65 | Reserved | | | | | | | | | |
| 0x66 | OSCCAL | 7:0 | CAL7 | CAL6 | CAL5 | CAL4 | CAL3 | CAL2 | CAL1 | CAL0 |
| 0x67 | Reserved | | | | | | | | | |
| 0x68 | PCICR | 7:0 | | | | | | PCIE2 | PCIE1 | PCIE0 |
| 0x69 | EICRA | 7:0 | | | | | ISC11 | ISC10 | ISC01 | ISC00 |
| 0x6A | Reserved | | | | | | | | | |
| 0x6B | PCMSK0 | 7:0 | PCINT7 | PCINT6 | PCINT5 | PCINT4 | PCINT3 | PCINT2 | PCINT1 | PCINT0 |
| 0x6C | PCMSK1 | 7:0 | | PCINT14 | PCINT13 | PCINT12 | PCINT11 | PCINT10 | PCINT9 | PCINT8 |
| 0x6D | PCMSK2 | 7:0 | PCINT23 | PCINT22 | PCINT21 | PCINT20 | PCINT19 | PCINT18 | PCINT17 | PCINT16 |
| 0x6E | TIMSK0 | 7:0 | | | | | | OCIEB | OCIEA | TOIE |
| 0x6F | TIMSK1 | 7:0 | | | ICIE | | | OCIEB | OCIEA | TOIE |
| 0x70 | TIMSK2 | 7:0 | | | | | | OCIEB | OCIEA | TOIE |
| 0x71 ... 0x77 | Reserved | | | | | | | | | |
| 0x78 | ADCL | 7:0 | ADC7 | ADC6 | ADC5 | ADC4 | ADC3 | ADC2 | ADC1 | ADC0 |
| 0x79 | ADCH | 7:0 | | | | | | | ADC9 | ADC8 |
| 0x7A | ADCSRA | 7:0 | ADEN | ADSC | ADATE | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 |
| 0x7B | ADCSRB | 7:0 | | ACME | | | | ADTS2 | ADTS1 | ADTS0 |
| 0x7C | ADMUX | 7:0 | REFS1 | REFS0 | ADLAR | | MUX3 | MUX2 | MUX1 | MUX0 |
| 0x7D | Reserved | | | | | | | | | |
| 0x7E | DIDR0 | 7:0 | ADC7D | ADC6D | ADC5D | ADC4D | ADC3D | ADC2D | ADC1D | ADC0D |
| 0x7F | DIDR1 | 7:0 | | | | | | | AIN1D | AIN0D |
| 0x80 | TCCR1A | 7:0 | COM1 | COM1 | COM1 | COM1 | | | WGM11 | WGM10 |
| 0x81 | TCCR1B | 7:0 | ICNC1 | ICES1 | | WGM13 | WGM12 | CS12 | CS11 | CS10 |
| 0x82 | TCCR1C | 7:0 | FOC1A | FOC1B | | | | | | |
| 0x83 | Reserved | | | | | | | | | |
| 0x84 | TCNT1L | 7:0 | TCNT1L[7:0] | | | | | | | |
| 0x85 | TCNT1H | 7:0 | TCNT1H[7:0] | | | | | | | |
| 0x86 | ICR1L | 7:0 | ICR1L[7:0] | | | | | | | |
| 0x87 | ICR1H | 7:0 | ICR1H[7:0] | | | | | | | |
| 0x88 | OCR1AL | 7:0 | OCR1AL[7:0] | | | | | | | |

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x89 | OCR1AH | 7:0 | OCR1AH[7:0] | | | | | | | |
| 0x8A | OCR1BL | 7:0 | OCR1BL[7:0] | | | | | | | |
| 0x8B | OCR1BH | 7:0 | OCR1BH[7:0] | | | | | | | |
| 0x8C ... 0xAF | Reserved | | | | | | | | | |
| 0xB0 | TCCR2A | 7:0 | COM2A1 | COM2A0 | COM2B1 | COM2B0 | | | WGM21 | WGM20 |
| 0xB1 | TCCR2B | 7:0 | FOC2A | FOC2B | | | WGM22 | CS2[2:0] | | |
| 0xB2 | TCNT2 | 7:0 | TCNT2[7:0] | | | | | | | |
| 0xB3 | OCR2A | 7:0 | OCR2A[7:0] | | | | | | | |
| 0xB4 | OCR2B | 7:0 | OCR2B[7:0] | | | | | | | |
| 0xB5 | Reserved | | | | | | | | | |
| 0xB6 | ASSR | 7:0 | | EXCLK | AS2 | TCN2UB | OCR2AUB | OCR2BUB | TCR2AUB | TCR2BUB |
| 0xB7 | Reserved | | | | | | | | | |
| 0xB8 | TWBR | 7:0 | TWBR7 | TWBR6 | TWBR5 | TWBR4 | TWBR3 | TWBR2 | TWBR1 | TWBR0 |
| 0xB9 | TWSR | 7:0 | TWS4 | TWS3 | TWS2 | TWS1 | TWS0 | | TWPS1 | TWPS0 |
| 0xBA | TWAR | 7:0 | TWA6 | TWA5 | TWA4 | TWA3 | TWA2 | TWA1 | TWA0 | TWGCE |
| 0xBB | TWDR | 7:0 | TWD7 | TWD6 | TWD5 | TWD4 | TWD3 | TWD2 | TWD1 | TWD0 |
| 0xBC | TWCR | 7:0 | TWINT | TWEA | TWSTA | TWSTO | TWWC | TWEN | | TWIE |
| 0xBD | TWAMR | 7:0 | TWAM6 | TWAM5 | TWAM4 | TWAM3 | TWAM2 | TWAM1 | TWAM0 | |
| 0xBE ... 0xBF | Reserved | | | | | | | | | |
| 0xC0 | UCSR0A | 7:0 | RXC0 | TXC0 | UDRE0 | FE0 | DOR0 | UPE0 | U2X0 | MPCM0 |
| 0xC1 | UCSR0B | 7:0 | RXCIE0 | TXCIE0 | UDRIE0 | RXEN0 | TXEN0 | UCSZ02 | RXB80 | TXB80 |
| 0xC2 | UCSR0C | 7:0 | UMSEL01 | UMSEL00 | UPM01 | UPM00 | USBS0 | UCSZ01 / UDORD0 | UCSZ00 / UCPHA0 | UCPOL0 |
| 0xC3 | Reserved | | | | | | | | | |
| 0xC4 | UBRR0L | 7:0 | UBRR0[7:0] | | | | | | | |
| 0xC5 | UBRR0H | 7:0 | | | | | UBRR0[3:0] | | | |
| 0xC6 | UDR0 | 7:0 | TXB / RXB[7:0] | | | | | | | |

## 35.1. 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 operate on the specified bit, and can therefore be used on registers containing such Status Flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATmega48A/88A/168A is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT

instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

5. SPH.SP10, SPMCSR.RWWSB, SPMCSR.RWWSRE and EEARH registers are only available for ATmega88A and ATmega168A.

# 36. Instruction Set Summary

| ARITHMETIC AND LOGIC INSTRUCTIONS | | | | | |
|---|---|---|---|---|---|
| **Mnemonics** | **Operands** | **Description** | **Operation** | **Flags** | **#Clocks** |
| ADD | Rd, Rr | Add two Registers without Carry | Rd ← Rd + Rr | Z,C,N,V,H | 1 |
| ADC | Rd, Rr | Add two Registers with Carry | 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 two Registers with Carry | Rd ← Rd - Rr - C | Z,C,N,V,H | 1 |
| SBCI | Rd, K | Subtract Constant from Reg with Carry. | 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 |
| MUL | Rd, Rr | Multiply Unsigned | R1:R0 ← Rd x Rr | Z,C | 2 |
| MULS | Rd, Rr | Multiply Signed | R1:R0 ← Rd x Rr | Z,C | 2 |
| MULSU | Rd, Rr | Multiply Signed with Unsigned | R1:R0 ← Rd x Rr | Z,C | 2 |
| FMUL | Rd, Rr | Fractional Multiply Unsigned | R1:R0 ← (Rd x Rr) << 1 | Z,C | 2 |
| FMULS | Rd, Rr | Fractional Multiply Signed | R1:R0 ← (Rd x Rr) << 1 | Z,C | 2 |
| FMULSU | Rd, Rr | Fractional Multiply Signed with Unsigned | R1:R0 ← (Rd x Rr) << 1 | Z,C | 2 |

| BRANCH INSTRUCTIONS | | | | | |
|---|---|---|---|---|---|
| **Mnemonics** | **Operands** | **Description** | **Operation** | **Flags** | **#Clocks** |
| RJMP | k | Relative Jump | PC ← PC + k + 1 | None | 2 |
| IJMP | | Indirect Jump to (Z) | PC ← Z | None | 2 |
| JMP(1) | k | Direct Jump | PC ← k | None | 3 |
| RCALL | k | Relative Subroutine Call | PC ← PC + k + 1 | None | 3 |

| BRANCH INSTRUCTIONS | | | | | |
|---|---|---|---|---|---|
| **Mnemonics** | **Operands** | **Description** | **Operation** | **Flags** | **#Clocks** |
| ICALL | | Indirect Call to (Z) | PC ← Z | None | 3 |
| CALL(1) | k | Direct Subroutine Call | PC ← k | None | 4 |
| 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 | A, b | Skip if Bit in I/O Register Cleared | if (I/O(A,b)=1) PC ← PC + 2 or 3 | None | 1/2/3 |
| SBIS | A, b | Skip if Bit in I/O Register is Set | if (I/O(A,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 | | | | | |
|---|---|---|---|---|---|
| **Mnemonics** | **Operands** | **Description** | **Operation** | **Flags** | **#Clocks** |
| 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 |

| BIT AND BIT-TEST INSTRUCTIONS | | | | | |
|---|---|---|---|---|---|
| **Mnemonics** | **Operands** | **Description** | **Operation** | **Flags** | **#Clocks** |
| LSL | Rd | Logical Shift Left | Rd(n+1) ← Rd(n), Rd(0) ← 0 | Z,C,N,V | 1 |
| LSR | Rd | Logical Shift Right | Rd(n) ← Rd(n+1), Rd(7) ← 0 | Z,C,N,V | 1 |
| ROL | Rd | Rotate Left Through Carry | Rd(0)←C,Rd(n+1)← Rd(n),C¬Rd(7) | Z,C,N,V | 1 |
| ROR | Rd | Rotate Right Through Carry | Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0) | Z,C,N,V | 1 |
| ASR | Rd | Arithmetic Shift Right | Rd(n) ← Rd(n+1), n=0...6 | Z,C,N,V | 1 |
| SWAP | Rd | Swap Nibbles | Rd(3...0)←Rd(7...4),Rd(7...4)¬Rd(3...0) | None | 1 |
| BSET | s | Flag Set | SREG(s) ← 1 | SREG(s) | 1 |
| BCLR | s | Flag Clear | SREG(s) ← 0 | SREG(s) | 1 |
| BST | Rr, b | Bit Store from Register to T | T ← Rr(b) | T | 1 |
| BLD | Rd, b | Bit load from T to Register | Rd(b) ← T | None | 1 |
| SEC | | Set Carry | C ← 1 | C | 1 |
| CLC | | Clear Carry | C ← 0 | C | 1 |
| SEN | | Set Negative Flag | N ← 1 | N | 1 |
| CLN | | Clear Negative Flag | N ← 0 | N | 1 |
| SEZ | | Set Zero Flag | Z ← 1 | Z | 1 |
| CLZ | | Clear Zero Flag | Z ← 0 | Z | 1 |
| SEI | | Global Interrupt Enable | I ← 1 | I | 1 |
| CLI | | Global Interrupt Disable | I ← 0 | I | 1 |
| SES | | Set Signed Test Flag | S ← 1 | S | 1 |
| CLS | | Clear Signed Test Flag | S ← 0 | S | 1 |
| SEV | | Set Two's Complement Overflow. | V ← 1 | V | 1 |
| CLV | | Clear Two's Complement Overflow | V ← 0 | V | 1 |
| SET | | Set T in SREG | T ← 1 | T | 1 |
| CLT | | Clear T in SREG | T ← 0 | T | 1 |
| SEH | | Set Half Carry Flag in SREG | H ← 1 | H | 1 |
| CLH | | Clear Half Carry Flag in SREG | H ← 0 | H | 1 |

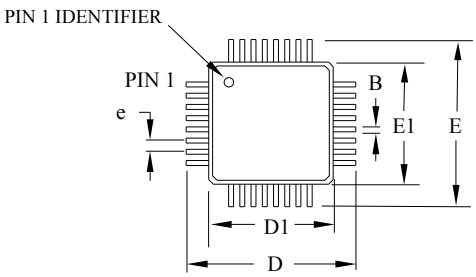
| DATA TRANSFER INSTRUCTIONS | | | | | |
|---|---|---|---|---|---|
| **Mnemonics** | **Operands** | **Description** | **Operation** | **Flags** | **#Clocks** |
| MOV | Rd, Rr | Move Between Registers | Rd ← Rr | None | 1 |
| MOVW | Rd, Rr | Copy Register Word | Rd+1:Rd ← Rr+1:Rr | None | 1 |
| LDI | Rd, K | Load Immediate | Rd ← K | None | 1 |
| LD | Rd, X | Load Indirect | Rd ← (X) | None | 2 |
| LD | Rd, X+ | Load Indirect and Post-Increment | Rd ← (X), X ← X + 1 | None | 2 |
| LD | Rd, - X | Load Indirect and Pre-Decrement | X ← X - 1, Rd ← (X) | None | 2 |
| LD | Rd, Y | Load Indirect | Rd ← (Y) | None | 2 |
| LD | Rd, Y+ | Load Indirect and Post-Increment | Rd ← (Y), Y ← Y + 1 | None | 2 |

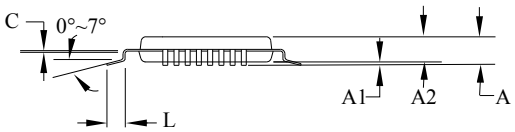
| DATA TRANSFER INSTRUCTIONS | | | | | |
|---|---|---|---|---|---|
| **Mnemonics** | **Operands** | **Description** | **Operation** | **Flags** | **#Clocks** |
| LD | Rd, - Y | Load Indirect and Pre-Decrement | Y ← Y - 1, Rd ← (Y) | None | 2 |
| LDD | Rd,Y+q | Load Indirect with Displacement | Rd ← (Y + q) | None | 2 |
| LD | Rd, Z | Load Indirect | Rd ← (Z) | None | 2 |
| LD | Rd, Z+ | Load Indirect and Post-Increment | Rd ← (Z), Z ← Z+1 | None | 2 |
| LD | Rd, -Z | Load Indirect and Pre-Decrement | 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-Increment | (X) ← Rr, X ← X + 1 | None | 2 |
| ST | - X, Rr | Store Indirect and Pre-Decrement | X ← X - 1, (X) ← Rr | None | 2 |
| ST | Y, Rr | Store Indirect | (Y) ← Rr | None | 2 |
| ST | Y+, Rr | Store Indirect and Post-Increment | (Y) ← Rr, Y ← Y + 1 | None | 2 |
| ST | - Y, Rr | Store Indirect and Pre-Decrement | 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-Increment | (Z) ← Rr, Z ← Z + 1 | None | 2 |
| ST | -Z, Rr | Store Indirect and Pre-Decrement | 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, A | In from I/O Location | Rd ← I/O (A) | None | 1 |
| OUT | A, Rr | Out to I/O Location | I/O (A) ← Rr | None | 1 |
| PUSH | Rr | Push Register on Stack | STACK ← Rr | None | 2 |
| POP | Rd | Pop Register from Stack | Rd ← STACK | None | 2 |

| MCU CONTROL INSTRUCTIONS | | | | | |
|---|---|---|---|---|---|
| **Mnemonics** | **Operands** | **Description** | **Operation** | **Flags** | **#Clocks** |
| NOP | | No Operation | 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 |

# 37. Packaging Information

## 37.1. 32-pin 32A




**COMMON DIMENSIONS**
(Unit of measure = mm)

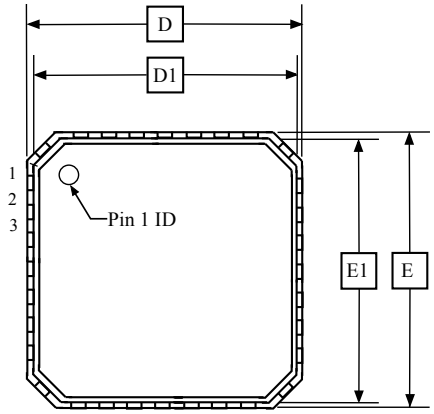
| SYMBOL | MIN | NOM | MAX | NOTE |
|---|---|---|---|---|
| A | – | – | 1.20 | |
| A1 | 0.05 | – | 0.15 | |
| A2 | 0.95 | 1.00 | 1.05 | |
| D | 8.75 | 9.00 | 9.25 | |
| D1 | 6.90 | 7.00 | 7.10 | Note 2 |
| E | 8.75 | 9.00 | 9.25 | |
| E1 | 6.90 | 7.00 | 7.10 | Note 2 |
| B | 0.30 | – | 0.45 | |
| C | 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.

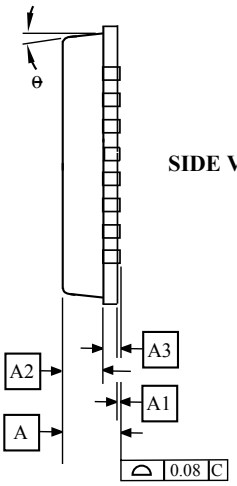
2010-10-20

| | TITLE | DRAWING NO. | REV. |
|---|---|---|---|
| Atmel | **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 |

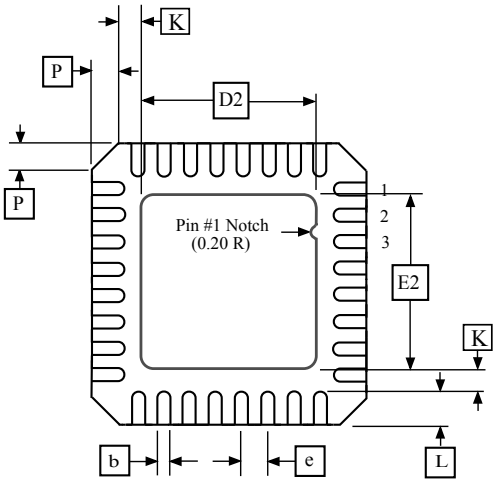
## 37.2. 32-pin 32M1-A




TOP VIEW




SIDE VIEW




BOTTOM VIEW

**COMMON DIMENSIONS**
(Unit of Measure = mm)

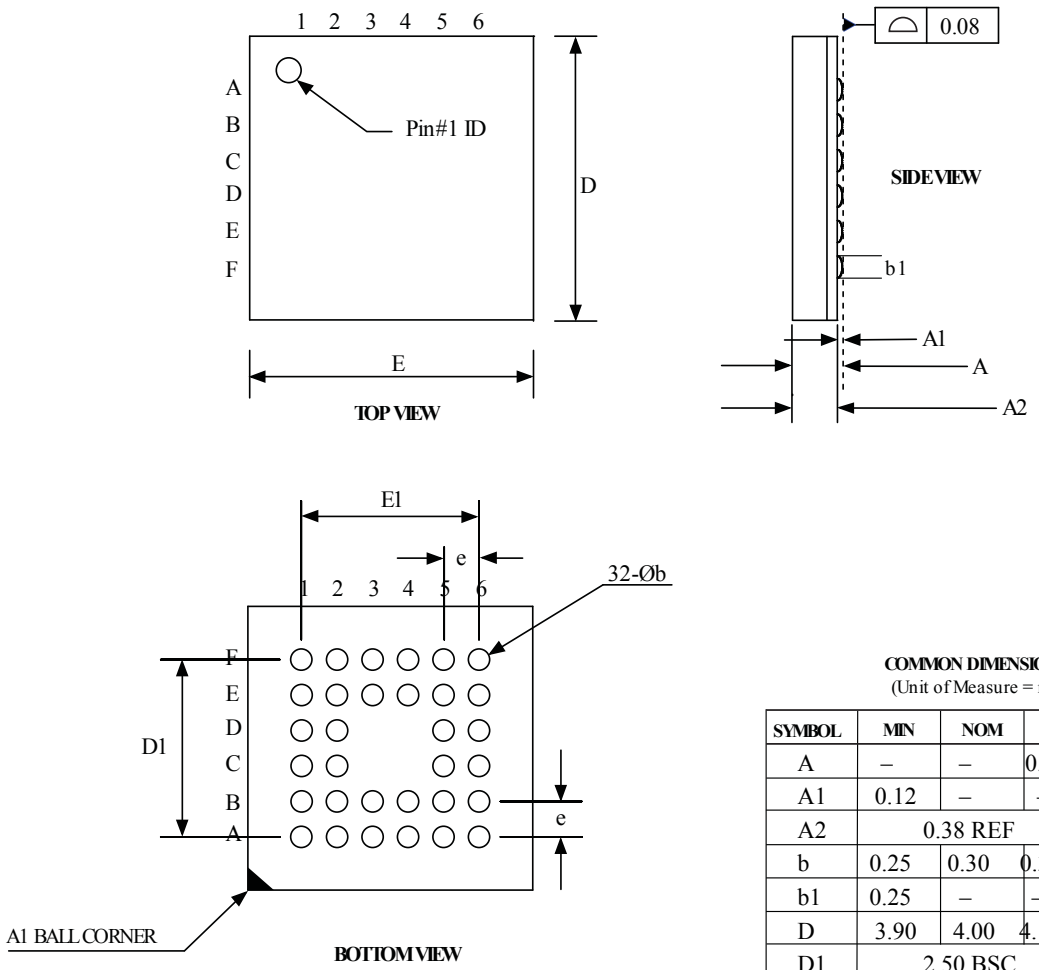
| SYMBOL | MIN | NOM | MAX | NOTE |
|---|---|---|---|---|
| A | 0.80 | 0.90 | 1.00 | |
| A1 | – | 0.02 | 0.05 | |
| A2 | – | 0.65 | 1.00 | |
| A3 | | 0.20 REF | | |
| b | 0.18 | 0.23 | 0.30 | |
| D | 4.90 | 5.00 | 5.10 | |
| D1 | 4.70 | 4.75 | 4.80 | |
| D2 | 2.95 | 3.10 | 3.25 | |
| E | 4.90 | 5.00 | 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° | |
| K | 0.20 | – | – | |

Note : JEDEC Standard MO-220, Fig . 2 (Anvil Singulation), VHHD-2 .

03/14/2014

| | TITLE | DRAWING NO. | REV. |
|---|---|---|---|
| Atmel Package Drawing Contact: packagedrawings@atmel.com | **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 | F |

## 37.3. 32-pin 32CC1




**COMMON DIMENSIONS**
(Unit of Measure = mm)

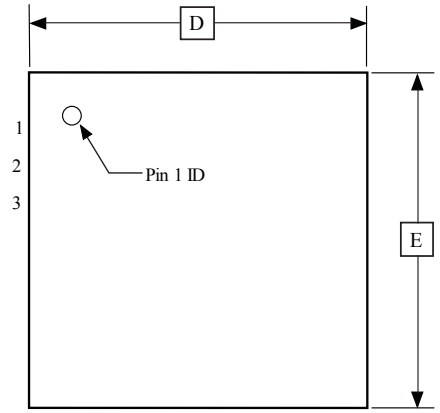
| SYMBOL | MIN | NOM | MAX | NOTE |
|---|---|---|---|---|
| A | – | – | 0.60 | |
| A1 | 0.12 | – | – | |
| A2 | | 0.38 REF | | |
| b | 0.25 | 0.30 | 0.35 | 1 |
| b1 | 0.25 | – | – | 2 |
| D | 3.90 | 4.00 | 4.10 | |
| D1 | | 2.50 BSC | | |
| E | 3.90 | 4.00 | 4.10 | |
| E1 | | 2.50 BSC | | |
| e | | 0.50 BSC | | |

Note1: Dimension "b" is measured at the maximum ball dia. in a plane parallel to the seating plane.
Note2: Dimension "b1" is the solderable surface defined by the opening of the solder resist layer.

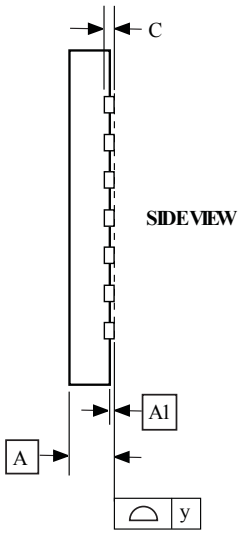
07/06/10

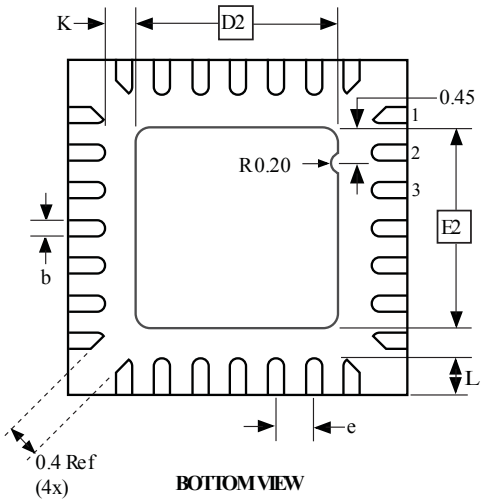
| Atmel | Package Drawing Contact: packagedrawings@atmel.com | TITLE | GPC | DRAWING NO. | REV. |
|---|---|---|---|---|---|
| | | **32CC1**, 32-ball (6 x 6 Array), 4 x 4 x 0.6 mm package, ball pitch 0.50 mm, Ultra Thin, Fine-Pitch Ball Grid Array (UFBGA) | CAG | 32CC1 | B |

## 37.4. 28-pin 28M1




TOP VIEW




BOTTOM VIEW

Note: The terminal #1 ID is a laser-marked feature.

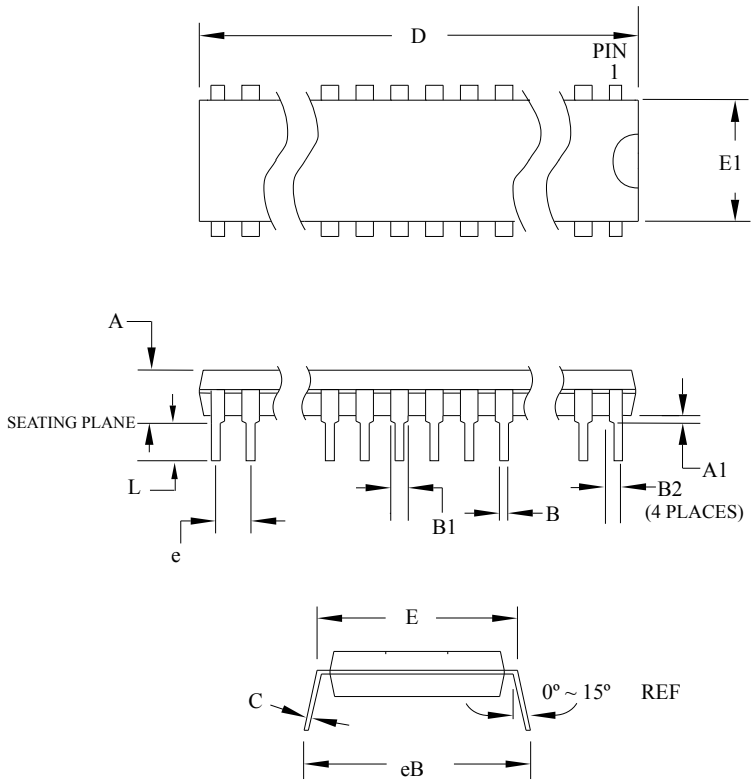
**COMMON DIMENSIONS**
(Unit of Measure = mm)

| SYMBOL | MIN | NOM | MAX | NOTE |
|---|---|---|---|---|
| A | 0.80 | 0.90 | 1.00 | |
| A1 | 0.00 | 0.02 | 0.05 | |
| b | 0.17 | 0.22 | 0.27 | |
| C | | 0.20 REF | | |
| D | 3.95 | 4.00 | 4.05 | |
| D2 | 2.35 | 2.40 | 2.45 | |
| E | 3.95 | 4.00 | 4.05 | |
| E2 | 2.35 | 2.40 | 2.45 | |
| e | | 0.45 | | |
| L | 0.35 | 0.40 | 0.45 | |
| y | 0.00 | – | 0.08 | |
| K | 0.20 | – | – | |

10/24/08

| Package Drawing Contact: packagedrawings@atmel.com | TITLE | GPC | DRAWING NO. | REV. |
|---|---|---|---|---|
| | **28M1**, 28-pad, 4 x 4 x 1.0mm Body, Lead Pitch 0.45mm, 2.4 x 2.4mm Exposed Pad, Thermally Enhanced Plastic Very Thin Quad Flat No Lead Package (VQFN) | ZBV | 28M1 | B |

## 37.5. 28-pin 28P3




**COMMON DIMENSIONS**
(Unit of Measure = mm)

| SYMBOL | MIN | NOM | MAX | NOTE |
|---|---|---|---|---|
| A | – | – | 4.5724 | |
| A1 | 0.508 | – | – | |
| D | 34.544 | – | 34.798 | Note 1 |
| E | 7.620 | – | 8.255 | |
| E1 | 7.112 | – | 7.493 | Note 1 |
| B | 0.381 | – | 0.533 | |
| B1 | 1.143 | – | 1.397 | |
| B2 | 0.762 | – | 1.143 | |
| L | 3.175 | – | 3.429 | |
| C | 0.203 | – | 0.356 | |
| eB | – | – | 10.160 | |
| e | 2.540 TYP | | | |

Note: 1. Dimensions D and E1 do not include mold Flash or Protrusion. Mold Flash or Protrusion shall not exceed 0.25mm (0.010").

09/28/01

| | TITLE | DRAWING NO. | REV. |
|---|---|---|---|
| Atmel 2325 Orchard Parkway San Jose, CA 95131 | **28P3**, 28-lead (0.300"/7.62mm Wide) Plastic Dual Inline Package (PDIP) | 28P3 | B |

# 38. Errata

## 38.1. Errata ATmega48A

The revision letter in this section refers to the revision of the ATmega48A device.

### 38.1.1. Rev. D

**1 – Analog MUX can be turned off when setting ACME bit**

If the ACME (Analog Comparator Multiplexer Enabled) bit in ADCSRB is set while MUX3 in ADMUX is '1' (ADMUX[3:0]=1xxx), all MUXes are turned off until the ACME bit is cleared.

**Fix/Workaround:**

Clear the MUX3 bit before setting the ACME bit.

**2 – TWI Data setup time can be too short**

When running the device as a TWI slave with a system clock above 2MHz, the data setup time for the first bit after ACK may in some cases be too short. This may cause a false start or stop condition on the TWI line.

**Fix/Workaround:**

Insert a delay between setting TWDR and TWCR.

### 38.1.2. Rev. B to C

Not sampled.

### 38.1.3. Rev. A

**1 – Power consumption in power save modes**

Power consumption in power save modes will be higher due to improper control of internal power management.

**Fix/Workaround:**

This problem will be corrected in Rev B.

**2 – Start-up time for the device**

Due to implementation of a different NVM structure, the startup sequence for the device will require longer start-up time.

**Fix/Workaround:**

There is no fix for this problem.

## 38.2. Errata ATmega88A

The revision letter in this section refers to the revision of the ATmega88A device.

### 38.2.1. Rev. F

**1 –**

**Analog MUX can be turned off when setting ACME bit**

If the ACME (Analog Comparator Multiplexer Enabled) bit in ADCSRB is set while MUX3 in ADMUX is '1' (ADMUX[3:0]=1xxx), all MUXes are turned off until the ACME bit is cleared.

**Fix/Workaround:**

Clear the MUX3 bit before setting the ACME bit.

**2 –**

**TWI Data setup time can be too short**

When running the device as a TWI slave with a system clock above 2MHz, the data setup time for the first bit after ACK may in some cases be too short. This may cause a false start or stop condition on the TWI line.

**Fix/Workaround:**

Insert a delay between setting TWDR and TWCR.

### 38.2.2. Rev. B to E

Not sampled.

### 38.2.3. Rev. A

**1 – Power consumption in power save modes**

Power consumption in power save modes will be higher due to improper control of internal power management.

**Fix/Workaround:**

This problem will be corrected in Rev B.

**2 – Start-up time for the device**

Due to implementation of a different NVM structure, the startup sequence for the device will require longer start-up time.

**Fix/Workaround:**

There is no fix for this problem.

## 38.3. Errata ATmega168A

The revision letter in this section refers to the revision of the ATmega168A device.

### 38.3.1. Rev. E

**1 – Analog MUX can be turned off when setting ACME bit**

If the ACME (Analog Comparator Multiplexer Enabled) bit in ADCSRB is set while MUX3 in ADMUX is '1' (ADMUX[3:0]=1xxx), all MUXes are turned off until the ACME bit is cleared.

**Fix/Workaround:**

Clear the MUX3 bit before setting the ACME bit.

**2 – TWI Data setup time can be too short**

When running the device as a TWI slave with a system clock above 2MHz, the data setup time for the first bit after ACK may in some cases be too short. This may cause a false start or stop condition on the TWI line.

**Fix/Workaround:**

Insert a delay between setting TWDR and TWCR.

### 38.3.2. Rev. A to D

Not sampled.

# 39. Datasheet Revision History

Please note that the referring page numbers in this section are referred to this document. The referring revision in this section are referring to the document revision.

## 39.1. Rev. B – 11/2016

1. Update I/O Multiplexing
2. Errata section updated
   - ATmega48PA: Removed die revision E to K
     - Die revision E to J was not sampled.
     - Die revision K was not released to production.
   - ATmega88PA: Removed die revision G to K
     - Die revision G to J was not sampled.
     - Die revision K was not released to production.
   - ATmega168PA: Removed die revision F to K
     - Die revision F to J was not sampled.
     - Die revision K was not released to production.

## 39.2. Rev. A – 06/2016

Initial document release.

**Atmel Corporation** 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**

© 2016 Atmel Corporation. / Rev.: Atmel-42733B-ATmega48A/88A/168A_Datasheet_Complete-11/2016

Atmel®, Atmel logo and combinations thereof, Enabling Unlimited Possibilities®, AVR®, and others are registered trademarks or trademarks of Atmel Corporation in U.S. and other countries. Other terms and product names may be trademarks of others.

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.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death (“Safety-Critical Applications”) without an Atmel officer's specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.

# Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

Microchip:

ATMEGA168A-CCUR ATMEGA48A-CCUR ATMEGA168A-AU ATMEGA168A-AUR ATMEGA168A-CCU ATMEGA168A-MMH ATMEGA168A-MMHR ATMEGA168A-MU ATMEGA168A-MUR ATMEGA168A-PU ATMEGA48A-AU ATMEGA48A-AUR ATMEGA48A-CCU ATMEGA48A-MMH ATMEGA48A-MMHR ATMEGA48A-MU ATMEGA48A-MUR ATMEGA48A-PU ATMEGA88A-AU ATMEGA88A-AUR ATMEGA88A-CCU ATMEGA88A-CCUR ATMEGA88A-MMH ATMEGA88A-MMHR ATMEGA88A-MU ATMEGA88A-MUR ATMEGA88A-PU

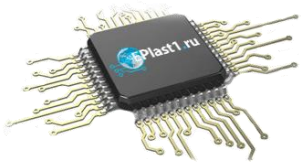
Компания «ЭлектроПласт» предлагает заключение долгосрочных отношений при поставках импортных электронных компонентов на взаимовыгодных условиях!

Наши преимущества:

- Оперативные поставки широкого спектра электронных компонентов отечественного и импортного производства напрямую от производителей и с крупнейших мировых складов;
- Поставка более 17-ти миллионов наименований электронных компонентов;
- Поставка сложных, дефицитных, либо снятых с производства позиций;
- Оперативные сроки поставки под заказ (от 5 рабочих дней);
- Экспресс доставка в любую точку России;
- Техническая поддержка проекта, помощь в подборе аналогов, поставка прототипов;
- Система менеджмента качества сертифицирована по Международному стандарту ISO 9001;
- Лицензия ФСБ на осуществление работ с использованием сведений, составляющих государственную тайну;
- Поставка специализированных компонентов (Xilinx, Altera, Analog Devices, Intersil, Interpoint, Microsemi, Aeroflex, Peregrine, Syfer, Eurofarad, Texas Instrument, Miteq, Cobham, E2V, MA-COM, Hittite, Mini-Circuits,General Dynamics и др.);

Помимо этого, одним из направлений компании «ЭлектроПласт» является направление «Источники питания». Мы предлагаем Вам помощь Конструкторского отдела:

- Подбор оптимального решения, техническое обоснование при выборе компонента;
- Подбор аналогов;
- Консультации по применению компонента;
- Поставка образцов и прототипов;
- Техническая поддержка проекта;
- Защита от снятия компонента с производства.




**Как с нами связаться**

**Телефон:** 8 (812) 309 58 32 (многоканальный)
**Факс:** 8 (812) 320-02-42
**Электронная почта:** org@eplast1.ru
**Адрес:** 198099, г. Санкт-Петербург, ул. Калинина, дом 2, корпус 4, литера А.