# ATtiny212/412

## AVR® Microcontroller with Core Independent Peripherals and picoPower® technology

## Introduction

The ATtiny212/412 microcontrollers are using the high-performance low-power AVR® RISC architecture, and is capable of running at up to 20MHz, with up to 2/4KB Flash, 128/256bytes of SRAM and 64/128bytes of EEPROM in a 8- pin package. The series uses the latest technologies with a flexible and low power architecture including Event System and SleepWalking, accurate analog features and advanced peripherals.

## Features

- CPU
  - AVR® 8-bit CPU
  - Running at up to 20MHz
  - Single Cycle I/O Access
  - Two-level Interrupt Controller
  - Two-cycle Hardware Multiplier
- Memories
  - 2/4KB In-system self-programmable Flash Memory
  - 64/128B EEPROM
  - 128/256B SRAM
- System
  - Power-on Reset (POR)
  - Brown-out Detection (BOD)
  - Clock Options:
    - 16/20MHz Low Power Internal RC Oscillator
    - 32.768kHz Ultra Low Power (ULP) Internal RC Oscillator
    - 32.768kHz External Crystal Oscillator
    - External Clock Input
  - Single Pin Unified Program Debug Interface (UPDI)
  - Three Sleep Modes:
    - Idle with All Peripherals Running and Mode for Immediate Wake Up Time
    - Standby
      - Configurable Operation of Selected Peripherals
      - SleepWalking Peripherals
    - Power Down with Wake-up Functionality
- Peripherals

© 2017 Microchip Technology Inc.

  - 6-channel Event System
  - One 16-bit Timer/Counter Type A with Dedicated Period Register, Three Compare Channels (TCA)
  - One 16-bit Timer/Counter type B with Input Capture (TCB)
  - One 12-bit Timer/Counter type D Optimized for Control Applications (TCD)
  - One 16-bit Real Time Counter (RTC) Running from External Crystal or Internal RC Oscillator
  - One USART with Fractional Baud Rate Generator, Auto-baud, and Start-of-frame Detection
  - Master/Slave Serial Peripheral Interface (SPI)
  - Master/Slave TWI with Dual Address Match
    - Standard Mode (Sm, 100kHz)
    - Fast Mode (Fm, 400kHz)
    - Fast Mode Plus (Fm+, 1MHz)
  - Configurable Custom Logic (CCL) with Two Programmable Lookup Tables (LUT)
  - Analog Comparator (AC) with Low Propagation Delay
  - 10-bit 115ksps Analog to Digital Converter (ADC)
  - 8-bit Digital to Analog Converter (DAC)
  - Five Selectable Internal Voltage References: 0.55V, 1.1V, 1.5V, 2.5V and 4.3V
  - Automated CRC Memory Scan
  - Watchdog Timer (WDT) with Window Mode, with Separate On-chip Oscillator
  - External Interrupt on All General Purpose Pins
- I/O and Packages:
  - 6 Programmable I/O Lines
  - 8-pin SOIC150
- Temperature Ranges:
  - -40°C to 105°C
  - -40°C to 125°C Temperature Graded Device Options Available
- Speed Grades:
  - 0-5MHz @ 1.8V – 5.5V
  - 0-10MHz @ 2.7V – 5.5V
  - 0-20MHz @ 4.5V – 5.5V

© 2017 Microchip Technology Inc.

# Table of Contents


Introduction....1

Features....1

1. tinyAVR® 1-Series Overview....9
   1.1. Configuration Summary....10
2. Ordering Information....11
   2.1. ATtiny212....11
   2.2. ATtiny412....11
3. Block Diagram....12
4. Pinout....13
   4.1. 8-pin SOIC....13
5. I/O Multiplexing and Considerations....14
   5.1. Multiplexed Signals....14
6. Memories....15
   6.1. Overview....15
   6.2. Memory Map....16
   6.3. In-System Reprogrammable Flash Program Memory....16
   6.4. SRAM Data Memory....17
   6.5. EEPROM Data Memory....17
   6.6. User Row....17
   6.7. Signature Bytes....17
   6.8. I/O Memory....18
   6.9. FUSES - Configuration and User Fuses....19
7. Peripherals and Architecture....30
   7.1. Peripheral Module Address Map....30
   7.2. Interrupt Vector Mapping....31
   7.3. SYSCFG - System Configuration....32
8. AVR CPU....34
   8.1. Features....34
   8.2. Overview....34
   8.3. Architecture....34
   8.4. ALU - Arithmetic Logic Unit....36
   8.5. Functional Description....37
   8.6. Register Summary - CPU....42
   8.7. Register Description....42
9. NVMCTRL - Non Volatile Memory Controller....45
   9.1. Features....45

9.2. Overview ... 45
9.3. Functional Description ... 46
9.4. Register Summary - NVMCTRL ... 52
9.5. Register Description ... 52

10. CLKCTRL - Clock Controller ... 56
10.1. Features ... 56
10.2. Overview ... 56
10.3. Functional Description ... 58
10.4. Register Summary - CLKCTRL ... 63
10.5. Register Description ... 63

11. SLPCTRL - Sleep Controller ... 70
11.1. Features ... 70
11.2. Overview ... 70
11.3. Functional Description ... 71
11.4. Register Summary - SLPCTRL ... 74
11.5. Register Description ... 74

12. RSTCTRL - Reset Controller ... 75
12.1. Features ... 75
12.2. Overview ... 75
12.3. Functional Description ... 76
12.4. Register Summary - RSTCTRL ... 79
12.5. Register Description ... 79

13. CPUINT - CPU Interrupt Controller ... 81
13.1. Features ... 81
13.2. Overview ... 81
13.3. Functional Description ... 83
13.4. Register Summary - CPUINT ... 89
13.5. Register Description ... 89

14. EVSYS - Event System ... 92
14.1. Features ... 92
14.2. Overview ... 92
14.3. Functional Description ... 95
14.4. Register Summary - EVSYS ... 97
14.5. Register Description ... 97

15. PORTMUX - Port Multiplexer ... 103
15.1. Overview ... 103
15.2. Register Summary - PORTMUX ... 104
15.3. Register Description ... 104

16. PORT - I/O Pin Configuration ... 105
16.1. Features ... 105
16.2. Overview ... 105
16.3. Functional Description ... 107

16.4. Register Summary - PORT....111
16.5. Register Description - Ports....111
16.6. Register Summary - VPORT....117
16.7. Register Description - Virtual Ports....117

17. BOD - Brownout Detector....119
17.1. Features....119
17.2. Overview....119
17.3. Functional Description....121
17.4. Register Summary - BOD....123
17.5. Register Description....123

18. VREF - Voltage Reference....127
18.1. Features....127
18.2. Overview....127
18.3. Functional Description....127
18.4. Register Summary - VREF....129
18.5. Register Description....129

19. WDT - Watchdog Timer....131
19.1. Features....131
19.2. Overview....131
19.3. Functional Description....133
19.4. Register Summary - WDT....137
19.5. Register Description....137

20. TCA - 16-bit Timer/Counter Type A....139
20.1. Features....139
20.2. Overview....139
20.3. Functional Description....143
20.4. Register Summary - TCA in Normal Mode (CTRLD.SPLITM=0)....153
20.5. Register Description - Normal Mode....154
20.6. Register Summary - TCA in Split Mode (CTRLD.SPLITM=1)....166
20.7. Register Description - Split Mode....166

21. TCB - 16-bit Timer/Counter Type B....175
21.1. Features....175
21.2. Overview....175
21.3. Functional Description....178
21.4. Register Summary - TCB....186
21.5. Register Description....186

22. TCD - 12-bit Timer/Counter Type D....193
22.1. Features....193
22.2. Overview....193
22.3. Functional Description....197
22.4. Register Summary - TCD....219
22.5. Register Description....220

23. RTC - Real Time Counter....232
23.1. Features....232
23.2. Overview....232
23.3. RTC Functional Description....235
23.4. PIT Functional Description....235
23.5. Events....236
23.6. Interrupts....236
23.7. Sleep Mode Operation....237
23.8. Synchronization....237
23.9. Configuration Change Protection....237
23.10. Register Summary - RTC....238
23.11. Register Description....238

24. USART - Universal Synchronous and Asynchronous Receiver and Transmitter..247
24.1. Features....247
24.2. Overview....247
24.3. Functional Description....251
24.4. Register Summary - USART....265
24.5. Register Description....265

25. SPI - Serial Peripheral Interface....276
25.1. Features....276
25.2. Overview....276
25.3. Functional Description....279
25.4. Register Summary - SPI....287
25.5. Register Description....287

26. TWI - Two Wire Interface....292
26.1. Features....292
26.2. Overview....292
26.3. Functional Description....294
26.4. Register Summary - TWI....307
26.5. Register Description....307

27. CRCSCAN - Cyclic Redundancy Check Memory Scan....321
27.1. Features....321
27.2. Overview....321
27.3. Functional Description....323
27.4. Register Summary - CRCSCAN....326
27.5. Register Description....326

28. CCL – Configurable Custom Logic....329
28.1. Features....329
28.2. Overview....329
28.3. Functional Description....331
28.4. Register Summary - CCL....340
28.5. Register Description....340

29. AC – Analog Comparator....345

29.1. Features....345
29.2. Overview....345
29.3. Functional Description....347
29.4. Register Summary - AC....349
29.5. Register Description....349

30. ADC - Analog to Digital Converter....352
30.1. Features....352
30.2. Overview....352
30.3. Functional Description....356
30.4. Register Summary - ADC....364
30.5. Register Description....364

31. DAC - Digital to Analog Converter....374
31.1. Features....374
31.2. Overview....374
31.3. Functional Description....376
31.4. Register Summary - DAC....378
31.5. Register Description....378

32. UPDI - Unified Program and Debug Interface....379
32.1. Features....379
32.2. Overview....379
32.3. Functional Description....381
32.4. Register Summary - UPDI....402
32.5. Register Description....402

33. Electrical Characteristics....409
33.1. Disclaimer....409
33.2. Absolute Maximum Ratings....409
33.3. General Operating Ratings....410
33.4. Power Consumption....411
33.5. Wake-Up Time....412
33.6. Peripherals Power Consumption....413
33.7. BOD and POR Characteristics....414
33.8. External Reset Characteristics....415
33.9. Oscillators and Clocks....415
33.10. I/O Pin Characteristics....417
33.11. USART....419
33.12. SPI....420
33.13. TWI....421
33.14. Bandgap and VREF....423
33.15. ADC....424
33.16. DAC....426
33.17. AC....427
33.18. Programming Time....428

34. Typical Characteristics....429
34.1. Power Consumption....429

34.2. GPIO ... 436
34.3. VREF Characteristics ... 444
34.4. BOD Characteristics ... 446
34.5. ADC Characteristics ... 449
34.6. AC Characteristics ... 454
34.7. OSC20M Characteristics ... 456
34.8. OSCULP32K Characteristics ... 458

35. Packaging Information ... 459
35.1. Package Drawings ... 459
35.2. Thermal Considerations ... 460

36. Instruction Set Summary ... 461

37. Conventions ... 467
37.1. Numerical Notation ... 467
37.2. Memory Size and Type ... 467
37.3. Frequency and Time ... 467
37.4. Registers and Bits ... 468

38. Acronyms and Abbreviations ... 469

39. Errata ... 472
39.1. Errata - ATtiny212/ATtiny412 ... 472

40. Datasheet Revision History ... 475
40.1. Rev. A - 01/2017 ... 475

The Microchip Web Site ... 476

Customer Change Notification Service ... 476

Customer Support ... 476

Microchip Devices Code Protection Feature ... 476

Legal Notice ... 477

Trademarks ... 477

Quality Management System Certified by DNV ... 478

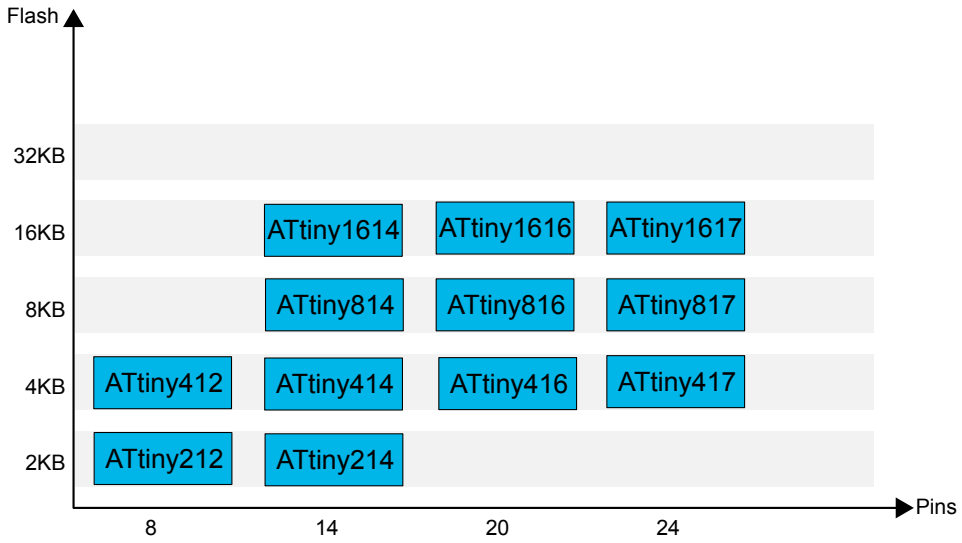
Worldwide Sales and Service ... 479

# 1. tinyAVR® 1-Series Overview

The figure below shows the tinyAVR 1-series, laying out pin count variants and memory sizes:

- Vertical migration can be done upwards without code modification, since these devices are pin compatible and provide the same or even more features. Downward migration may require code modification due to fewer available instances of some peripherals.
- Horizontal migration to the left reduces the pin count and therefore also the available features.

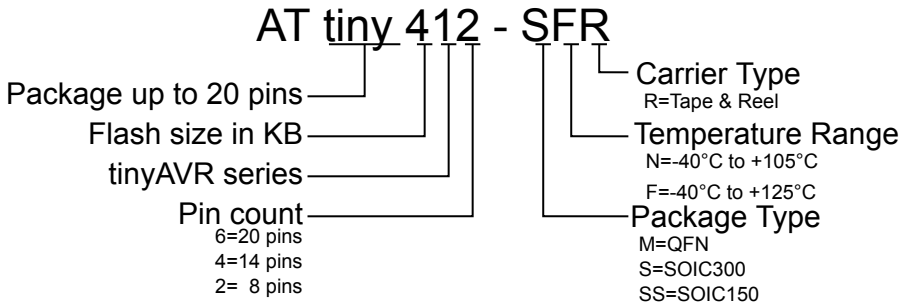
**Figure 1-1. tinyAVR®1-Series Overview**




Devices with different Flash memory size typically also have different SRAM and EEPROM.

The name of a device of the series contains information as depicted below:

**Figure 1-2. Device Designations**



## 1.1 Configuration Summary

### 1.1.1 Peripheral Summary

**Table 1-1. Peripheral Summary**

| | ATtiny212 | ATtiny412 |
|---|---|---|
| Pins | 8 | 8 |
| SRAM | 128B | 256B |
| Flash | 2KB | 4KB |
| EEPROM | 64B | 128B |
| Max. frequency (MHz) | 20 | 20 |
| 16-bit Timer/Counter type A (TCA) | 1 | 1 |
| 16-bit Timer/Counter type B (TCB) | 1 | 1 |
| 12-bit Timer/Counter type D (TCD) | 1 | 1 |
| Real Time Counter (RTC) | 1 | 1 |
| USART | 1 | 1 |
| SPI | 1 | 1 |
| TWI ($I^2C$) | 1 | 1 |
| ADC | 1 | 1 |
| ADC channels | 6 | 6 |
| DAC | 1 | 1 |
| AC | 1 | 1 |
| Custom Logic/Configurable Lookup Tables | 1 | 1 |
| Window Watchdog | 1 | 1 |
| Event System channels | 6 | 6 |
| General purpose I/O | 6 | 6 |
| External interrupts | 6 | 6 |
| CRCSCAN | 1 | 1 |

# 2. Ordering Information

## 2.1 ATtiny212

**Table 2-1. ATtiny212 Ordering Codes**

| Ordering Code[1] | Flash | Package Type (GPC) | Leads | Power Supply | Operational Range | Carrier Type |
|---|---|---|---|---|---|---|
| ATtiny212-SSNR | 2KB | SOIC150 (SWB) | 8 | 1.8V - 5.5V | Industrial (-40°C +105°C) | Tape & Reel |
| ATtiny212-SSFR | 2KB | SOIC150 (SWB) | 8 | 1.8V - 5.5V | Industrial (-40°C +125°C) | Tape & Reel |

1. Pb-free packaging complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.

## 2.2 ATtiny412

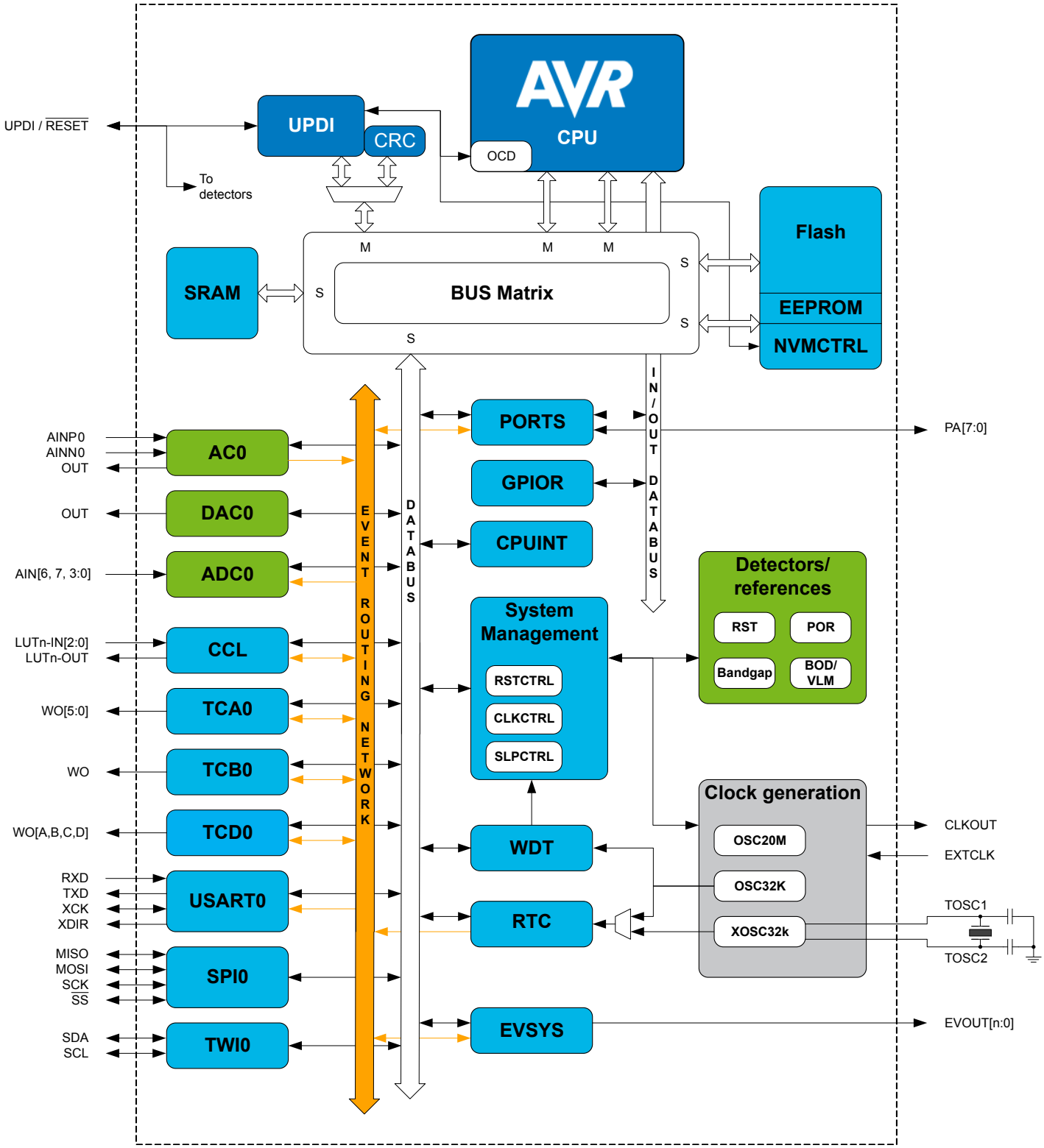
**Table 2-2. ATtiny412 Ordering Codes**

| Ordering Code[1] | Flash | Package Type (GPC) | Leads | Power Supply | Operational Range | Carrier Type |
|---|---|---|---|---|---|---|
| ATtiny412-SSNR | 4KB | SOIC150 (SWB) | 8 | 1.8V - 5.5V | Industrial (-40°C +105°C) | Tape & Reel |
| ATtiny412-SSFR | 4KB | SOIC150 (SWB) | 8 | 1.8V - 5.5V | Industrial (-40°C +125°C) | Tape & Reel |

1. Pb-free packaging complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.

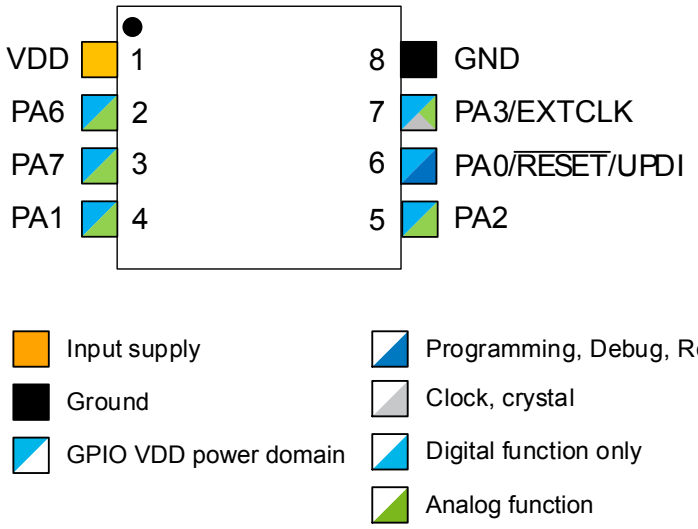
# 3. Block Diagram

**Figure 3-1. ATtiny212 / ATtiny412 Block Diagram**



# 4. Pinout

## 4.1 8-pin SOIC



# 5. I/O Multiplexing and Considerations

## 5.1 Multiplexed Signals

**Table 5-1. PORT Function Multiplexing**

| SOIC 8-pin | Pin Name (1,2) | Other/ Special | ADC0 | AC0 | DAC0 | USART0 | SPI0 | TWI0 | TCA0 | TCB0 | TCD0 | CCL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | PA0 | $\overline{\text{RESET}}$ UPDI | AIN0 | | | | | | | | | LUT0-IN0 |
| 4 | PA1 | BREAK | AIN1 | | | TXD | MOSI | SDA | | | | LUT0-IN1 |
| 5 | PA2 | EVOUT0 | AIN2 | | | RxD | MISO | SCL | | | | LUT0-IN2 |
| 7 | PA3 | EXTCLK | AIN3 | | | XCK | SCK | | WO3 | | | |
| 8 | GND | | | | | | | | | | | |
| 1 | VDD | | | | | | | | | | | |
| 2 | PA6 | | AIN6 | AINN0 | OUT | | | | | | | |
| 3 | PA7 | | AIN7 | AINP0 | | | | | | | | LUT1-OUT |

**Note:**

1. Pins names are of type P*xn*, with *x* being the PORT instance (A,B) and *n* the pin number. Notation for signals is PORT*x*_PIN*n*. All pins can be used as event input.
2. All pins can be used for external interrupt, where pins P*x*2 and P*x*6 of each port have full asynchronous detection.

**Tip:** Signals on alternative pin locations are in `typewriter` font.

# 6. Memories

## 6.1 Overview

The main memories are SRAM data memory, EEPROM data memory, and Flash program memory. In addition, the peripheral registers are located in the I/O memory space.

**Table 6-1. Physical Properties of Flash Memory**

| Property | ATtiny212 | ATtiny412 |
|---|---|---|
| Size | 2KB | 4KB |
| Page size | 64B | 64B |
| Number of pages | 32 | 64 |
| Start address | 0x8000 | 0x8000 |

**Table 6-2. Physical Properties of SRAM**

| Property | ATtiny212 | ATtiny412 |
|---|---|---|
| Size | 128B | 256B |
| Start address | 0x3F80 | 0x3F00 |

**Table 6-3. Physical Properties of EEPROM**

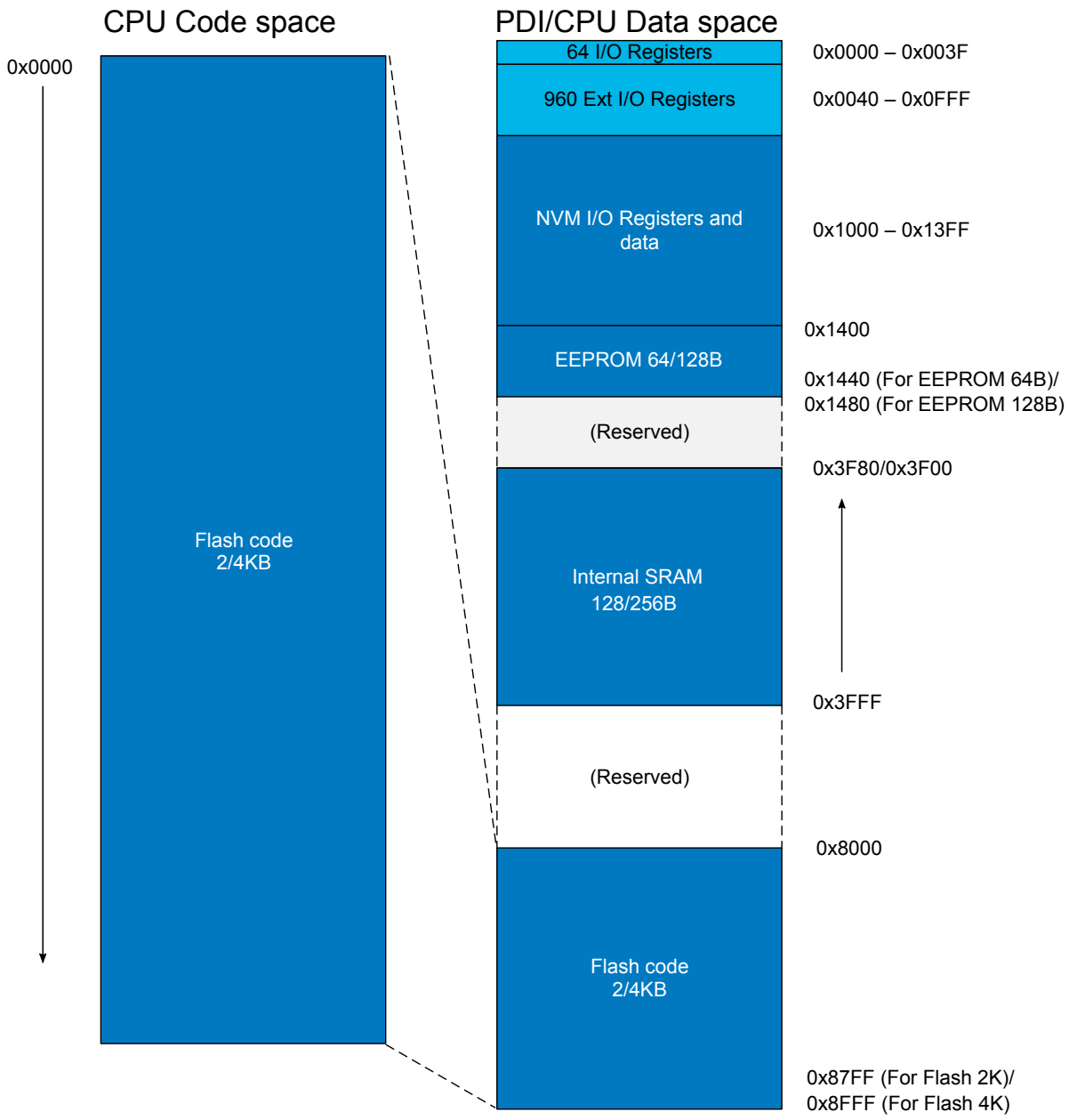
| Property | ATtiny212 | ATtiny412 |
|---|---|---|
| Size | 64B | 128B |
| Page size | 32B | 32B |
| Number of pages | 2 | 4 |
| Start address | 0x1400 | 0x1400 |

**Related Links**

I/O Memory

## 6.2 Memory Map

**Figure 6-1. Memory Map: Flash 2/4KB, Internal SRAM 128/256B, EEPROM 64/128B**




## 6.3 In-System Reprogrammable Flash Program Memory

The ATtiny212/412 contains 2/4KB 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 4K x 16. For write protection, the Flash Program memory space can be divided into three sections: Boot Loader section, Application code section and Application data section, with restricted access rights among them.

The program counter is 11 bits wide to address the whole program memory. The procedure for writing Flash memory is described in detail in the documentation of the Non-Volatile Memory Controller (NVMCTRL) peripheral.

The entire Flash memory is mapped in the memory space and is accessible with normal LD/ST instructions as well as the LPM instruction. For LD/ST instructions, the Flash is mapped from address 0x8000. For the LPM instruction, the Flash start address is 0x0000.

The ATtiny212/412 also has a CRC module that is a master on the bus. If the CRC is configured to run in the background it will read the Flash memory and can affect the program timing.

**Related Links**

Configuration Summary

NVMCTRL - Non Volatile Memory Controller

## 6.4 SRAM Data Memory

The 128B/256B SRAM is used for data storage and stack.

**Related Links**

AVR CPU

Stack and Stack Pointer

## 6.5 EEPROM Data Memory

The ATtiny212/412 has 64/128 bytes of EEPROM data memory, see Memory Map. The EEPROM memory supports single byte read and write. The EEPROM is controlled by the Non-Volatile Memory Controller (NVMCTRL).

**Related Links**

Memory Map

NVMCTRL - Non Volatile Memory Controller

## 6.6 User Row

In addition to the EEPROM, the ATtiny212/412 has one extra page of EEPROM memory that can be used for firmware settings, the User Row (USERROW). This memory supports single byte read and write as the normal EEPROM. The CPU can write and read this memory as normal EEPROM and the UPDI can write and read it as a normal EEPROM memory if the part is unlocked. The User Row can also be written by the UPDI when the part is locked. USERROW is not affected by a chip erase.

**Related Links**

Memory Map

NVMCTRL - Non Volatile Memory Controller

UPDI - Unified Program and Debug Interface

## 6.7 Signature Bytes

All Atmel microcontrollers have a three-byte signature code which identifies the device. This code can be read in both serial and parallel mode, also when the device is locked. The three bytes reside in a separate address space. For the device the signature bytes are given in the following table.

**Table 6-4. Device ID**

| Device Name | Signature Bytes Address | | |
|---|---|---|---|
| | 0x00 | 0x01 | 0x02 |
| ATtiny212 | 0x1E | 0x91 | 0x21 |
| ATtiny412 | 0x1E | 0x92 | 0x23 |

## 6.8 I/O Memory

All ATtiny212/412 I/Os and peripherals are located in the I/O space. The I/O address range from 0x00 to 0x3F can be accessed in a single cycle using IN and OUT instructions. The Extended I/O space from 0x0040 - 0x0FFF can 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. Refer to the Instruction Set section for more details.

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 interrupt flags are cleared by writing a '1' to them. On ATtiny212/412 devices, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such interrupt flags. The CBI and SBI instructions work with registers 0x00 - 0x1F only.

### General Purpose I/O Registers

The ATtiny212/412 devices provide four General Purpose I/O Registers. These registers can be used for storing any information, and they are particularly useful for storing global variables and interrupt flags. General Purpose I/O Registers, which recide in the address range 0x1C - 0x1F, are directly bit-accessible using the SBI, CBI, SBIS, and SBIC instructions.

### Related Links

Memory Map

Peripheral Module Address Map

Instruction Set Summary

### 6.8.1 Register Summary - GPIOR

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | GPIOR0 | 7:0 | GPIOR[7:0] | | | | | | | |
| 0x01 | GPIOR1 | 7:0 | GPIOR[7:0] | | | | | | | |
| 0x02 | GPIOR2 | 7:0 | GPIOR[7:0] | | | | | | | |
| 0x03 | GPIOR3 | 7:0 | GPIOR[7:0] | | | | | | | |

### 6.8.2 Register Description - GPIOR

#### 6.8.2.1 General Purpose I/O register n

These are general purpose registers that can be used to store data, such as global variables and flags, in the bitaccessible I/O memory space.

**Name:** GPIOR
**Offset:** 0x00 + n*0x01 [n=0..3]
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | GPIOR[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 – GPIOR[7:0]: GPIO Register byte**

## 6.9 FUSES - Configuration and User Fuses

Fuses are part of the non-volatile memory and holds factory calibration data and device configuration. The fuses are available from device power-up. The fuses can be read by the CPU or the UPDI, but can only be programmed or cleared by the UPDI. The configuration and calibration values stored in the fuses are written to their respective target registers at the end of the start-up sequence.

The content of the Signature Row fuses (SIGROW) is pre-programmed, and cannot be altered. SIGROW holds information such as device ID, serial number, and calibration values.

The fuses for peripheral configuration (FUSE) are pre-programmed, but can be altered by the user. Altered values in the configuration fuses will be effective only after a Reset.

This device also provides a User Row fuse area (USERROW) that can hold application data. The USERROW can be programmed on a locked device by the UPDI. This can be used for final configuration without having programming or debugging capabilities enabled.

**Related Links**

Signature Row Description

Fuse Description

### 6.9.1 Signature Row Summary - SIGROW

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | DEVICEID0 | 7:0 | DEVICEID[7:0] | | | | | | | |
| 0x01 | DEVICEID1 | 7:0 | DEVICEID[7:0] | | | | | | | |
| 0x02 | DEVICEID2 | 7:0 | DEVICEID[7:0] | | | | | | | |
| 0x03 | SERNUM0 | 7:0 | SERNUM[7:0] | | | | | | | |
| 0x04 | SERNUM1 | 7:0 | SERNUM[7:0] | | | | | | | |
| 0x05 | SERNUM2 | 7:0 | SERNUM[7:0] | | | | | | | |
| 0x06 | SERNUM3 | 7:0 | SERNUM[7:0] | | | | | | | |
| 0x07 | SERNUM4 | 7:0 | SERNUM[7:0] | | | | | | | |
| 0x08 | SERNUM5 | 7:0 | SERNUM[7:0] | | | | | | | |
| 0x09 | SERNUM6 | 7:0 | SERNUM[7:0] | | | | | | | |
| 0x0A | SERNUM7 | 7:0 | SERNUM[7:0] | | | | | | | |
| 0x0B | SERNUM8 | 7:0 | SERNUM[7:0] | | | | | | | |
| 0x0C | SERNUM9 | 7:0 | SERNUM[7:0] | | | | | | | |
| 0x0D ... 0x1F | Reserved | | | | | | | | | |
| 0x20 | TEMPSENSE0 | 7:0 | TEMPSENSE[7:0] | | | | | | | |
| 0x21 | TEMPSENSE1 | 7:0 | TEMPSENSE[7:0] | | | | | | | |
| 0x22 | OSC16ERR3V | 7:0 | OSC16ERR3V[7:0] | | | | | | | |
| 0x23 | OSC16ERR5V | 7:0 | OSC16ERR5V[7:0] | | | | | | | |
| 0x24 | OSC20ERR3V | 7:0 | OSC20ERR3V[7:0] | | | | | | | |
| 0x25 | OSC20ERR5V | 7:0 | OSC20ERR5V[7:0] | | | | | | | |

### 6.9.2 Signature Row Description

#### 6.9.2.1 Device ID n

Each device has a Device ID, identifying the device and its properties, such as memory sizes, pin count, and die revision. This can be used to identify a device and hence, the available features by software. The Device ID consists of three bytes: SIGROW.DEVICEID[2:0].

**Name:** DEVICEIDn
**Offset:** 0x00 + n*0x01 [n=0..2]
**Reset:** [Device ID]
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DEVICEID[7:0] | | | | | | | |
| Access | R | R | R | R | R | R | R | R |
| Reset | x | x | x | x | x | x | x | x |

**Bits 7:0 – DEVICEID[7:0]: Byte n of the Device ID**

#### 6.9.2.2 Serial Number Byte n

Each device has an individual serial number, representing a unique ID. This can be used to identify a specific device in the field. The serial number consists of ten bytes: SIGROW.SERNUM[9:0].

**Name:** SERNUMn
**Offset:** 0x03 + n*0x01 [n=0..9]
**Reset:** [device serial number]
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | SERNUM[7:0] | | | | | | | |
| Access | R | R | R | R | R | R | R | R |
| Reset | x | x | x | x | x | x | x | x |

**Bits 7:0 – SERNUM[7:0]: Serial Number Byte n**

#### 6.9.2.3 Temperature Sensor Calibration n

These registers contain correction factors for temperature measurements by the ADC. SIGROW.TEMPSENSE0 is a correction factor for the gain/slope (unsigned), SIGROW.TEMPSENSE1 is a correction factor for the offset (signed).

**Name:** TEMPSENSEn
**Offset:** 0x20 + n*0x01 [n=0..1]
**Reset:** [Temperature sensor calibration value]
**Property:** -

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

**Bits 7:0 – TEMPSENSE[7:0]: Temperature Sensor Calibration Byte n**
Refer to Temperature Measurement for how to use the values; Signature Row Description section for location of values.

#### 6.9.2.4 OSC16 error at 3V

**Name:** OSC16ERR3V
**Offset:** 0x22
**Reset:** [Oscillator frequency error value]
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OSC16ERR3V[7:0] | | | | | | | |
| Access | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – OSC16ERR3V[7:0]: OSC16 error at 3V**
This registers contain the signed oscillator frequency error value when running at internal 16MHz at 3V, as measured during production.

#### 6.9.2.5 OSC16 error at 5V

**Name:** OSC16ERR5V
**Offset:** 0x23
**Reset:** [Oscillator frequency error value]
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OSC16ERR5V[7:0] | | | | | | | |
| Access | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – OSC16ERR5V[7:0]: OSC16 error at 5V**
This registers contain the signed oscillator frequency error value when running at internal 16MHz at 5V, as measured during production.

#### 6.9.2.6 OSC20 error at 3V

**Name:** OSC20ERR3V
**Offset:** 0x24
**Reset:** [Oscillator frequency error value]
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OSC20ERR3V[7:0] | | | | | | | |
| Access | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – OSC20ERR3V[7:0]: OSC20 error at 3V**
This registers contain the signed oscillator frequency error value when running at internal 20MHz at 3V, as measured during production.

#### 6.9.2.7 OSC20 error at 5V

**Name:** OSC20ERR5V
**Offset:** 0x25
**Reset:** [Oscillator frequency error value]
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OSC20ERR5V[7:0] | | | | | | | |
| Access | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:0 – OSC20ERR5V[7:0]: OSC20 error at 5V**
This registers contain the signed oscillator frequency error value when running at internal 20MHz at 5V, as measured during production.

### 6.9.3 Fuse Summary - FUSE

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | WDTCFG | 7:0 | WINDOW[3:0] | | | | PERIOD[3:0] | | | |
| 0x01 | BODCFG | 7:0 | LVL[2:0] | | | SAMPFREQ | ACTIVE[1:0] | | SLEEP[1:0] | |
| 0x02 | OSCCFG | 7:0 | OSCLOCK | | | | | | FREQSEL[1:0] | |
| 0x03 | Reserved | | | | | | | | | |
| 0x04 | TCD0CFG | 7:0 | CMPDEN | CMPCEN | CMPBEN | CMPAEN | CMPD | CMPC | CMPB | CMPA |
| 0x05 | SYSCFG0 | 7:0 | CRCSRC[1:0] | | | | RSTPINCFG[1:0] | | | EESAVE |
| 0x06 | SYSCFG1 | 7:0 | | | | | | SUT[2:0] | | |
| 0x07 | APPEND | 7:0 | APPEND[7:0] | | | | | | | |
| 0x08 | BOOTEND | 7:0 | BOOTEND[7:0] | | | | | | | |
| 0x09 | Reserved | | | | | | | | | |
| 0x0A | LOCKBIT | 7:0 | LOCKBIT[7:0] | | | | | | | |

### 6.9.4 Fuse Description

#### 6.9.4.1 Watchdog Configuration

**Name:** WDTCFG
**Offset:** 0x00
**Reset:** -
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | WINDOW[3:0] | | | | PERIOD[3:0] | | | |
| Access | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:4 – WINDOW[3:0]: Watchdog Window Timeout Period**
This value is loaded into the WINDOW bit field of the Watchdog Control A register (WDT.CTRLA) during Reset.

**Bits 3:0 – PERIOD[3:0]: Watchdog Timeout Period**
This value is loaded into the PERIOD bit field of the Watchdog Control A register (WDT.CTRLA) during Reset.

**Related Links**

Register Summary - WDT

RSTCTRL - Reset Controller

#### 6.9.4.2 BOD Configuration

The settings of the BOD will be reloaded from this Fuse after a Power-On Reset. For all other Resets, the BOD configuration remains unchanged.

**Name:** BODCFG
**Offset:** 0x01
**Reset:** -
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | LVL[2:0] | | | SAMPFREQ | ACTIVE[1:0] | | SLEEP[1:0] | |
| Access | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 7:5 – LVL[2:0]: BOD Level**
This value is loaded into the LVL bit field of the BOD Control B register (BOD.CTRLB) during Reset.

| Value | Name | Description |
|---|---|---|
| 0x0 | BODLEVEL0 | 1.8V |
| 0x1 | BODLEVEL1 | 2.15V |
| 0x2 | BODLEVEL2 | 2.60V |
| 0x3 | BODLEVEL3 | 2.95V |
| 0x4 | BODLEVEL4 | 3.30V |
| 0x5 | BODLEVEL5 | 3.70V |
| 0x6 | BODLEVEL6 | 4.00V |
| 0x7 | BODLEVEL7 | 4.30V |

**Bit 4 – SAMPFREQ: BOD Sample Frequency**
This value is loaded into the SAMPFREQ bit of the BOD Control A register (BOD.CTRLA) during Reset.

| Value | Description |
|---|---|
| 0x0 | Sample frequency is 1kHz |
| 0x1 | Sample frequency is 125Hz |

**Bits 3:2 – ACTIVE[1:0]: BOD Operation Mode in Active and Idle**
This value is loaded into the ACTIVE bit field of the BOD Control A register (BOD.CTRLA) during Reset.

| Value | Description |
|---|---|
| 0x0 | Disabled |
| 0x1 | Enabled |
| 0x2 | Sampled |
| 0x3 | Enabled with wake-up halted until BOD is ready |

**Bits 1:0 – SLEEP[1:0]: BOD Operation Mode in Sleep**
This value is loaded into the SLEEP bit field of the BOD Control A register (BOD.CTRLA) during Reset.

| Value | Description |
|---|---|
| 0x0 | Disabled |
| 0x1 | Enabled |
| 0x2 | Sampled |
| 0x3 | Reserved |

**Related Links**
Register Summary - BOD

RSTCTRL - Reset Controller

#### 6.9.4.3 Oscillator Configuration

**Name:** OSCCFG
**Offset:** 0x02
**Reset:** -
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OSCLOCK | | | | | | FREQSEL[1:0] | |
| Access | R | | | | | | R | R |
| Reset | 0 | | | | | | 1 | 0 |

**Bit 7 – OSCLOCK: Oscillator Lock**
This fuse bit is loaded to LOCK in CLKCTRL.OSC20MCALIBB during reset.

| Value | Description |
|---|---|
| 0 | Calibration registers of the 20 MHz oscillator are accessible |
| 1 | Calibration registers of the 20 MHz oscillator are locked |

**Bits 1:0 – FREQSEL[1:0]: Frequency Select**
These bits selects the operation frequency of the 16/20MHz internal oscillator (OSC20M), and determine the respective factory calibration values to be written to CAL20M in CLKCTRL.OSC20MCALIBA and TEMPCAL20M in CLKCTRL.OSC20MCALIBB.

| Value | Description |
|---|---|
| 0x1 | Run at 16MHz with corresponding factory calibration |
| 0x2 | Run at 20MHz with corresponding factory calibration |
| Other | Reserved |

**Related Links**
Register Summary - CLKCTRL
RSTCTRL - Reset Controller

#### 6.9.4.4 Timer Counter Type D Configuration

The bit values of this fuse register are written to the corresponding bits in the TCD.FAULTCTRL register of TCD0 at start-up.
The CMPEN and CMP settings of the TCD will only be reloaded from the FUSE values after a Power-On Reset. For all other resets the corresponding TCD settings of the device will remain unchanged.

**Name:** TCD0CFG
**Offset:** 0x04
**Reset:** -
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CMPDEN | CMPCEN | CMPBEN | CMPAEN | CMPD | CMPC | CMPB | CMPA |
| Access | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 4, 5, 6, 7 – CMPAEN, CMPBEN, CMPCEN, CMPDEN: Compare x Enable**

| Value | Description |
|---|---|
| 0 | Compare x output on Pin is disabled |
| 1 | Compare x output on Pin is enabled |

**Bits 0, 1, 2, 3 – CMPA, CMPB, CMPC, CMPD: Compare x**

This bit selects the default state of Compare x after Reset, or when entering debug if FAULTDET is '1'.

| Value | Description |
|---|---|
| 0 | Compare x default state is 0 |
| 1 | Compare x default state is 1 |

**Related Links**

Register Summary - TCD

RSTCTRL - Reset Controller

### 6.9.4.5 System Configuration 0

**Name:** SYSCFG0
**Offset:** 0x05
**Reset:** 0xC4
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CRCSRC[1:0] | | | | RSTPINCFG[1:0] | | | EESAVE |
| Access | R | R | | | R | R | | R |
| Reset | 1 | 1 | | | 0 | 1 | | 0 |

**Bits 7:6 – CRCSRC[1:0]: CRC Source**

See CRC description for more information about the functionality.

| Value | Name | Description |
|---|---|---|
| 00 | FLASH | CRC of full Flash (boot, application code and application data) |
| 01 | BOOT | CRC of boot section |
| 10 | BOOTAPP | CRC of application code and boot sections |
| 11 | NOCRC | No CRC |

**Bits 3:2 – RSTPINCFG[1:0]: Reset Pin Configuration**

These bits select the Reset/UPDI pin configuration.

| Value | Description |
|---|---|
| 0x0 | GPIO |
| 0x1 | UPDI |
| 0x2 | RESET |

| Value | Description |
|---|---|
| 0x3 | Reserved |

**Bit 0 – EESAVE: EEPROM Save during chip erase**
If the device is locked the EEPROM is always erased by a chip erase, regardless of this bit.

| Value | Description |
|---|---|
| 0 | EEPROM erased during chip erase |
| 1 | EEPROM not erased under chip erase |

#### 6.9.4.6 System Configuration 1

**Name:** SYSCFG1
**Offset:** 0x06
**Reset:** -
**Property:** -

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

**Bits 2:0 – SUT[2:0]: Start Up Time Setting**
These bits selects the start-up time between power-on and code execution.

| Value | Description |
|---|---|
| 0x0 | 0ms |
| 0x1 | 1ms |
| 0x2 | 2ms |
| 0x3 | 4ms |
| 0x4 | 8ms |
| 0x5 | 16ms |
| 0x6 | 32ms |
| 0x7 | 64ms |

#### 6.9.4.7 Application Code End

**Name:** APPEND
**Offset:** 0x07
**Reset:** -
**Property:** -

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

**Bits 7:0 – APPEND[7:0]: Application Code Section End**

These bits set the end of the application code section in blocks of 256 bytes. The end of the application code section should be set as BOOT size + application code size. The remaining Flash will be application data. A value of 0x00 defines the Flash from BOOTEND*256 to end of Flash as application code. When both FUSE.APPEND and FUSE.BOOTEND are 0x00, the entire Flash is BOOT section.

**Related Links**

NVMCTRL - Non Volatile Memory Controller

Flash

### 6.9.4.8 Boot End

**Name:** BOOTEND
**Offset:** 0x08
**Reset:** -
**Property:** -

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

**Bits 7:0 – BOOTEND[7:0]: Boot Section End**

These bits set the end of the boot section in blocks of 256 bytes. A value of 0x00 defines the whole Flash as BOOT section. When both FUSE.APPEND and FUSE.BOOTEND are 0x00, the entire Flash is BOOT section.

**Related Links**

NVMCTRL - Non Volatile Memory Controller

Flash

### 6.9.4.9 Lock Bits

**Name:** LOCKBIT
**Offset:** 0x0A
**Reset:** -
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | LOCKBIT[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 – LOCKBIT[7:0]: Lock Bits**
When the part is locked, UPDI cannot access the system bus, so it cannot read out anything but CS-space.

| Value | Description |
|---|---|
| 0xC5 | The device is open |
| other | The device is locked |

# 7. Peripherals and Architecture

## 7.1 Peripheral Module Address Map

The address map show the base address for each peripheral. For complete register description and summary for each peripheral module, refer to the respective module chapters.

**Table 7-1. Peripheral Module Address Map**

| Base Address | Name | Description |
|---|---|---|
| 0x0000 | VPORTA | Virtual Port A |
| 0x001C | GPIO | General Purpose IO registers |
| 0x0030 | CPU | CPU |
| 0x0040 | RSTCTRL | Reset Controller |
| 0x0050 | SLPCTRL | Sleep Controller |
| 0x0060 | CLKCTRL | Clock Controller |
| 0x0080 | BOD | Brown-Out Detector |
| 0x00A0 | VREF | Voltage Reference |
| 0x0100 | WDT | Watchdog Timer |
| 0x0110 | CPUINT | Interrupt Controller |
| 0x0120 | CRCSCAN | Cyclic Redundancy Check Memory Scan |
| 0x0140 | RTC | Real Time Counter |
| 0x0180 | EVSYS | Event System |
| 0x01C0 | CCL | Configurable Custom Logic |
| 0x0200 | PORTMUX | Port Multiplexer |
| 0x0400 | PORTA | Port A Configuration |
| 0x0600 | ADC0 | Analog to Digital Converter/Peripheral Touch Controller |
| 0x06A0 | DAC0 | Digital to Analog Converter 0 |
| 0x0670 | AC0 | Analog Comparator |
| 0x0680 | DAC0 | Digital to Analog Converter |
| 0x0800 | USART0 | Universal Synchronous Asynchronous Receiver Transmitter |
| 0x0810 | TWI0 | Two Wire Interface |
| 0x0820 | SPI0 | Serial Peripheral Interface |
| 0x0A00 | TCA0 | Timer/Counter Type A instance 0 |
| 0x0A40 | TCB0 | Timer/Counter Type B instance 0 |
| 0x0A80 | TCD0 | Timer/Counter Type D instance 0 |

© 2017 Microchip Technology Inc.

| Base Address | Name | Description |
|---|---|---|
| 0x0F00 | SYSCFG | System Configuration |
| 0x1000 | NVMCTRL | Non Volatile Memory Controller |
| 0x1100 | SIGROW | Signature Row |
| 0x1280 | FUSES | Device specific fuses |
| 0x1300 | USERROW | User Row |

## 7.2 Interrupt Vector Mapping

Each of the 26 interrupt vectors is connected to one peripheral instance, as shown in the table below. A peripheral can have one or more interrupt sources, see the 'Interrupt' section in the 'Functional Description' of the respective peripheral for more details on the available interrupt sources.

When the interrupt condition occurs, an Interrupt Flag (*name*IF) is set in the Interrupt Flags register of the peripheral (*peripheral*.INTFLAGS).

An interrupt is enabled or disabled by writing to the corresponding Interrupt Enable bit (*name*IE) in the peripheral's Interrupt Control register (*peripheral*.INTCTRL).

The naming of the registers may vary slightly in some peripherals.

An interrupt request is generated when the corresponding interrupt is enabled and the Interrupt Flag is set. The interrupt request remains active until the Interrupt Flag is cleared. See the peripheral's INTFLAGS register for details on how to clear Interrupt Flags.

Interrupts must be enabled globally for interrupt requests to be generated.

**Table 7-2. Interrupt Vector Mapping**

| Vector Number | Peripheral Source | Definition |
|---|---|---|
| 0 | RESET | RESET |
| 1 | CRCSCAN_NMI | NMI - Non-Maskable Interrupt from CRC |
| 2 | BOD_VLM | VLM - Voltage Level Monitor |
| 3 | PORTA_PORT | PORTA - Port A |
| 6 | RTC_CNT | RTC - Real Time Counter |
| 7 | RTC_PIT | PIT - Periodic Interrupt Timer (in RTC peripheral) |
| 8 | TCA0_LUNF / TCA0_OVF | TCA0 - Timer Counter Type A, LUNF / OVF |
| 9 | TCA0_HUNF | TCA0, HUNF |
| 10 | TCA0_LCMP0 / TCA0_CMP0 | TCA0, LCMP0 / CMP0 |
| 11 | TCA0_LCMP1 / TCA0_CMP1 | TCA0, LCMP1 / CMP1 |
| 12 | TCA0_CMP2 / TCA0_LCMP2 | TCA0, LCMP2 / CMP2 |
| 13 | TCB0_INT | TCB0 - Timer Counter Type B |
| 14 | TCD0_OVF | TCD0 - Timer Counter Type D, OVF |
| 15 | TCD0_TRIG | TCD0, TRIG |

© 2017 Microchip Technology Inc.

| Vector Number | Peripheral Source | Definition |
|---|---|---|
| 16 | AC0_AC | AC0 – Analog Comparator |
| 17 | ADC0_RESRDY | ADC0 – Analog-to-Digital Converter, RESRDY |
| 18 | ADC0_WCOMP | ADC0, WCOMP |
| 19 | TWI0_TWIS | TWI0 - Two Wire Interface / I2C, TWIS |
| 20 | TWI0_TWIM | TWI0, TWIM |
| 21 | SPI0_INT | SPI0 - Serial Peripheral Interface |
| 22 | USART0_RXC | USART0 - Universal Asynchronous Receiver-Transmitter, RXC |
| 23 | USART0_DRE | USART0, DRE |
| 24 | USART0_TXC | USART0, TXC |
| 25 | NVMCTRL_EE | NVM - Non Volatile Memory |

**Related Links**

NVMCTRL - Non Volatile Memory Controller
PORT - I/O Pin Configuration
RTC - Real Time Counter
SPI - Serial Peripheral Interface
USART - Universal Synchronous and Asynchronous Receiver and Transmitter
TWI - Two Wire Interface
CRCSCAN - Cyclic Redundancy Check Memory Scan
TCA - 16-bit Timer/Counter Type A
TCB - 16-bit Timer/Counter Type B
TCD - 12-bit Timer/Counter Type D
AC – Analog Comparator
ADC - Analog to Digital Converter

## 7.3 SYSCFG - System Configuration

The System Configuration contains the revision ID of the part. The Revision ID is readable from the CPU, making it useful for implementing application changes between part revisions.

### 7.3.1 Register Summary - SYSCFG

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x01 | REVID | 7:0 | REVID[7:0] | | | | | | | |

### 7.3.2 Register Description - SYSCFG

#### 7.3.2.1 Device Revision ID Register

This register is read only and give the device revision ID.

**Name:** REVID
**Offset:** 0x01
**Reset:** [revision ID]
**Property:** -

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

**Bits 7:0 – REVID[7:0]: Revision ID**
These bits contain the device revision. 0x00 = A, 0x01 = B, and so on.

# 8. AVR CPU

## 8.1 Features

- 8-bit, high-performance AVR RISC CPU
  - 135 instructions
  - Hardware multiplier
- 32 8-bit registers directly connected to the ALU
- Stack in RAM
- Stack pointer accessible in I/O memory space
- Direct addressing of up to 64KB of unified memory
  - Entire Flash accessible with all LD/ST instructions
- True 16-bit access to 16-bit I/O registers
- Efficient support for 8-, 16-, and 32-bit arithmetic
- Configuration Change Protection for system-critical features

## 8.2 Overview

All AVR devices use the 8-bit AVR CPU. The CPU is able to access memories, perform calculations, control peripherals, and execute instructions in the program memory. Interrupt handling is described in a separate section.

**Related Links**

Memories

NVMCTRL - Non Volatile Memory Controller

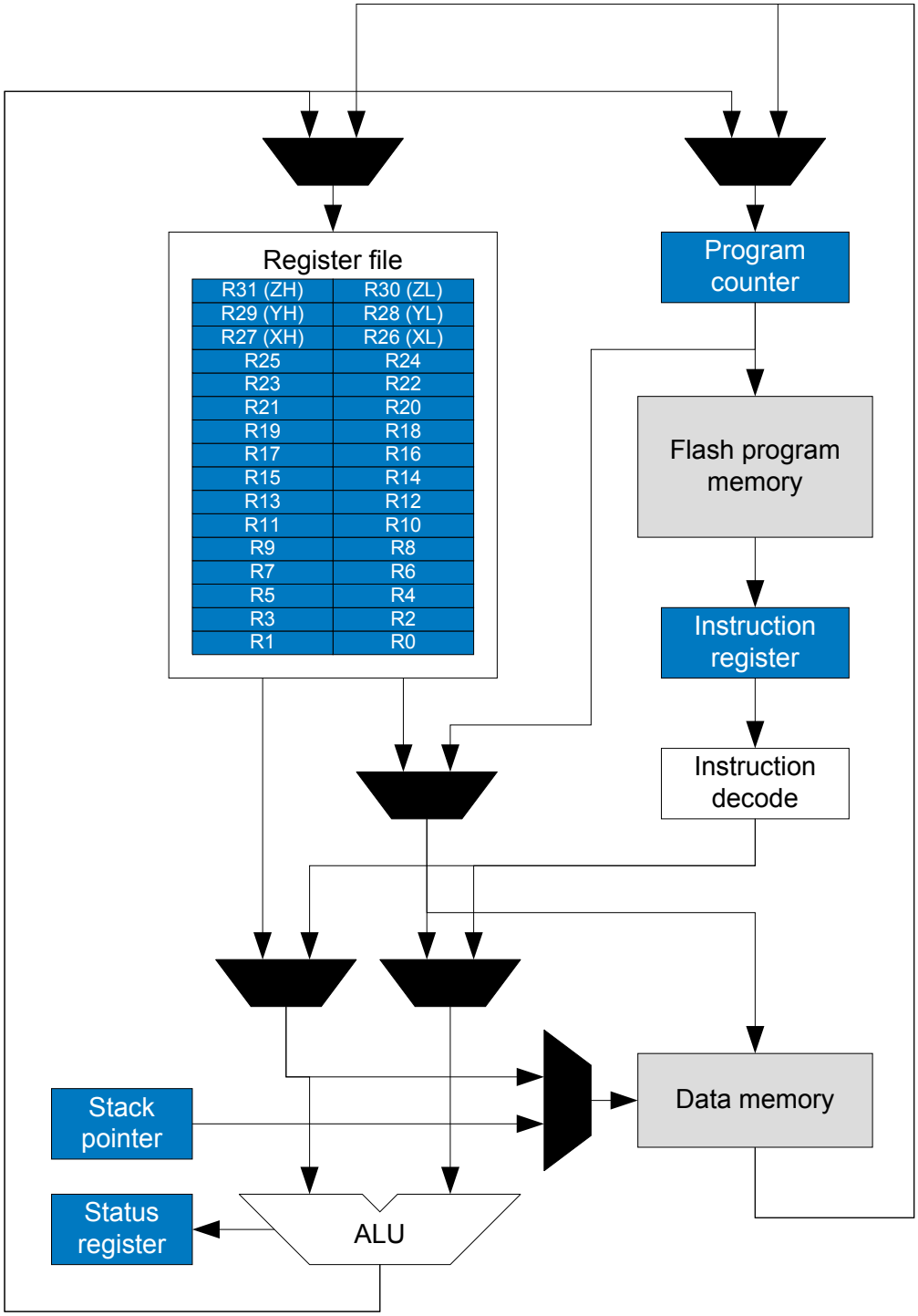
CPUINT - CPU Interrupt Controller

## 8.3 Architecture

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

© 2017 Microchip Technology Inc.

**Figure 8-1. AVR CPU Architecture**




The Arithmetic Logic Unit (ALU) supports arithmetic and logic operations between registers or between a constant and a register. Single-register operations can also be executed in the ALU. After an arithmetic operation, the status register is updated to reflect information about the result of the operation.

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

The program memory bus is connected to Flash, and the first program memory Flash address is 0x0000.

The data memory space is divided into I/O registers, SRAM, EEPROM and Flash.

All I/O status and control registers reside in the lowest 4KB addresses of the data memory. This is referred to as the I/O memory space. The lowest 64 addresses are accessed directly with single cycle IN/OUT instructions, or as the data space locations from 0x00 to 0x3F. These addresses can also be accessed using load (LD/LDS/LDD) and store (ST/STS/STD) instructions. The lowest 32 addresses can even be accessed with single cycle SBI/CBI instructions and SBIS/SBIC instructions. The rest is the extended I/O memory space, ranging from 0x0040 to 0x0FFF. I/O registers here must be accessed as data space locations using load and store instructions.

Data addresses 0x1000 to 0x1800 are reserved for memory mapping of fuses, the NVM controller and EEPROM. The addresses from 0x1800 to 0x7FFF are reserved for other memories, such as SRAM.

The Flash is mapped in the data space from 0x8000 and above. The Flash can be accessed with all load and store instructions by using addresses above 0x8000. The LPM instruction accesses the Flash as in the code space, where the Flash starts at address 0x0000.

For a summary of all AVR instructions, refer to the *Instruction Set Summary*. For details of all AVR instructions, refer to http://www.microchip.com/design-centers/8-bit.

**Related Links**

NVMCTRL - Non Volatile Memory Controller

Memories

Instruction Set Summary

## 8.4 ALU - Arithmetic Logic Unit

The Arithmetic Logic Unit supports arithmetic and logic operations between registers, or between a constant and a register. Single-register operations can also be executed.

The ALU operates in direct connection with all 32 general purpose registers. Arithmetic operations between general purpose registers or between a register and an immediate are executed in a single clock cycle, and the result is stored in the register file. After an arithmetic or logic operation, the Status register (CPU.SREG) is updated to reflect information about the result of the operation.

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

### 8.4.1 Hardware Multiplier

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

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

A multiplication takes two CPU clock cycles.

## 8.5 Functional Description

### 8.5.1 Program Flow

After Reset, the CPU will execute instructions from the lowest address in the Flash program memory, 0x0000. The program counter (PC) address the next instruction to be fetched.

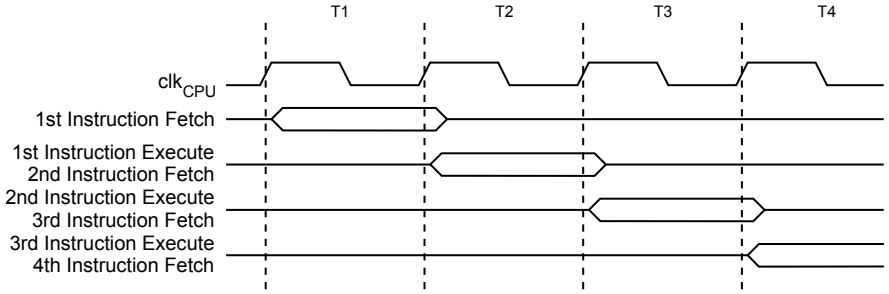
Program flow is supported by conditional and unconditional jump and call instructions, capable of addressing the whole address space directly. Most AVR instructions use a 16-bit word format, and a limited number uses a 32-bit format.

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

### 8.5.2 Instruction Execution Timing

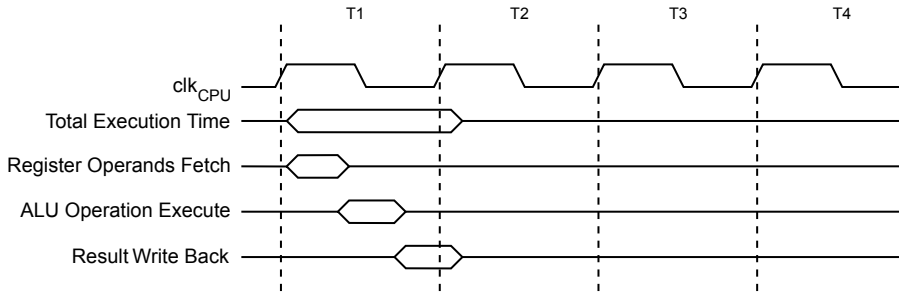
The AVR CPU is clocked by the CPU clock, CLK_CPU. No internal clock division is applied. The Figure below shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast-access register file concept. This is the basic pipelining concept enabling up to 1MIPS/MHz performance with high efficiency.

**Figure 8-2. 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 in the destination register.

**Figure 8-3. Single Cycle ALU Operation**




### 8.5.3 Status Register

The Status register (CPU.SREG) contains information about the result of the most recently executed arithmetic or logic instruction. This information can be used for altering program flow in order to perform conditional operations.

CPU.SREG is updated after all ALU operations, as specified in the Instruction Set Summary. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code. CPU.SREG is not automatically stored/restored when entering/returning from an interrupt service routine. Maintaining the status register between context switches must therefore be handled by user defined software. CPU.SREG is accessible in the I/O memory space.

**Related Links**

Instruction Set Summary

### 8.5.4 Stack and Stack Pointer

The Stack is used for storing return addresses after interrupts and subroutine calls. It can also be used for storing temporary data. The Stack Pointer (SP) always point to the top of the Stack. The SP is defined by the Stack Pointer bits (SP) in the Stack Pointer register (CPU.SP). CPU.SP is implemented as two 8-bit registers that are accessible in the I/O memory space.

Data is pushed and popped from the Stack using the PUSH and POP instructions. The Stack grows from higher to lower memory locations. This implies that pushing data onto the Stack decreases the SP, and popping data off the Stack increases the SP. The Stack Pointer is automatically set to the highest address of the internal SRAM after Reset. If the Stack is changed, it must be set to point above address 0x2000, and it must be defined before both any subroutine calls are executed and before interrupts are enabled.

During interrupts or subroutine calls, the return address is automatically pushed on the Stack as a word pointer and the SP is decremented by '2'. The return address consists of two bytes and the least significant byte is pushed on the Stack first (at the higher address). As an example, a byte pointer return address of 0x0006 is saved on the Stack as 0x0003 (shifted one bit to the right), pointing to the fourth 16-bit instruction word in the program memory. The return address is popped off the Stack with RETI (when returning from interrupts) and RET (when returning from subroutine calls) and the SP is incremented by '2'.

The SP is decremented by '1' when data is pushed on the Stack with the PUSH instruction, and incremented by '1' when data is popped off the Stack using the POP instruction.

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

### 8.5.5 Register File

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

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

Six of the 32 registers can be used as three 16-bit address register pointers for data space addressing, enabling efficient address calculations.

**Figure 8-4. AVR CPU General Purpose Working Registers**

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

The register file is located in a separate address space and is therefore not accessible trough instructions operation on data memory.

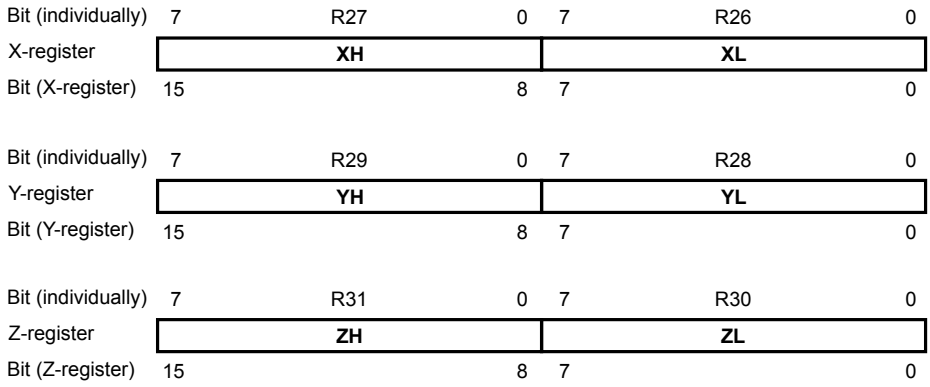
#### 8.5.5.1 The X-, Y-, and Z- Registers

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

These registers can form 16-bit address pointers for addressing data memory. These three address registers are called the X-register, Y-register, and Z-register. Load and store instructions can use all X-, Y- and Z-registers, while the LPM instructions can only use the Z-register. Indirect calls and jumps (ICALL and IJMP) also use the Z-register.

Refer to the instruction set or Instruction Set Summary for more information about how the X-, Y- and Z-registers are used.

**Figure 8-5. The X-, Y- and Z-registers**




The lowest register address holds the least-significant byte (LSB), and the highest register address holds the most-significant byte (MSB). In the different addressing modes, these address registers function as fixed displacement, automatic increment, and automatic decrement.

**Related Links**

Instruction Set Summary

### 8.5.6 Accessing 16-bit Registers

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

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

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

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

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

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

### 8.5.7 CCP - Configuration Change Protection

System critical I/O register settings are protected from accidental modification. Flash self-programming (via store to NVM controller) is protected from accidental execution. This is handled globally by the configuration change protection (CCP) register.

Changes to the protected I/O registers or bits, or execution of protected instructions, are only possible after the CPU writes a signature to the CCP register. The different signatures are listed in the description of the CCP register (CPU.CCP).

There are two modes of operation: one for protected I/O registers, and one for the protected self-programming.

**Related Links**

CCP

#### 8.5.7.1 Sequence for Write Operation to Configuration Change Protected I/O Registers

In order to write to registers protected by CCP, these steps are required:

1. The software writes the signature that enables change of protected I/O registers to the CCP bit field in the CPU.CCP register.
2. Within four instructions, the software must write the appropriate data to the protected register. Most protected registers also contain a write enable/change enable/lock bit. This bit must be written to '1' in the same operation as the data are written.

   The protected change is immediately disabled if the CPU performs write operations to the I/O register or data memory, if load or store accesses to Flash, NVMCTRL, EEPROM are conducted, or if the SLEEP instruction is executed.

#### 8.5.7.2 Sequence for Execution of Self-Programming

In order to execute self-programming (the execution of writes to the NVM controller's command register), these steps are required:

1. The software temporary enables self-programming by writing the SPM signature to the CCP register (CPU.CCP).

2. Within four instructions, the software must execute the appropriate instruction. The protected change is immediately disabled if the CPU performs accesses to the Flash, NVMCTRL, or EEPROM, or if the SLEEP instruction is executed.

Once the correct signature is written by the CPU, interrupts will be ignored for the duration of the configuration change enable period. Any interrupt request (including non-maskable interrupts) during the CCP period will set the corresponding interrupt flag as normal, and the request is kept pending. After the CCP period is completed, any pending interrupts are executed according to their level and priority.

© 2017 Microchip Technology Inc.

## 8.6 Register Summary - CPU

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x04 | CCP | 7:0 | CCP[7:0] | | | | | | | |
| 0x05<br>...<br>0x0C | Reserved | | | | | | | | | |
| 0x0D | SP | 7:0 | SP[7:0] | | | | | | | |
| 0x0E | | 15:8 | SP[15:8] | | | | | | | |
| 0x0F | SREG | 7:0 | I | T | H | S | V | N | Z | C |

## 8.7 Register Description

### 8.7.1 Configuration Change Protection

**Name:** CCP
**Offset:** 0x04
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CCP[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 – CCP[7:0]: Configuration Change Protection**
Writing the correct signature to this bit field allows changing protected I/O registers or executing protected instructions within the next four CPU instructions executed.

All interrupts are ignored during these cycles. After these cycles, interrupts will automatically be handled again by the CPU, and any pending interrupts will be executed according to their level and priority.

When the protected I/O register signature is written, CCP[0] will read as '1' as long as the CCP feature is enabled.

When the protected self-programming signature is written, CCP[1] will read as '1' as long as the CCP feature is enabled.

CCP[7:2] will always read as zero.

| Value | Name | Description |
|---|---|---|
| 0x9D | SPM | Allow Self-Programming |
| 0xD8 | IOREG | Un-protect protected I/O registers |

### 8.7.2 Stack Pointer

The CPU.SP holds the Stack Pointer (SP) that points to the top of the Stack. After Reset, the Stack Pointer points to the highest internal SRAM address.

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

The CPU.SPL and CPU.SPH register pair represents the 16-bit value, CPU.SP. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

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

**Name:** SP
**Offset:** 0x0D
**Reset:** 0xxxxx
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | SP[15:8] | | | | | | | |
| 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 |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | SP[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 15:8 – SP[15:8]: Stack Pointer high byte**
These bits hold the MSB of the 16-bit register.

**Bits 7:0 – SP[7:0]: Stack Pointer low byte**
These bits hold the LSB of the 16-bit register.

### 8.7.3 Status Register

The Status register contains information about the result of the most recently executed arithmetic or logic instruction. For details about the bits in this register and how they are affected by the different instructions, see the *Instruction Set Summary*.

**Name:** SREG
**Offset:** 0x0F
**Reset:** 0x00
**Property:** -

| 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**
Writing a '1' to this bit enable interrupts on the device.

Writing a '0' to this bit disables interrupts on the device, independent of the individual interrupt enable settings of the peripherals.

This bit is not cleared by hardware after an interrupt has occurred.

This bit can be set and cleared by software with the SEI and CLI instructions.

Changing the I flag through the I/O-register result in a one-cycle wait state on the access.

**Bit 6 – T: Bit Copy Storage**
The bit copy instructions bit load (`BLD`) and bit store (`BST`) use the T bit as source or destination for the operated bit.

A bit from a register in the register file can be copied into this bit by the `BST` instruction, and this bit can be copied into a bit in a register in the register file by the `BLD` instruction.

**Bit 5 – H: Half Carry Flag**
This bit indicates a half carry in some arithmetic operations. Half carry is useful in BCD arithmetic.

**Bit 4 – S: Sign Bit, S = N ⊕ V**
The sign bit (S) is always an exclusive or (*xor*) between the negative flag (N) and the two's complement overflow flag (V).

**Bit 3 – V: Two's Complement Overflow Flag**
The two's complement overflow flag (V) supports two's complement arithmetic.

**Bit 2 – N: Negative Flag**
The negative flag (N) indicates a negative result in an arithmetic or logic operation.

**Bit 1 – Z: Zero Flag**
The zero flag (Z) indicates a zero result in an arithmetic or logic operation.

**Bit 0 – C: Carry Flag**
The carry flag (C) indicates a carry in an arithmetic or logic operation.

© 2017 Microchip Technology Inc.

# 9. NVMCTRL - Non Volatile Memory Controller

## 9.1 Features

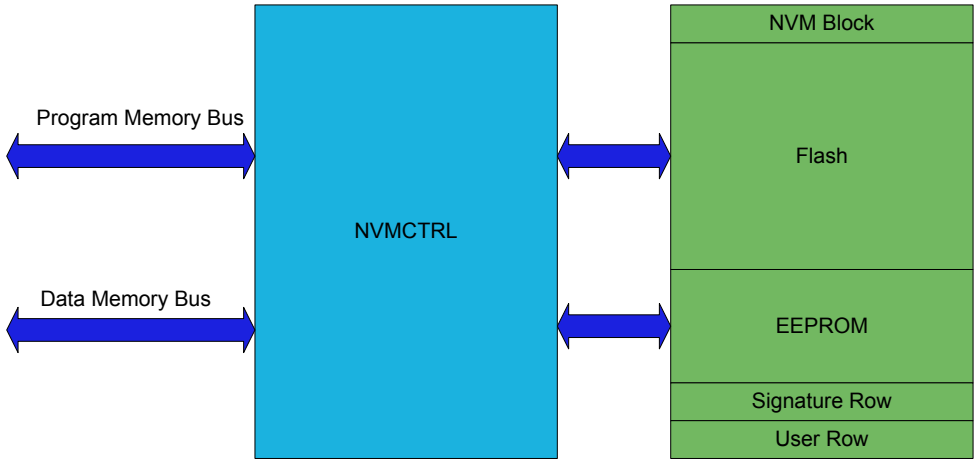
- Unified memory
- In-system programmable
- Self-programming and boot loader support
- Configurable sections for write protection:
  - Boot section for boot loader code or application code
  - Application code section for application code
  - Application data section for application code or data storage
- Signature Row for factory-programmed data:
  - ID for each device type
  - Serial number for each device
  - Calibration bytes for factory calibrated peripherals
- User Row for application data:
  - 32 bytes in size
  - Can be read and written from software
  - Can be written from UPDI on locked device
  - Content is kept after chip erase

## 9.2 Overview

The NVM Controller (NVMCTRL) is the interface between the device, the Flash, and the EEPROM. The Flash and EEPROM are reprogrammable memory blocks that retain their values even when not powered. The Flash is mainly used for program storage, but can also be used for data storage. The EEPROM is used for data storage and can be programmed while the CPU is running the program from the Flash.

### 9.2.1 Block Diagram

**Figure 9-1. NVMCTRL Block Diagram**




### 9.2.2 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 9-1. NVMCTRL System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | No | - |
| Interrupts | Yes | CPUINT |
| Events | No | - |
| Debug | Yes | UPDI |

**Related Links**

Clocks

Debug Operation

Interrupts

#### 9.2.2.1 Clocks

This peripheral always runs on the CPU clock (CLK_CPU). It will request this clock also in sleep modes if a write/erase is ongoing.

**Related Links**

CLKCTRL - Clock Controller

#### 9.2.2.2 I/O Lines and Connections

Not applicable.

#### 9.2.2.3 Interrupts

Using the interrupts of this peripheral requires the Interrupt Controller to be configured first.

#### 9.2.2.4 Events

Not applicable.

#### 9.2.2.5 Debug Operation

When run-time debugging, this peripheral will continue normal operation. Halting the CPU in debugging mode will halt normal operation of the peripheral.

If the peripheral is configured to require periodical service by the CPU through interrupts or similar, improper operation or data loss may result during halted debugging.

**Related Links**

UPDI - Unified Program and Debug Interface

## 9.3 Functional Description

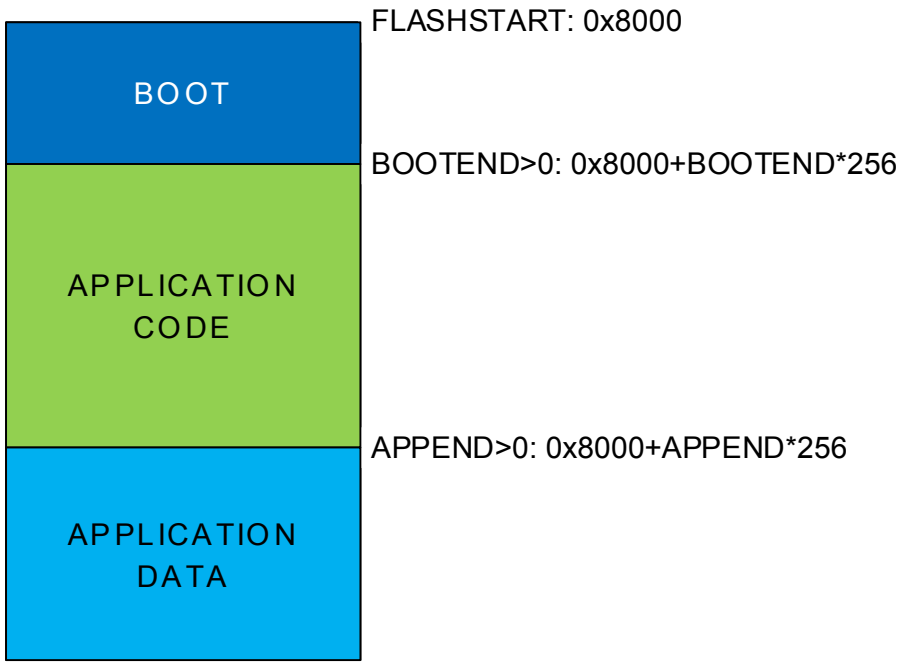
### 9.3.1 Memory Organization

#### 9.3.1.1 Flash

The Flash is divided into a set of pages. A page is the basic unit addressed when programming the Flash. It is only possible to write or erase a whole page at a time. One page consists of several words.

The Flash can be divided into three sections in blocks of 256 bytes for different security. The three different sections are BOOT, application code (APPCODE) and application data (APPDATA).

**Figure 9-2. Flash Sections**




**Section Sizes**

The sizes of these sections are set by the Boot Section End fuse (FUSE.BOOTEND) and Application Code Section End fuse (FUSE.APPEND).

The fuses select the section sizes in blocks of 256 bytes. As shown in Figure 9-2, the BOOT section stretches from the start of the Flash until BOOTEND. The APPCODE section runs from BOOTEND until APPEND. The remaining area is the APPDATA section. If APPEND is written to 0, the APPCODE section runs from BOOTEND to the end of Flash (removing the APPDATA section). If BOOTEND and APPEND are written to 0, the entire Flash is regarded as BOOT section. APPEND should either be set to 0 or a value greater or equal than BOOTEND.

**Table 9-2. Setting up Flash sections**

| BOOTEND | APPEND | BOOT section | APPCODE section | APPDATA section |
|---|---|---|---|---|
| 0 | 0 | 0 to FLASHEND | - | - |
| > 0 | 0 | 0 to 256*BOOTEND | 256*BOOTEND to FLASHEND | - |
| > 0 | == BOOTEND | 0 to 256*BOOTEND | - | 256*BOOTEND to FLASHEND |
| > 0 | > BOOTEND | 0 to 256*BOOTEND | 256*BOOTEND to 256*APPEND | 256*APPEND to FLASHEND |

**Note:**

- Also see BOOTEND and APPEND descriptions.
- Interrupt vectors are by default located after the BOOT section. This can be changed in the interrupt controller. Refer to Interrupt Vector Locations

If FUSE.BOOTEND is written to 0x04 and FUSE.APPEND is written to 0x08, the first 4*256 bytes will be BOOT, the next 4*256 bytes will be APPCODE, and the remaining Flash will be APPDATA.

**Inter-Section Write Protection**

Between the three Flash sections, a directional write protection is implemented:

- Code in the BOOT section can write to APPCODE and APPDATA.
- Code in APPCODE can write to APPDATA
- Code in APPDATA cannot write to Flash or EEPROM.

**Boot Section Lock and Application Code Section Write Protection**

The two lock bits (APCWP and BOOTLOCK in NVMCTRL.CTRLB) can be set to lock further updates of the respective APPCODE or BOOT section until the next Reset.

The CPU can never write to the BOOT section. NVMCTRL_CTRLB.BOOTLOCK prevents reads and execution of code from the BOOT section.

#### 9.3.1.2 EEPROM

The EEPROM is divided into a set of pages where one page consists of multiple bytes. The EEPROM has byte granularity on erase write. Within one page, only the bytes marked to be updated will be erased/written. The byte is marked by writing a new value to the page buffer for that address location.

#### 9.3.1.3 User Row

The User Row is one extra page of EEPROM. This page can be used to store various data, such as calibration/configuration data and serial numbers. This page is not erased by a chip erase. The User Row is written as normal EEPROM, but in addition it can be written through UPDI on a locked device.

### 9.3.2 Memory Access

#### 9.3.2.1 Read

Reading of the Flash and EEPROM is done by using a load instructions with address according to the memory map. Reading any of the arrays while a write or erase is in progress will result in a bus wait, and the instruction will be suspended until the ongoing operation is complete.

#### 9.3.2.2 Page Buffer Load

The page buffer is loaded by writing directly to the memories as defined in the memory map. Flash, EEPROM and User Row share the same page buffer so only one section can be programmed at one time. The least significant bits of the address are used to select where in the page buffer the data is written. The resulting data will be a binary AND operation between the new and the previous content of the page buffer. The page buffer will automatically be erased (all bits set) after:

- a device reset
- any page write or erase operation
- a clear page buffer command
- the device wakes up from any sleep mode

#### 9.3.2.3 Programming

For page programming, filling the page buffer and writing the page buffer into Flash, User Row and EEPROM are two separate operations.

Before programming a flash page with the data in the page buffer, the flash page must be erased. The page buffer is also erased when the device enter sleep mode. Programming an un-erased flash page will corrupt its content.

The flash can either be written with the erase and write separately or one command handling both:

Alternative 1:

- Fill the page buffer
- Write the page buffer to flash with the `erase-write` page command

Alternative 2:

- Write to a location in the page to set up the address
- Perform a `erase` page command
- Fill the page buffer
- Perform a `write` page command

The NVM command set supports both a single erase and write operation, and split page erase and page write commands. This split commands enable shorter programming time for each command, and the erase operations can be done during non-time-critical programming execution.

The EEPROM programming is similar, but only the bytes updated in the page buffer will be written or erased in the EEPROM.

#### 9.3.2.4 Commands

Reading of the Flash/EEPROM and writing of the page buffer is handled with normal load/store instructions. Other operations, such as writing and erasing the memory arrays, are handled by commands in the NVM.

To execute a command in the NVM:

1. Write the NVM command unlock to the Configuration Change Protection register in the CPU (CPU.CCP).
2. Write the desired command value to the CMD bits in the Control A register (NVMCTRL.CTRLA) within the next 4 instructions.

**Write Command**

The `write` command of the Flash controller writes the content of the page buffer to the Flash or EEPROM.

If the write is to the Flash, the CPU will stop executing code as long as the Flash is busy with the write operation. If the write is to the EEPROM, the CPU can continue executing code while the operation is ongoing.

The page buffer will be automatically cleared after the operation is finished.

**Erase Command**

The `erase` command erases the current page. There must be one byte written in the page buffer for the `erase` command to take effect.

For erasing the Flash, first write to one address in the desired page, then execute the command. The whole page in the Flash will then be erased. The CPU will be halted while the erase is ongoing.

For the EEPROM, only the bytes written in the page buffer will be erased when the command is executed. To erase a specific byte, write to its corresponding address before executing the command. To erase a whole page all the bytes in the page buffer has to be updated before executing the command. The CPU can continue running code while the operation is ongoing.

The page buffer will be automatically cleared after the operation is finished.

**Erase-Write Operation**

The `erase-write` command is a combination of the `erase` and `write` command, but without clearing the page buffer after the erase command: The `erase-write` operation first erases the selected page, then it writes the content of the page buffer to the same page.

When executed on the Flash, the CPU will be halted when the operations are ongoing. When executed on EEPROM, the CPU can continue executing code.

The page buffer will be automatically cleared after the operation is finished.

**Page Buffer Clear Command**

The `page buffer clear` command clears the page buffer. The contents of the page buffer will be all-ones after the operation. The CPU will be halted when the operation executes (7 CPU cycles).

**Chip Erase Command**

The `chip erase` command erases the Flash and the EEPROM. The EEPROM is unaltered if the EEPROM Save During Chip Erase (EESAVE) fuse in FUSE.SYSCFG0 is set. The memory will be all-ones after the operation.

**EEPROM Erase Command**

The `EEPROM erase` command erases the EEPROM. The EEPROM will be all-ones after the operation. The CPU will be halted while the EEPROM is being erased.

**Fuse Write Command**

The `fuse write` command writes the fuses. It can only be used by the UPDI, the CPU cannot start this command.

Follow this procedure to use this command:

- Write the address of the fuse to the Address register (NVMCTRL.ADDR)
- Write the data to be written to the fuse to the Data register (NVMCTRL.DATA)
- Execute the `fuse write` command.
- After the fuse is written, a Reset is required for the updated value to take effect.

For reading fuses, use a regular read on the memory location.

### 9.3.3 Preventing Flash/EEPROM Corruption

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

A Flash/EEPROM 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. Also, the CPU itself can execute instructions incorrectly when the supply voltage is too low. See the Electrical Characteristics chapter for Maximum Frequency vs. VDD.

Flash/EEPROM corruption can be avoided by these measures:

Keep the device in reset during periods of insufficient power supply voltage. This can be done by enabling the internal Brown-Out Detector (BOD).

The voltage level monitor in the BOD can be used to prevent starting a write to the EEPROM close to the BOD level.

If the detection levels of the internal BOD does not match the required detection level, an external low-VDD-reset protection circuit can be used. If a Reset occurs while a write operation is ongoing, the write operation will be aborted.

**Related Links**

General Operating Ratings
BOD - Brownout Detector

### 9.3.4 Interrupts

**Table 9-3. Available Interrupt Vectors and Sources**

| Offset | Name | Vector Description | Conditions |
|---|---|---|---|
| 0x00 | EEREADY | NVM | The EEPROM is ready for new write/erase operations. |

When an interrupt condition occurs, the corresponding Interrupt Flag is set in the Interrupt Flags register of the peripheral (*NVMCTRL*.INTFLAGS).

An interrupt source is enabled or disabled by writing to the corresponding bit in the peripheral's Interrupt Enable register (*NVMCTRL*.INTEN).

An interrupt request is generated when the corresponding interrupt source is enabled and the Interrupt Flag is set. The interrupt request remains active until the Interrupt Flag is cleared. See the peripheral's INTFLAGS register for details on how to clear Interrupt Flags.

### 9.3.5 Sleep Mode Operation

If there is no ongoing write operation, the NVMCTRL will enter sleep mode when the system enters sleep mode.

If a write operation is ongoing when the system enters a sleep mode, the NVM block, the NVM Controller and the system clock will remain on until the write is finished. This is valid for all sleep modes, including Power Down sleep mode.

The EEPROM Ready interrupt will wake up the device only from Idle sleep mode.

The page buffer is cleared when waking up from sleep.

### 9.3.6 Configuration Change Protection

This peripheral has registers that are under Configuration Change Protection (CCP). In order to write to these, a certain key must be written to the CPU.CCP register first, followed by a write access to the protected bits within four CPU instructions.

It is possible to try writing to these registers any time, but the values are not altered.

The following registers are under CCP:

**Table 9-4. NVMCTRL - Registers under Configuration Change Protection**

| Register | Key |
|---|---|
| NVMCTRL.CTRLA | SPM |

**Related Links**

Sequence for Execution of Self-Programming

## 9.4 Register Summary - NVMCTRL

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | | | | | | CMD[2:0] | | |
| 0x01 | CTRLB | 7:0 | | | | | | | BOOTLOCK | APCWP |
| 0x02 | STATUS | 7:0 | | | | | | WRERROR | EEBUSY | FBUSY |
| 0x03 | INTCTRL | 7:0 | | | | | | | | EEREADY |
| 0x04 | INTFLAGS | 7:0 | | | | | | | | EEREADY |
| 0x05 | Reserved | | | | | | | | | |
| 0x06 | DATA | 7:0 | DATA[7:0] | | | | | | | |
| 0x07 | | 15:8 | DATA[15:8] | | | | | | | |
| 0x08 | ADDR | 7:0 | ADDR[7:0] | | | | | | | |
| 0x09 | | 15:8 | ADDR[15:8] | | | | | | | |

## 9.5 Register Description

### 9.5.1 Control A

**Name:** CTRLA
**Offset:** 0x00
**Reset:** 0x00
**Property:** Configuration Change Protection

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

**Bits 2:0 – CMD[2:0]: Command**
Write this bit field to issue a command. The Configuration Change Protection key for self-programming (SPM) has to be written within four instructions before this write.

| Value | Name | Description |
|---|---|---|
| 0x0 | - | No command |
| 0x1 | WP | Write page buffer to memory (NVMCTRL.ADDR selects which memory) |
| 0x2 | ER | Erase page (NVMCTRL.ADDR selects which memory) |
| 0x3 | ERWP | Erase and write page (NVMCTRL.ADDR selects which memory) |
| 0x4 | PBC | Page buffer clear |
| 0x5 | CHER | Chip erase: erase Flash and EEPROM (unless EESAVE in FUSE.SYSCFG is '1') |
| 0x6 | EEER | EEPROM Erase |
| 0x7 | WFU | Write fuse (only accessible through UPDI) |

### 9.5.2 Control B

**Name:** CTRLB
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

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

**Bit 1 – BOOTLOCK: Boot Section Lock**
Writing a '1' to this bit locks the boot section from read and instruction fetch.

If this bit is '1', a read from the boot section will return '0'. A fetch from the boot section will also return 0 as instruction.

This bit can only be written from the boot section. It can only be cleared to '0' by a Reset.

This bit will only take effect when the boot section is left the first time after the bit is written.

**Bit 0 – APCWP: Application Code Section Write Protection**
Writing a '1' to this bit protects the application code section from further writes.

This bit can only be written to '1', it is cleared to '0' only by Reset.

### 9.5.3 Status

**Name:** STATUS
**Offset:** 0x02
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | WRERROR | EEBUSY | FBUSY |
| Access | | | | | | R | R | R |
| Reset | | | | | | 0 | 0 | 0 |

**Bit 2 – WRERROR: Write Error**
This bit will read '1' when a write error has happened. A write error could be writing to different sections before doing a page write or writing to a protected area. This bit is valid for the last operation.

**Bit 1 – EEBUSY: EEPROM Busy**
This bit will read '1' when the EEPROM is busy with a command.

**Bit 0 – FBUSY: Flash Busy**
This bit will read '1' when the Flash is busy with a command.

### 9.5.4 Interrupt Control

**Name:** INTCTRL
**Offset:** 0x03
**Reset:** 0x00
**Property:** -

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

**Bit 0 – EEREADY: EEPROM Ready Interrupt**
Writing a '1' to this bit enables the interrupt which indicates that the EEPROM is ready for new write/erase operations.

This is a level interrupt that will be triggered only when the EEREADY flag in the INTFLAGS register is set to zero. Thus, the interrupt should not be enabled before triggering an NVM command, as the EEREADY flag will not be set before the NVM command issued. The interrupt should be disabled in the interrupt handler.

### 9.5.5 Interrupt Flags

**Name:** INTFLAGS
**Offset:** 0x04
**Reset:** 0x00
**Property:** -

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

**Bit 0 – EEREADY: EEREADY Interrupt Flag**
Interrupt flag for the EEPROM interrupt. This bit is cleared by writing a '1' to it. When this interrupt is enabled, it will immediately request an interrupt, and it will continue to request interrupts continuously - even if no EEPROM writes are initiated.

### 9.5.6 Data

The NVMCTRL.DATAL and NVMCTRL.DATAH register pair represents the 16-bit value, NVMCTRL.DATA. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

**Name:** DATA
**Offset:** 0x06
**Reset:** 0x00
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | DATA[15:8] | | | | | | | |
| 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 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DATA[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 15:0 – DATA[15:0]: Data Register**
This register is used by the UPDI for fuse write operations.

### 9.5.7 Address

The NVMCTRL.ADDRL and NVMCTRL.ADDRH register pair represents the 16-bit value, NVMCTRL.ADDR. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

**Name:** ADDR
**Offset:** 0x08
**Reset:** 0x00
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | ADDR[15:8] | | | | | | | |
| 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 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ADDR[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 15:0 – ADDR[15:0]: Address**
The Address register contains the address to the last memory location that has been updated.

# 10. CLKCTRL - Clock Controller

## 10.1 Features

- All clocks and clock sources are automatically enabled when requested by peripherals
- Internal oscillators:
  - 16/20MHz oscillator (OSC20M)
  - 32KHz Ultra Low Power oscillator (OSCULP32K)
- External clock options:
  - 32.768kHz crystal oscillator (XOSC32K)
  - External clock
- Main clock features:
  - Safe run-time switching
  - Prescaler with 1x to 64x division in 12 different settings

## 10.2 Overview

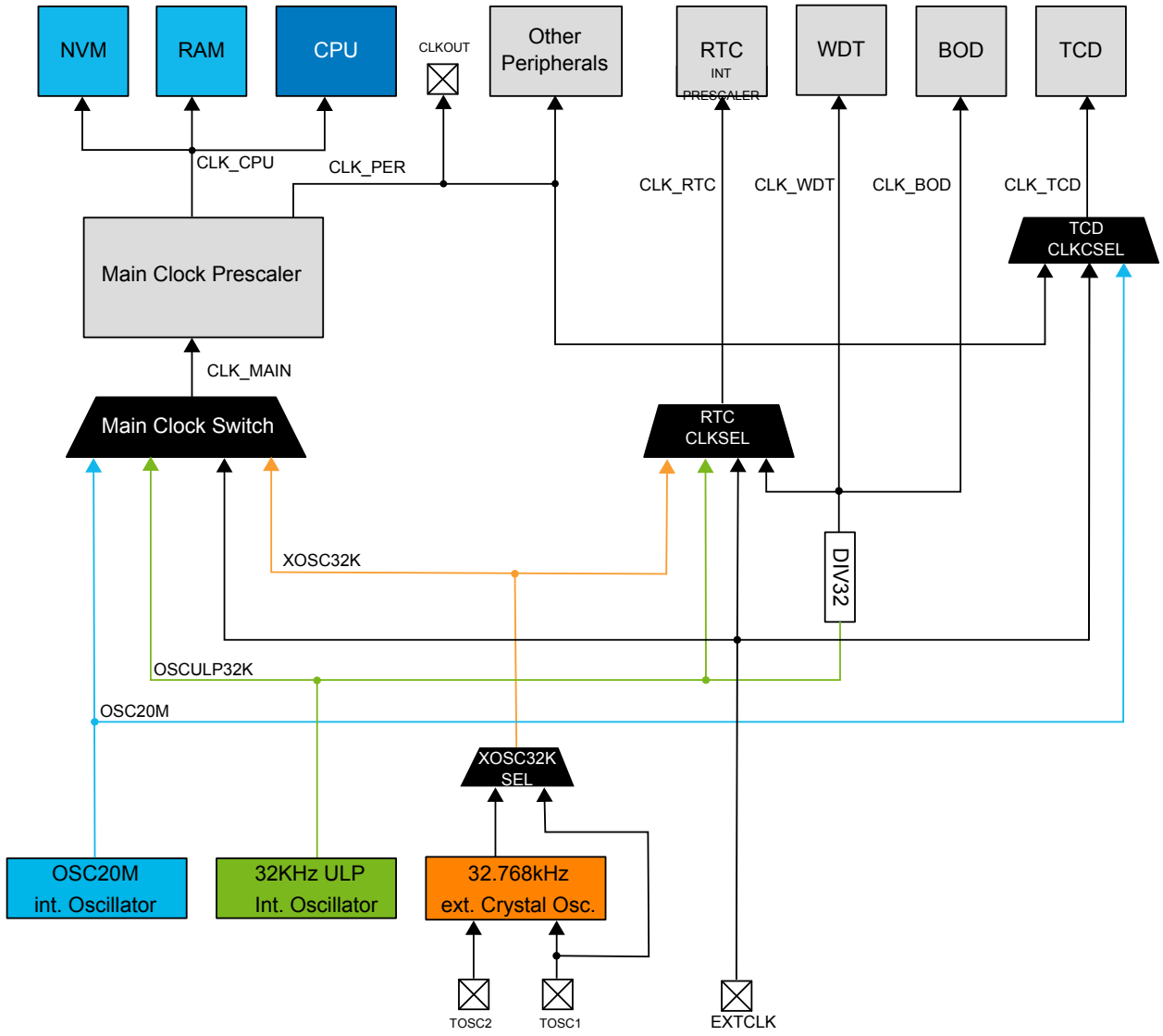
The Clock Controller peripheral (CLKCTRL) controls, distributes, and prescales the clock signals from the available oscillators. The CLKCTRL supports internal and external clock sources.

The CLKCTRL is based on an automatic clock request system, implemented in all peripherals on the device. The peripherals will automatically request the clocks needed. If multiple clock sources are available, the request is routed to the correct clock source.

The Main Clock (CLK_MAIN) is used by the CPU, RAM, and the IO bus. The Main Clock source can be selected and prescaled. Some peripherals can share the same clock source as the Main Clock, or run asynchronously to the Main Clock domain.

### 10.2.1 Block Diagram - CLKCTRL

**Figure 10-1. CLKCTRL Block Diagram**




The clock system consists of the Main Clock and other asynchronous clocks:

- Main Clock
  This clock is used by the CPU, RAM, Flash, the IO bus and all peripherals connected to the IO bus. It is always running in Active and Idle sleep mode, and can be running in Standby sleep mode if requested.

  The main clock CLK_MAIN is prescaled and distributed by the Clock Controller:
  - CLK_CPU is used by the CPU, SRAM and the NVMCTRL peripheral to access the non-volatile memory
  - CLK_PER is used by all peripherals that are not listed under asynchronous clocks.
- Clocks running asynchronously to the Main Clock domain:
  - CLK_RTC is used by the RTC/PIT. It will be requested when the RTC/PIT is enabled. The clock source for CLK_RTC should only be changed if the peripheral is disabled.
  - CLK_WDT is used by the WDT. It will be requested when the WDT is enabled.

- CLK_BOD is used by the BOD. It will be requested when the BOD is enabled in Sampled Mode.
- CLK_TCD is used by the TCD. It will be requested when the TCD is enabled. The clock source can only be changed if the peripheral is disabled.

The clock source for the for the Main Clock domain is configured by writing to the Clock Select bits (CLKSEL) in the Main Clock Control A register (CLKCTRL.MCLKCTRLA). The asynchronous clock sources are configured by registers in the respective peripheral.

### 10.2.2 Signal Description

| Signal | Type | Description |
|---|---|---|
| CLKOUT | Digital output | CLK_PER output |

**Related Links**

I/O Multiplexing and Considerations

## 10.3 Functional Description

### 10.3.1 Sleep Mode Operation

When a clock source is not used/requested it will turn off. It is possible to request an clock source directly by writing a '1' to the Run Standby bit (RUNSTDBY) in the respective oscillator's Control A register (CLKCTRL.[osc]CTRLA). This will cause the oscillator to run constantly, except for Power Down sleep mode. Additionally, when this bit is written to '1' the oscillator start-up time is eliminated when the clock source is requested by a peripheral.

The Main Clock will always run in Active and Idle sleep mode. In Standby sleep mode, the Main Clock will only run if any peripheral is requesting it, or the Run in Standby bit (RUNSTDBY) in the respective oscillator's Control A register (CLKCTRL.[osc]CTRLA) is written to '1'.

In Power Down sleep mode, the Main Clock will stop after all NVM operations are completed.

### 10.3.2 Main Clock Selection and Prescaler

All internal oscillators can be used as the Main Clock source for CLK_MAIN. The Main Clock source is selectable from software, and can be safely changed during normal operation.

Built-in hardware protection prevents unsafe clock switching:

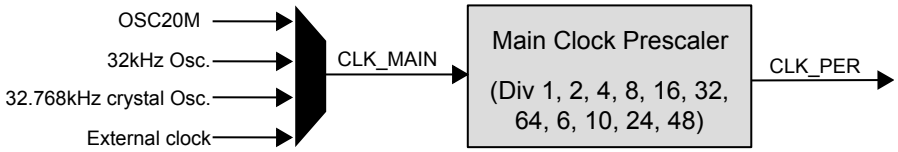
Upon selection of an external clock source, a switch to the chosen clock source will only occur if edges are detected, indicating it is stable. Until a sufficient number of clock edges are detected, the switch will not occur and it will not be possible to change to another clock source again without executing a Reset.

An ongoing clock source switch is indicated by the System Oscillator Changing flag (SOSC) in the Main Clock Status register (CLKCTRL.MCLKSTATUS). The stability of the external clock sources is indicated by the respective status flags (EXTS and XOSC32KS in CLKCTRL.MCLKSTATUS).

**Caution:** If an external clock source fails while used as CLK_MAIN source, only the WDT can provide a mechanism to switch back via System Reset.

CLK_MAIN is fed into a prescaler before it is used by the peripherals (CLK_PER) in the device. The prescaler divide CLK_MAIN by a factor from 1 to 64.

**Figure 10-2. Main Clock and Prescaler**




The Main Clock and Prescaler configuration registers (CLKCTRL.MCLKCTRLA, CLKCTRL.MCLKCTRLB) are protected by the Configuration Change Protection Mechanism, employing a timed write procedure for changing these registers.

**Related Links**

CCP - Configuration Change Protection

### 10.3.3 Main Clock after Reset

After any Reset, CLK_MAIN is provided by the 16/20MHz oscillator (OSC20M) and with a Prescaler division factor of 6. Since the actual frequency of the OSC20M is determined by the Frequency Select bits (FREQSEL) of the Oscillator Configuration fuse (FUSE.OSCCFG), these frequencies are possible after Reset:

**Table 10-1. Peripheral Clock Frequencies after Reset**

| CLK_MAIN as per FREQSEL in FUSE.OSCCFG | Resulting CLK_PER |
|---|---|
| 16MHz | 2.66MHz |
| 20MHz | 3.33MHz |

See the OSC20M description for further details.

**Related Links**

16/20MHz Oscillator (OSC20M)

### 10.3.4 Clock Sources

All internal clock sources are enabled automatically when they are requested by a peripheral. The crystal oscillator, based on an external crystal, must be enabled by writing a '1' to the ENABLE bit in the 32KHz Crystal Oscillator Control A register (CLKCTRL.XOSC32KCTRLA) before it can serve as clock source.

The respective Oscillator Status bits in the Main Clock Status register (CLKCTRL.MCLKSTATUS) indicate whether the clock source is running and stable.

**Related Links**

FUSES - Configuration and User Fuses

CCP - Configuration Change Protection

#### 10.3.4.1 Internal Oscillators

The internal oscillators do not require any external components to run. See the related links for accuracy and electrical characteristics.

**16/20MHz Oscillator (OSC20M)**

This oscillator can operate at multiple frequencies, selected by the value of the Frequency Select bits (FREQSEL) in the Oscillator Configuration fuse (FUSE.OSCCFG). The center frequencies are:

- 16MHz
- 20MHz

After a System Reset, FUSE.OSCCFG determines the initial frequency of CLK_MAIN.

During Reset the calibration values for the OSC20M are loaded from fuses . There are two different calibration bit fields. The Calibration bit field (CAL20M) in the Calibration A register (CLKCTRL.OSC20MCALIBA) enables calibration around the current center frequency. The Oscillator Temperature Coefficient Calibration bit field (TEMPCAL20M) in the Calibration B register (CLKCTRL.OSC20MCALIBB) enables adjustment of the slope of the temperature drift compensation.

For applications requiring more fine-tuned frequency setting than the oscillator calibration provides, factory stored frequency error after calibrations are available.

The oscillator calibration can be locked by the Oscillator Lock (OSCLOCK) fuse (FUSE.OSCCFG). When this fuse is '1', it is not possible to change the calibration. The calibration is also locked if this oscillator is used as Main Clock source and the Lock Enable bit (LOCKEN) in the Control B register (CLKCTRL.OSC20MCALIBB) is '1'.

The calibration bits are also protected by the Configuration Change Protection Mechanism, requiring a timed write procedure for changing the Main Clock and Prescaler settings.

The start-up time of this oscillator is analog start-up time plus 4 oscillator cycles. Refer to Electrical Characteristics chapter for the start-up time.

When changing oscillator calibration value, the frequency may overshoot. If the oscillator is used as the main clock (CLK_MAIN) it is recommended to change the main clock prescaler so that the main clock frequency does not exceed ¼ of the maximum operation main clock frequency as described in the General Operating Ratings. The system clock prescaler can be changed back after the oscillator calibration value has been updated.

**Related Links**

FUSES - Configuration and User Fuses
Configuration Change Protection
General Operating Ratings
Main Clock after Reset
Main Clock after Reset
Oscillators and Clocks

***OSC20M Stored Frequency Error Compensation***

This oscillator can operate at multiple frequencies, selected by the value of the Frequency Select bits (FREQSEL) in the Oscillator Configuration fuse (FUSE.OSCCFG) at Reset. As previously mentioned appropriate calibration values are loaded to adjust to center frequency (OSC20M), and temperature drift compensation (TEMPCAL20M), meeting the specifications defined in the Internal Oscillator Characteristics. For applications requiring wider operating range, the relative factory stored frequency error after calibrations can be used. The 4 errors are measured at different setting and are available in Signature Row as signed byte values.

- SIGROW.OSC16ERR3V is the frequency error from 16MHz measured at 3V
- SIGROW.OSC16ERR5V is the frequency error from 16MHz measured at 5V
- SIGROW.OSC20ERR3V is the frequency error from 20MHz measured at 3V
- SIGROW.OSC20ERR5V is the frequency error from 20MHz measured at 5V

The Error is stored as a compressed ***Q1.10*** fixed point 8 bit value, in order not to lose resolution, where the MSB is the sign bit and the 7 LSBs the lower bits of the ***Q.10***.

$$\mathrm{BAUD}_{actual} = \left(\mathrm{BAUD}_{ideal} + \frac{\mathrm{BAUD}_{ideal} * SigRowError}{1024}\right)$$

The minimum legal BAUD register value is 0x40, the target BAUD register value should therefor not be lower than 0x4A to ensure that the compensated BAUD value stays within the legal range, even for parts with negative compensation values. The example code below, demonstrates how to apply this value for more accurate USART baud rate:

```
  #include <assert.h>
  /* Baud rate compensated with factory stored frequency error */
  /* Asynchronous communication without Auto-baud (Sync Field) */
  /* 16MHz Clock, 3V and 600 BAUD              */

  int8_t  sigrow_val    = SIGROW.OSC16ERR3V;   // read signed error
  int32_t baud_reg_val  = 600;                 // ideal BAUD register value

  assert (baud >= 0x4A);                       // Verify legal min BAUD register value with
max neg comp
  baud_reg_val *= (1024 + sigrow_value);       // sum resolution + error
  baud_reg_val /=  1024;                       // divide by resolution
  USART0.BAUD = (int16_t) baud_reg_val;        // set adjusted baud rate
```

**Related Links**

Oscillators and Clocks

#### 32KHz Oscillator (OSCULP32K)

The 32KHz oscillator is optimized for Ultra Low Power (ULP) operation. Power consumption is decreased at the cost of decreased accuracy compared to an external crystal oscillator.

This oscillator also provides the 1KHz signal for the Real Time Counter (RTC), the Watchdog Timer (WDT), and the Brownout Detector (BOD).

The start-up time of this oscillator is the oscillator start-up time plus 4 oscillator cycles. Refer to Electrical Characteristics chapter for the start-up time.

**Related Links**

BOD - Brownout Detector

WDT - Watchdog Timer

RTC - Real Time Counter

### 10.3.4.2 External Clock Sources

These external clock sources are available:

- External clock from pin. (EXTCLK).
- The TOSC1 and TOSC2 pins are dedicated to driving a 32.768kHz crystal oscillator (XOSC32K).
- Instead of a crystal oscillator, TOSC1 can be configured to accept an external clock source.

#### 32.768kHz Crystal Oscillator (XOSC32K)

This oscillator supports two input options: Either a crystal is connected to the pins TOSC1 and TOSC2, or an external clock running at 32KHz is connected to TOSC1. The input option must be configured by writing the Source Select bit (SEL) in the XOSC32K Control A register (CLKCTRL.XOSC32KCTRLA).

The XOSC32K is enabled by writing a '1' to its ENABLE bit in CLKCTRL.XOSC32KCTRLA. When enabled, the configuration of the GPIO pins used by the XOSC32K is overridden as TOSC1, TOSC2 pins. The Enable bit needs to be set for the oscillator to start running when requested.

The start-up time of a given crystal oscillator can be accommodated by writing to the Crystal Start-up Time bits (CSUT) in CLKCTRL.XOSC32KCTRLA.

When XOSC32K is configured to use an external clock on TOSC1, the start-up time is fixed to 2 cycles.

**External Clock (EXTCLK)**
The External Clock is taken directly from the pin. This GPIO pin is automatically configured for EXTCLK if any peripheral is requesting this clock.

This clock source has a start-up time of 2 cycles when first requested.

### 10.3.5 Configuration Change Protection

This peripheral has registers that are under Configuration Change Protection (CCP). In order to write to these, a certain key must be written to the CPU.CCP register first, followed by a write access to the protected bits within four CPU instructions.

It is possible to try writing to these registers any time, but the values are not altered.

The following registers are under CCP:

**Table 10-2. CLKCTRL - Registers under Configuration Change Protection**

| Register | Key |
|---|---|
| CLKCTRL.MCLKCTRLB | IOREG |
| CLKCTRL.MCLKLOCK | IOREG |
| CLKCTRL.XOSC32KCTRLA | IOREG |
| CLKCTRL.MCLKCTRLA | IOREG |
| CLKCTRL.OSC20MCTRLA | IOREG |
| CLKCTRL.OSC20MCALIBA | IOREG |
| CLKCTRL.OSC20MCALIBB | IOREG |
| CLKCTRL.OSC32KCTRLA | IOREG |

**Related Links**

Sequence for Write Operation to Configuration Change Protected I/O Registers

## 10.4 Register Summary - CLKCTRL

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | MCLKCTRLA | 7:0 | CLKOUT | | | | | | CLKSEL[1:0] | |
| 0x01 | MCLKCTRLB | 7:0 | | | | PDIV[3:0] | | | | PEN |
| 0x02 | MCLKLOCK | 7:0 | | | | | | | | LOCKEN |
| 0x03 | MCLKSTATUS | 7:0 | EXTS | XOSC32KS | OSC32KS | OSC20MS | | | | SOSC |
| 0x04 ... 0x0F | Reserved | | | | | | | | | |
| 0x10 | OSC20MCTRLA | 7:0 | | | | | | | RUNSTDBY | |
| 0x11 | OSC20MCALIBA | 7:0 | | | CAL20M[5:0] | | | | | |
| 0x12 | OSC20MCALIBB | 7:0 | LOCK | | | | TEMPCAL20M[3:0] | | | |
| 0x13 ... 0x17 | Reserved | | | | | | | | | |
| 0x18 | OSC32KCTRLA | 7:0 | | | | | | | RUNSTDBY | |
| 0x19 ... 0x1B | Reserved | | | | | | | | | |
| 0x1C | XOSC32KCTRLA | 7:0 | | | CSUT[1:0] | | | SEL | RUNSTDBY | ENABLE |

## 10.5 Register Description

### 10.5.1 Main Clock Control A

**Name:** MCLKCTRLA
**Offset:** 0x00
**Reset:** 0x00
**Property:** Configuration Change Protection

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

**Bit 7 – CLKOUT: System Clock Out**
When this bit is written to '1', the system clock is output to CLKOUT pin.

When the device is in a sleep mode, there is no clock output unless a peripheral is using the system clock.

**Bits 1:0 – CLKSEL[1:0]: Clock Select**
This bit field selects the source for the Main Clock (CLK_MAIN).

| Value | Name | Description |
|---|---|---|
| 0x0 | OSC20M | 16/20MHz internal oscillator |
| 0x1 | OSCULP32K | 32KHz internal Ultra Low Power oscillator |
| 0x2 | XOSC32K | 32.768kHz external crystal oscillator |
| 0x3 | EXTCLK | External clock |

### 10.5.2 Main Clock Control B

**Name:** MCLKCTRLB
**Offset:** 0x01
**Reset:** 0x11
**Property:** Configuration Change Protection

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

**Bits 4:1 – PDIV[3:0]: Prescaler Division**
If the Prescaler Enable (PEN) bit is written to '1', these bits define the division ratio of the Main Clock prescaler.

These bits can be written during run-time to vary the clock frequency of the system to suit the application requirements.

User software must ensure a correct configuration of input frequency (CLK_MAIN) and Prescaler settings, such that the resulting frequency of CLK_PER never exceeds the allowed maximum (see Electrical Characteristics).

| Value | Description |
|---|---|
| Value | Division |
| 0x0 | 2 |
| 0x1 | 4 |
| 0x2 | 8 |
| 0x3 | 16 |
| 0x4 | 32 |
| 0x5 | 64 |
| 0x8 | 6 |
| 0x9 | 10 |
| 0xA | 12 |
| 0xB | 24 |
| 0xC | 48 |
| other | Reserved |

**Bit 0 – PEN: Prescaler Enable**
This bit must be written '1' to enable the prescaler. When enabled, the division ratio is selected by the PDIV bit field.

When this bit is written to '0', the Main Clock will pass through undivided (CLK_PER=CLK_MAIN), regardless of the value of PDIV.

### 10.5.3 Main Clock Lock

**Name:** MCLKLOCK
**Offset:** 0x02
**Reset:** 0x00
**Property:** Configuration Change Protection

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

**Bit 0 – LOCKEN: Lock Enable**
Writing this bit to '1' will lock the CLKCTRL.MCLKCTRLA and CLKCTRL.MCLKCTRLB registers, and, if applicable, the calibration settings for the current Main Clock source from further software updates. Once locked, the CLKCTRL.MCLKLOCK registers cannot be accessed until the next hardware Reset.

This provides protection for the CLKCTRL.MCLKCTRLA and CLKCTRL.MCLKCTRLB registers and calibration settings for the Main Clock source from unintentional modification by software.

**Related Links**
FUSES - Configuration and User Fuses

### 10.5.4 Main Clock Status

**Name:** MCLKSTATUS
**Offset:** 0x03
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | EXTS | XOSC32KS | OSC32KS | OSC20MS | | | | SOSC |
| Access | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bit 7 – EXTS: External Clock Status**

| Value | Description |
|---|---|
| 0 | EXTCLK has not started |
| 1 | EXTCLK has started |

**Bit 6 – XOSC32KS: XOSC32K Status**
The status bit will only be available if the source is requested as the main clock or by another module. If the oscillators RUNSTDBY bit is set but the oscillator is unused/not requested this bit will be 0.

| Value | Description |
|---|---|
| 0 | XOSC32K is not stable |
| 1 | XOSC32K is stable |

**Bit 5 – OSC32KS: OSCULP32K Status**
The status bit will only be available if the source is requested as the main clock or by another module. If the oscillators RUNSTDBY bit is set but the oscillator is unused/not requested this bit will be 0.

| Value | Description |
|---|---|
| 0 | OSCULP32K is not stable |
| 1 | OSCULP32K is stable |

**Bit 4 – OSC20MS: OSC20M Status**
The status bit will only be available if the source is requested as the main clock or by another module. If the oscillators RUNSTDBY bit is set but the oscillator is unused/not requested this bit will be 0.

| Value | Description |
|---|---|
| 0 | OSC20M is not stable |
| 1 | OSC20M is stable |

**Bit 0 – SOSC: Main Clock Oscillator Changing**

| Value | Description |
|---|---|
| 0 | The clock source for CLK_MAIN is not undergoing a switch. |
| 1 | The clock source for CLK_MAIN is undergoing a switch, and will change as soon as the new source is stable. |

### 10.5.5 16/20MHz Oscillator Control A

**Name:** OSC20MCTRLA
**Offset:** 0x10
**Reset:** 0x00
**Property:** Configuration Change Protection

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

**Bit 1 – RUNSTDBY: Run Standby**
This bit force the oscillator on in all modes, even when unused by the system. In standby sleep mode this can be used to ensure immediate wake-up and not waiting for oscillator start-up time.

When not requested by peripherals, no oscillator output is provided.

It takes 4 oscillator cycles to open the clock gate after a request but the oscillator analog start-up time will be removed when this bit is set.

### 10.5.6 16/20MHz Oscillator Calibration A

**Name:** OSC20MCALIBA
**Offset:** 0x11
**Reset:** Based on FREQSEL in FUSE.OSCCFG
**Property:** Configuration Change Protection

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | CAL20M[5:0] | | | | | |
| Access | | | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | | | x | x | x | x | x | x |

**Bits 5:0 – CAL20M[5:0]: Calibration**
These bits change the frequency around the current center frequency of the OSC20M for fine tuning.

At Reset factory calibrated values are loaded based on FREQSEL bits in FUSE.OSCCFG.

### 10.5.7 16/20MHz Oscillator Calibration B

**Name:** OSC20MCALIBB
**Offset:** 0x12
**Reset:** Based on FUSE.OSCCFG
**Property:** Configuration Change Protection

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | LOCK | | | | TEMPCAL20M[3:0] | | | |
| Access | R | | | | R/W | R/W | R/W | R/W |
| Reset | x | | | | x | x | x | x |

**Bit 7 – LOCK: Oscillator Calibration Locked by Fuse**
When this bit is set, the calibration settings in CLKCTRL.OSC20MCALIBA and CLKCTRL.OSC20MCALIBB cannot be changed.
The Reset value is loaded from the OSCLOCK bit in the Oscillator Configuration fuse (FUSE.OSCCFG).

**Bits 3:0 – TEMPCAL20M[3:0]: Oscillator Temperature Coefficient Calibration**
These bits tune the slope of the temperature compensation.

At Reset factory calibrated values are loaded based on FREQSEL bits in FUSE.OSCCFG.

### 10.5.8 32KHz Oscillator Control A

**Name:** OSC32KCTRLA
**Offset:** 0x18
**Reset:** 0x00
**Property:** Configuration Change Protection

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

**Bit 1 – RUNSTDBY: Run Standby**
This bit force the oscillator on in all modes, even when unused by the system. In standby sleep mode this can be used to ensure immediate wake-up and not waiting for oscillator start-up time.

When not requested by peripherals, no oscillator output is provided.

It takes 4 oscillator cycles to open the clock gate after a request but the oscillator analog start-up time will be removed when this bit is set.

### 10.5.9 32.768kHz Crystal Oscillator Control A

The SEL and CSUT bits cannot be changed as long as the ENABLE bit is set or the XOSC32K Stable bit (XOSC32KS) in CLKCTRL.MCLKSTATUS is high.
To change settings in a safe way: write a '0' to the ENABLE bit and wait until XOSC32KS is '0' before re-enabling the XOSC32K with new settings.

**Name:** XOSC32KCTRLA
**Offset:** 0x1C
**Reset:** 0x00
**Property:** Configuration Change Protection

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | CSUT[1:0] | | | SEL | RUNSTDBY | ENABLE |
| Access | R | R | R/W | R/W | R | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 5:4 – CSUT[1:0]: Crystal Start-Up Time**
These bits select the start-up time for the XOSC32K. It is write protected when the oscillator is enabled (ENABLE=1).

If SEL=1, the start-up time will not be applied.

| Value | Name | Description |
|---|---|---|
| 0x0 | 1K | 1k cycles |
| 0x1 | 16K | 16k cycles |
| 0x2 | 32K | 32k cycles |
| 0x3 | 64K | 64k cycles |

**Bit 2 – SEL: Source Select**
This bit select the external source type. It is write protected when the oscillator is enabled (ENABLE=1).

| Value | Description |
|---|---|
| 0 | External crystal |
| 1 | External clock on TOSC1 pin |

**Bit 1 – RUNSTDBY: Run Standby**
Writing this bit to '1' starts the crystal oscillator and forces the oscillator on in all modes, even when unused by the system if the ENABLE bit is set. In standby sleep mode this can be used to ensure immediate wake-up and not waiting for oscillator start-up time. When this bit is '0', the crystal oscillator is only running when requested and the ENABLE bit is set.

The output of XOSC32K is not sent to other peripherals unless it is requested by one or more peripherals.

When the RUNSTDBY bit is set there will only be a delay of two to three crystal oscillator cycles after a request until the oscillator output is received, if the initial crystal start-up time already has completed.

According to RUNSTBY bit, the oscillator will be turned on all the time if the device is in Active, Idle or Standby sleep mode, or only be enabled when requested.

This bit is I/O protected to prevent unintentional enabling of the oscillator.

© 2017 Microchip Technology Inc.

**Bit 0 – ENABLE: Enable**
When this bit is written to '1', the configuration of the respective input pins is overridden to TOSC1 and TOSC2. Also, the Source Select bit (SEL) and Crystal Start-Up Time (CSUT) become read-only.

This bit is I/O protected to prevent unintentional enabling of the oscillator.

# 11. SLPCTRL - Sleep Controller

## 11.1 Features

- Three sleep modes:
  - Idle
  - Standby
  - Power Down
- Configurable Standby sleep mode where peripherals can be configured as on or off.

## 11.2 Overview

Sleep modes are used to shut down peripherals and clock domains in the device in order to save power. The Sleep Controller (SLPCTRL) controls and handles the transitions between active mode and sleep mode.

There are in total four modes available, one active mode in which software is executed, and three sleep modes. The available sleep modes are: Idle, Standby and Power Down.

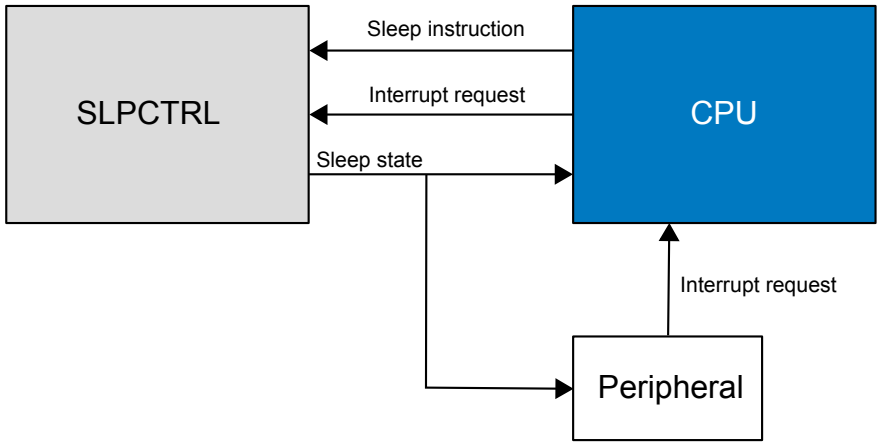
When the device enters a sleep mode, program execution is stopped, and depending on the entered sleep mode, different peripherals and clock domains are turned off.

To enter a sleep mode, the SLPCTRL must be enabled and the desired sleep mode must be stated. The software decides when to enter that sleep mode by using a dedicated sleep instruction (`SLEEP`).

Interrupts are used to wake up the device from sleep. The available Interrupt wake-up sources depend on the configured sleep mode. When an interrupt occurs, the device will wake up and execute the interrupt service routine before continuing normal program execution from the first instruction after the sleep instruction. Any Reset will also take the device out of a sleep mode.

### 11.2.1 Block Diagram

**Figure 11-1. Sleep Controller in System**




### 11.2.2 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 11-1. SLPCTRL System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | No | - |
| Interrupts | no | - |
| Events | No | - |
| Debug | Yes | UPDI |

#### 11.2.2.1 Clocks

This peripheral depends on the peripheral clock.

**Related Links**

CLKCTRL - Clock Controller

#### 11.2.2.2 I/O Lines and Connections

Not applicable.

#### 11.2.2.3 Interrupts

Not applicable.

#### 11.2.2.4 Events

Not applicable.

#### 11.2.2.5 Debug Operation

When run-time debugging, this peripheral will continue normal operation. The SLPCTRL is only affected by a break in debug operation: If the SLPCTRL is in a sleep mode when a break occurs, the device will wake up and the SLPCTRL will go to active mode, even if there are no pending interrupt requests.

If the peripheral is configured to require periodical service by the CPU through interrupts or similar, improper operation or data loss may result during halted debugging.

## 11.3 Functional Description

### 11.3.1 Initialization

To put the device into a sleep mode, follow these steps:

- Configure and enable the interrupts that should wake up the device from sleep. Also enable global interrupts.

  **Warning:** If there are no interrupts enabled when going to sleep, the device cannot wake up again. Only a reset will allow the device to continue operation.

- Select the sleep mode to be entered and enable the Sleep Controller by writing to the Sleep Mode bits (SMODE) and the Enable bit (SEN) in the Control A register (SLPCTRL.CTRLA). A SLEEP instruction must be run to make the device actually go to sleep.

### 11.3.2 Operation

#### 11.3.2.1 Sleep Modes

In addition to Active mode, there are three different sleep modes, with decreasing power consumption and functionality.

**Idle** The CPU stops executing code, no peripherals are disabled.
All interrupt sources can wake up the device.

**Standby** The user can configure peripherals to be enabled or not, using the respective RUNSTBY bit. This means that the power consumption is highly dependent on what functionality is enabled, and thus may vary between the Idle and Power Down levels.
SleepWalking is available for the ADC module.
The wake-up sources are Pin interrupts, TWI address match, UART Start-of-Frame interrupt (if USART is enabled to run in Standby), RTC interrupt (if RTC enabled to run in Standby), and TCB interrupt.

**Power Down** Only the WDT and the PIT (component of the RTC) are active.
The only wake-up sources are the pin change interrupt and TWI address match.

**Table 11-2. Sleep Mode Activity Overview**

| Group | Peripheral | | Active in Sleep Mode | | |
|---|---|---|---|---|---|
| | | Clock | Idle | Standby | Power Down |
| Active Clock Domain | CPU | CLK_CPU | | | |
| | Peripherals | CLK_PER | X | | |
| | RTC | CLK_RTC | X | X* | |
| | ADC | CLK_PER | X | X* | |
| | PIT (RTC) | CLK_RTC | X | X | X |
| | WDT | CLK_WDT | X | X | X |
| | Oscillators | Main Clock Source | X | X* | |
| | | RTC Clock Source | X | X* | |
| WDT Oscillator | | X | X | X | |
| Wake-Up Sources | INTn and pin change | | X | X | X |
| | TWI address match | | X | X | X |
| | Periodic Interrupt Timer | | X | X | X |
| | UART start of frame | | X | X* | |
| | ADC window | | X | X* | |
| | RTC interrupt | | X | X* | |
| | All other interrupts | | X | | |

**Note:**

- X means active. X* indicates that the RUNSTBY bit of the corresponding peripheral must be set to enter active state.

#### 11.3.2.2 Wake-Up Time

The normal wake-up time for the device is 6 Main Clock cycles (CLK_PER), plus the time it takes to start up the Main Clock source:

- In Idle sleep mode the Main Clock source is kept running so it will not be any extra wake-up time.
- In Standby sleep mode the Main Clock might be running, so it depends on the peripheral configuration.
- In Power Down sleep mode only the ULP 32KHz oscillator and RTC clock may be running, if it is used by the BOD or WDT. All other clock sources will be off.

**Table 11-3. Sleep modes and Start-up time**

| Sleep mode | Start-up time |
|---|---|
| IDLE | 6 CLK |
| Standby | 6 CLK + OSC start-up |
| Power Down | 6 CLK + OSC start-up |

The start-up time for the different clock sources is described in the Clock Controller (CLKCTRL) section.

In addition to the normal wake-up time it is possible to make the device wait until the BOD is ready before executing code. This is done by writing 0x3 to the BOD Operation Mode in Active and Idle bits (ACTIVE) in the BOD Configuration fuse (FUSE.BODCFG). If the BOD is ready before the normal wake-up time, the net wake-up time will be the same. If the BOD takes longer than the normal wake-up time, the wake-up time will be extended until the BOD is ready. This is ensures correct supply voltage whenever code is executed.

### 11.3.3 Configuration Change Protection

Not applicable.

## 11.4 Register Summary - SLPCTRL

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | | | | | | SMODE[1:0] | | SEN |

## 11.5 Register Description

### 11.5.1 Control A

**Name:** CTRLA
**Offset:** 0x00
**Reset:** 0x00
**Property:** -

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

**Bits 2:1 – SMODE[1:0]: Sleep Mode**
Writing these bits selects the sleep mode entered when the Sleep Enable bit (SEN) is written to '1' and the SLEEP instruction is executed.

| Value | Name | Description |
|---|---|---|
| 0x0 | IDLE | Idle sleep mode enabled |
| 0x1 | STANDBY | Standby sleep mode enabled |
| 0x2 | PDOWN | Power Down sleep mode enabled |
| other | - | Reserved |

**Bit 0 – SEN: Sleep Enable**
This bit must be written to '1' before the SLEEP instruction is executed to make the MCU enter the selected sleep mode.

# 12. RSTCTRL - Reset Controller

## 12.1 Features

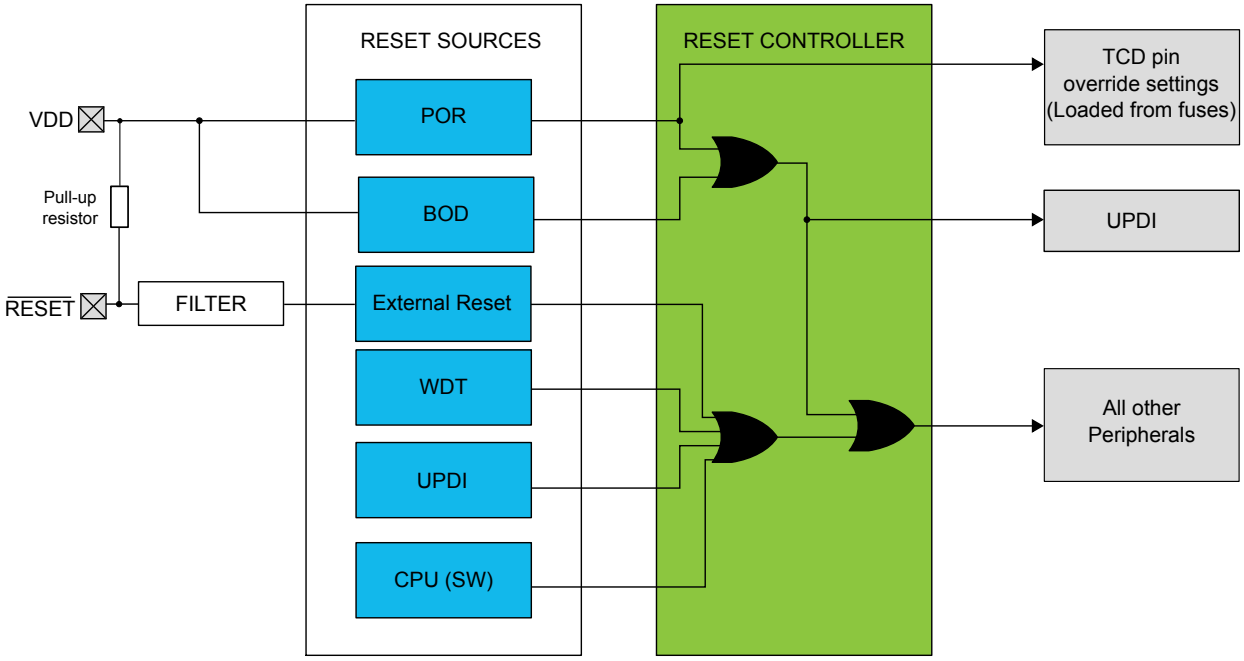
- Reset the device and set it to an initial state.
- Reset Flag register for identifying the Reset source in software.
- Multiple Reset sources:
  - Power supply Reset sources: Brownout Detect (BOD), Power On Reset (POR)
  - User Reset sources: External Reset pin ($\overline{\text{RESET}}$), Watchdog Reset (WDT), Software Reset (SW) and UPDI Reset.

## 12.2 Overview

The Reset Controller (RSTCTRL) manages the Reset of the device. It issues a device Reset, sets the device to its initial state, and allows the Reset source to be identified by software.

### 12.2.1 Block Diagram

**Figure 12-1. Reset System Overview**




### 12.2.2 Signal Description

| Signal | Description | Type |
|---|---|---|
| $\overline{\text{RESET}}$ | External Reset (active low) | Digital input |

## 12.3 Functional Description

### 12.3.1 Initialization

The Reset Controller (RSTCTRL) is always enabled, but some of the Reset sources must be enabled (either by fuses or by software) before they can request a Reset.

After any Reset, the Reset source that caused the Reset is found in the Reset Flag register (RSTCTRL.RSTFR).

After a Power-On Reset, only the POR flag will be set.

The flags are kept until they are cleared by writing a '1' to them.

After reset from any source, all registers that are loaded from fuses are reloaded.

### 12.3.2 Operation

#### 12.3.2.1 Reset Sources

There are two kind of sources for Resets:

- Power supply Resets, which are caused by changes in the power supply voltage: Power-on Reset (POR) and Brownout Detector (BOD).
- User Resets, which are issued by the application, by debug operation or by pin changes (Software Reset, Watchdog Reset, UPDI Reset and external Reset pin $\overline{\text{RESET}}$).

**Power-On Reset (POR)**

A power-on-reset (POR) is generated by an on-chip detection circuit. The POR is activated when the $V_{DD}$ rises until it reach the POR treshold voltage. The POR is always enabled and will also detect when the $V_{DD}$ falls below the treshold voltage.

All logic is reset on POR. All fuses are reloaded after the reset is released. While POR is active, TCD pin override functionality is not available.

**Brownout Detector (BOD) Reset Source**

The on-chip Brown-out Detection circuit will monitor the $V_{DD}$ level during operation by comparing it to a fixed trigger level. The trigger level for the BOD can be selected by fuses. If BOD is unused in application it is forced on to a minimum level in order to guarantee safe operation during internal reset and chip erase.

All logic is reset on BOD reset, except TCD pin override settings and BOD configuration. All fuses are reloaded after the reset is released.

**Related Links**

BOD - Brownout Detector

**Software Reset**

The software reset makes it possible to issue a system reset from software. The reset is generated by writing a '1' to the Software Reset Enable bit (SWRE) in the Software Reset register (RSTCTRL.SWRR).

The reset will take place immediately after the bit is written and the device will be kept in reset until the reset sequence is completed. All logic is reset on software reset, except UPDI, TCD pin override settings and BOD configuration. All fuses are reloaded after the reset is released.

**External Reset**

The External Reset is enabled by fuse (see fuse map).

When enabled, the External Reset requests a Reset as long as the $\overline{\text{RESET}}$ pin is low. The device will stay in Reset until $\overline{\text{RESET}}$ is high again. All logic is reset on external reset, except UPDI, TCD pin override settings and BOD configuration. All fuses are reloaded after the reset is released.

**Related Links**

FUSES - Configuration and User Fuses

**Watchdog Reset**

The watchdog timer (WDT) is a system function for monitoring correct program operation. If the WDT is not reset from software according to the programmed timeout period, a watchdog reset will be issued. See the WDT documentation for further details.

All logic is reset on WDT reset, except UPDI, TCD pin override settings and BOD configuration. All fuses are reloaded after the reset is released.

**Related Links**

WDT - Watchdog Timer

**Universal Program Debug Interface (UPDI) Reset**

The UPDI contains a separate reset source that is used to reset the device during external programming and debugging. The reset source is accessible only from external debuggers and programmers. All logic is reset on UPDI reset, except the UPDI itself, TCD pin override settings and BOD configuration. All fuses are reloaded after the reset is released. See UPDI chapter on how to generate a UPDI reset request.

**Related Links**

UPDI - Unified Program and Debug Interface

#### 12.3.2.2 Reset time

The reset time can be split in two.

The first part is when any of the Reset sources are active. This part depends on the input to the Reset sources. The external reset is active as long as the $\overline{\text{RESET}}$ pin is low, the power-on-reset (POR) and Brownout detector (BOD) is active as long as the supply voltage is below reset source threshold.

When all the Reset sources are released an Internal reset initialization of the device is done. The time it take to do the Internal reset initialization is typical 250 µs. This time will be increased with the Start Up time given by the Start Up time fuse setting (SYSCFG1.SUT). The Internal reset initialization time will also increase if the CRC source is setup to run (SYSCFG0.CRCSRC). The additional Internal reset time will increase by 500 µs for each KB of memory that is checked.

### 12.3.3 Sleep Mode Operation

The Reset Controller continues to operate in all active and sleep modes.

### 12.3.4 Configuration Change Protection

This peripheral has registers that are under Configuration Change Protection (CCP). In order to write to these, a certain key must be written to the CPU.CCP register first, followed by a write access to the protected bits within four CPU instructions.

It is possible to try writing to these registers any time, but the values are not altered.

The following registers are under CCP:

**Table 12-1. RSTCTRL - Registers under Configuration Change Protection**

| Register | Key |
|---|---|
| RSTCTRL.SWRR | IOREG |

**Related Links**

Sequence for Write Operation to Configuration Change Protected I/O Registers

## 12.4 Register Summary - RSTCTRL

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | RSTFR | 7:0 | | | UPDIRF | SWRF | WDRF | EXTRF | BORF | PORF |
| 0x01 | SWRR | 7:0 | | | | | | | | SWRE |

## 12.5 Register Description

### 12.5.1 Reset Flag Register

All flags are cleared by writing a '1' to them. They are also cleared by a Power-on Reset, with the exception of the Power-On Reset Flag (PORF).

**Name:** RSTFR
**Offset:** 0x00
**Reset:** 0xXX
**Property:** -

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

**Bit 5 – UPDIRF: UPDI Reset Flag**
This bit is set if a UPDI Reset occurs.

**Bit 4 – SWRF: Software Reset Flag**
This bit is set if a Software Reset occurs.

**Bit 3 – WDRF: Watchdog Reset Flag**
This bit is set if a Watchdog Reset occurs.

**Bit 2 – EXTRF: External Reset Flag**
This bit is set if an External reset occurs.

**Bit 1 – BORF: Brownout Reset Flag**
This bit is set if a Brownout Reset occurs.

**Bit 0 – PORF: Power-On Reset Flag**
This bit is set if a Power-on Reset occurs.

This flag is only cleared by writing a '1' it.

After a POR, only the POR flag is set and all other flags are cleared. No other flag can be set before a full system boot is run after the POR.

### 12.5.2 Software Reset Register

**Name:** SWRR
**Offset:** 0x01
**Reset:** 0x00
**Property:** Configuration Change Protection

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

**Bit 0 – SWRE: Software Reset Enable**
When this bit is written to '1', a software reset will occur.

This bit will always read as '0'.

# 13. CPUINT - CPU Interrupt Controller

## 13.1 Features

- Short and predictable interrupt response time
- Separate interrupt configuration and vector address for each interrupt
- Interrupt prioritizing by level and vector address
- Two interrupt priority levels: 0 (normal) and 1 (high)
- Higher priority for one interrupt
- Optional round-robin priority scheme for priority level 0 interrupts
- Non-maskable interrupts (NMI) for critical functions
- Interrupt vectors optionally placed in the application section or the boot loader section
- Selectable compact vector table

## 13.2 Overview

An interrupt request signals a change of state inside a peripheral, and can be used to alter program execution. Peripherals can have one or more interrupts, and all are individually enabled and configured.

When an interrupt is enabled and configured, it will generate an interrupt request when the interrupt condition occurs.

The CPU Interrupt Controller (CPUINT) handles and prioritizes interrupt requests. When an interrupt is enabled and the interrupt condition occurs, the CPUINT will receive the interrupt request. Based on the interrupt's priority level and the priority level of any ongoing interrupts, the interrupt request is either acknowledged or kept pending until it has priority. When an interrupt request is acknowledged by the CPUINT, the program counter is set to point to the interrupt vector. The interrupt vector is normally a jump to the interrupt handler (i.e., the software routine that handles the interrupt). After returning from the interrupt handler, program execution continues from where it was before the interrupt occurred. One instruction is always executed before any pending interrupt is served.

The CPUINT Status register (CPUINT.STATUS) contains state information that ensures that the CPUINT returns to the correct interrupt level when the `RETI` (interrupt return) instruction is executed at the end of an interrupt handler. Returning from an interrupt will return the CPUINT to the state it had before entering the interrupt. CPUINT.STATUS is not saved automatically upon an interrupt request.

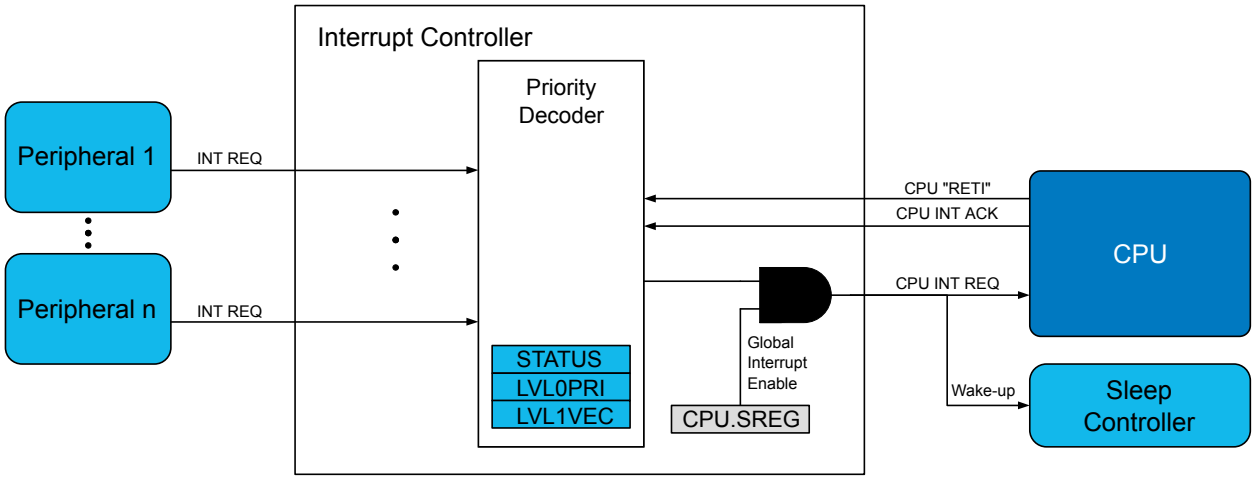
By default, all peripherals are priority level 0. It is possible to set one single interrupt vector to the higher priority level 1. Interrupts are prioritized according to their priority level and their interrupt vector address. Priority level 1 interrupts will interrupt level 0 interrupt handlers. Among priority level 0 interrupts, the priority is determined from the interrupt vector address, where the lowest interrupt vector address has the highest interrupt priority.

Optionally, a round-robin scheduling scheme can be enabled for priority level 0 interrupts. This ensures that all interrupts are serviced within a certain amount of time.

Interrupt generation must be globally enabled by writing a '1' to the Global Interrupt Enable bit (I) in the CPU Status register (CPU.SREG). This bit is not cleared when an interrupt is acknowledged.

### 13.2.1 Block Diagram

**Figure 13-1. CPUINT Block Diagram**




### 13.2.2 Signal Description

Not applicable.

### 13.2.3 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 13-1. CPUINT System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | No | - |
| Interrupts | No | - |
| Events | No | - |
| Debug | Yes | UPDI |

**Related Links**

Debug Operation

Clocks

#### 13.2.3.1 Clocks

This peripheral depends on the peripheral clock.

**Related Links**

CLKCTRL - Clock Controller

#### 13.2.3.2 I/O Lines and Connections

Not applicable.

#### 13.2.3.3 Interrupts

Not applicable.

#### 13.2.3.4 Events

Not applicable.

#### 13.2.3.5 Debug Operation

When run-time debugging, this peripheral will continue normal operation. Halting the CPU in debugging mode will halt normal operation of the peripheral.

If the peripheral is configured to require periodical service by the CPU through interrupts or similar, improper operation or data loss may result during halted debugging.

**Related Links**

UPDI - Unified Program and Debug Interface

## 13.3 Functional Description

### 13.3.1 Initialization

An interrupt must be initialized in the following order:

1. Configure the CPUINT if the default configuration is not adequate (optional):
   - Vector handling is configured by writing to the respective bits (IVSEL and CVT) in the Control A register (CPUINT.CTRLA).
   - Vector prioritizing by round-robin is enabled by writing a '1' to the Round-Robin Priority Enable bit (LVL0RR) in CPUINT.CTRLA.
   - Select the priority level 1 vector by writing its address to the Interrupt Vector (LVL1VEC) in the Level 1 Priority register (CPUINT.LVL1VEC).
2. Configure the interrupt conditions within the peripheral, and enable the peripheral's interrupt.
3. Enable interrupts globally by writing a '1' to the Global Interrupt Enable bit (I) in the CPU Status register (CPU.SREG).

### 13.3.2 Operation

#### 13.3.2.1 Enabling, Disabling, and Resetting

Global enabling of interrupts is done by writing a '1' to the Global Interrupt Enable bit (I) in the CPU Status register (CPU.SREG). To disable interrupts globally, write a '0' to the I bit in CPU.SREG.

The desired interrupt lines must also be enabled in the respective peripheral, by writing to the peripheral's Interrupt Control register ([peripheral].INTCTRL).

Interrupt flags are not automatically cleared after the interrupt is executed. The respective INTFLAGS register descriptions provide information on how to clear specific flags.

#### 13.3.2.2 Interrupt Vector Locations

The table below shows Reset addresses and Interrupt vector placement, dependent on the value of Interrupt Vector Select bit (IVSEL) in the Control A register (CPUINT.CTRLA).

If the program never enables an interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations.

**Table 13-2. Reset and Interrupt Vector Placement**

| IVSEL | Reset address | Interrupt vectors start address |
|---|---|---|
| 0 | 0x0000 | Application start address + Interrupt vector offset address |
| 1 | 0x0000 | Interrupt vector offset address |

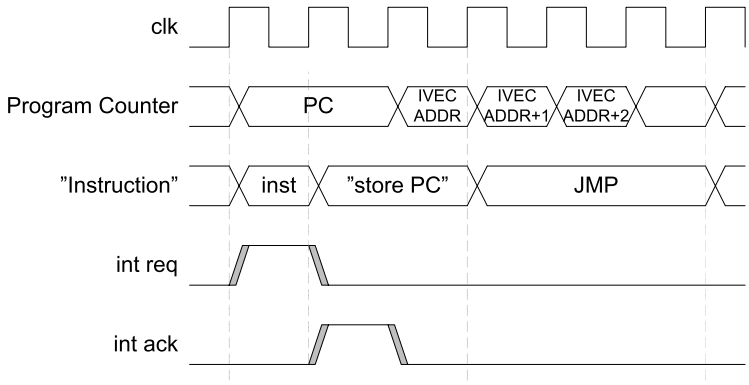
#### 13.3.2.3 Interrupt Response Time

The minimum interrupt response time for all enabled interrupts is three CPU clock cycles: one cycle to finish the ongoing instruction, and two cycles to store the program counter to the stack. After the program counter is pushed on the stack, the program vector for the interrupt is executed. See also figure below, first diagram.

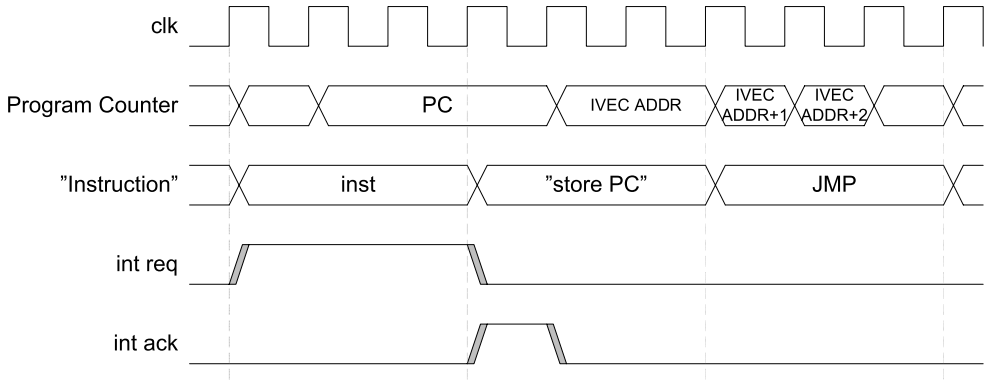
The jump to the interrupt handler takes three clock cycles. If an interrupt occurs during execution of a multicycle instruction, this instruction is completed before the interrupt is served. See also figure below, second diagram.

**Figure 13-2. Interrupt Execution of a Single Cycle Instruction, Multicycle Instruction and from Sleep**

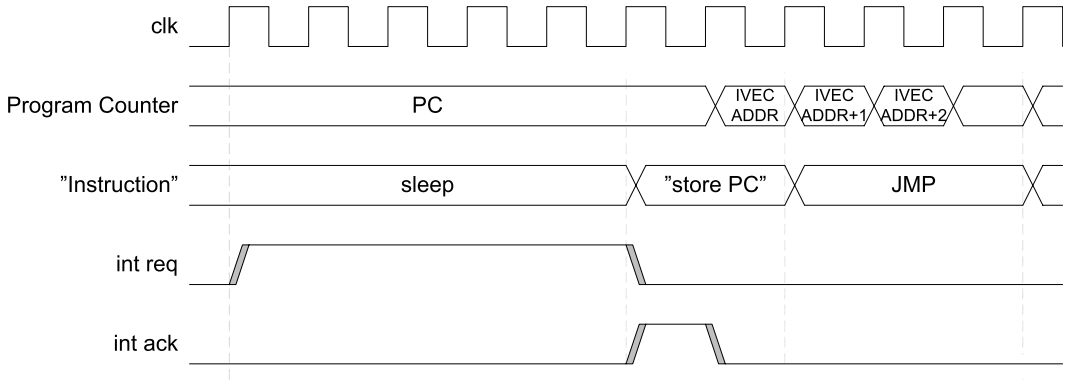
Single Cycle Instruction




Multicycle Instruction




Sleep




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

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

#### 13.3.2.4 Interrupt Level

The interrupt level is default on level 0 (normal) for all interrupt sources. It is possible to select one interrupt source to level 1 (high) by writing interrupt address to CPUINT.LVL1VEC register. This source will have higher priority than normal level interrupts.

An interrupt request from a level 1 source will interrupt any ongoing interrupt handler from a level 0 interrupt. When returning from the level 1 interrupt handler, the execution of the level 0 interrupt handler will continue.

#### 13.3.2.5 Interrupt Priority

**NMI - Non-Maskable Interrupts**

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

Which interrupts are non-maskable is device-dependent and not subject to configuration. Non-maskable interrupts must be enabled before they can be used. Refer to the Interrupt Vector Mapping of the device for available NMI lines.

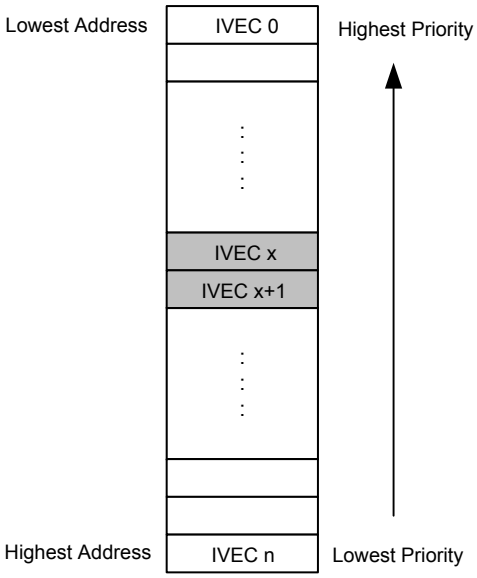
**Related Links**

Interrupt Vector Mapping

**Static Priority**

Interrupt vectors (IVEC) are located at fixed addresses. For static priority, the interrupt vector address decides the priority within normal interrupt level, where the lowest interrupt vector address has the highest priority. Refer to the Interrupt Vector Mapping of the device for available interrupt lines and their base address offset.

**Figure 13-3. Static Priority**




**Related Links**

Interrupt Vector Mapping

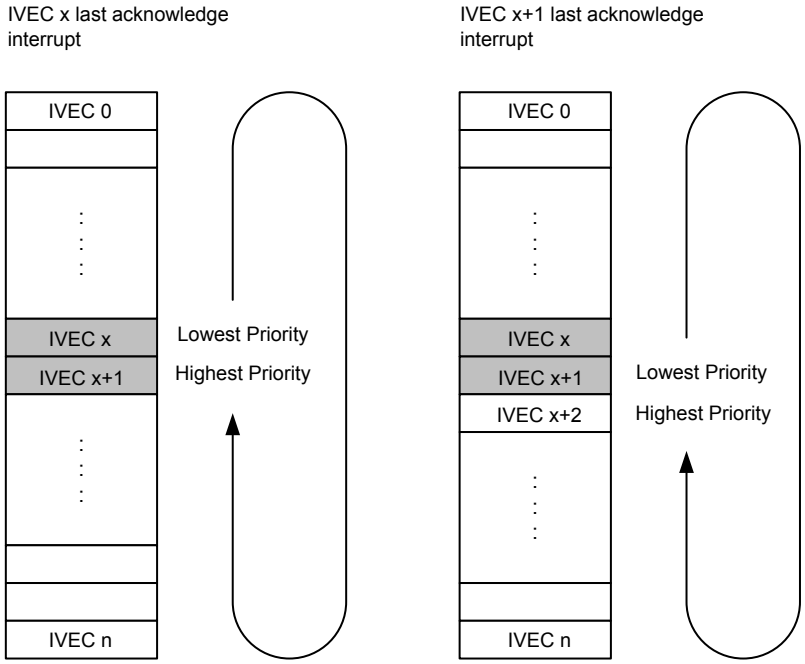
**Round-Robin Scheduling**

To avoid "starvation" for priority level 0 (LVL0) interrupt requests with static priority, i.e. some interrupts might never be served, the CPUINT offers round-robin scheduling for LVL0 interrupts.

Round-robin scheduling for LVL0 interrupt requests is enabled by writing a '1' to the Round-Robin Priority Enable bit (LVL0RR) in the Control A register (CPUINT.CTRLA).

When round-robin scheduling is enabled, the interrupt vector address for the last acknowledged LVL0 interrupt will have the lowest priority the next time one or more LVL0 interrupts are requested, as illustrated in the figure below.

**Figure 13-4. Round-Robin Scheduling**




**Compact Vector Table**

The Compact Vector Table (CVT) is a feature to allow for writing of compact code.

When CVT is enabled by writing a '1' to the CVT bit in the Control A register (CPUINT.CTRLA), the vector table contains these three interrupt vectors:

1. The non-maskable interrupts (NMI) at vector address 1
2. The priority level 1 (LVL1) interrupt at vector address 2
3. All priority level 0 (LVL0) interrupts share vector address 3.

This feature is most suitable for applications using a small number of interrupt generators.

### 13.3.3 Events

Not applicable.

### 13.3.4 Sleep Mode Operation

Not applicable.

### 13.3.5 Configuration Change Protection

This peripheral has registers that are under Configuration Change Protection (CCP). In order to write to these, a certain key must be written to the CPU.CCP register first, followed by a write access to the protected bits within four CPU instructions.

It is possible to try writing to these registers any time, but the values are not altered.

The following registers are under CCP:

**Table 13-3. INTCTRL - Registers under Configuration Change Protection**

| Register | Key |
|---|---|
| IVSEL in CPUINT.CTRLA | IOREG |
| CVT in CPUINT.CTRLA | IOREG |

**Related Links**

Sequence for Write Operation to Configuration Change Protected I/O Registers

## 13.4 Register Summary - CPUINT

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | | IVSEL | CVT | | | | | LVL0RR |
| 0x01 | STATUS | 7:0 | NMIEX | | | | | | LVL1EX | LVL0EX |
| 0x02 | LVL0PRI | 7:0 | LVL0PRI[7:0] | | | | | | | |
| 0x03 | LVL1VEC | 7:0 | LVL1VEC[7:0] | | | | | | | |

## 13.5 Register Description

### 13.5.1 Control A

**Name:** CTRLA
**Offset:** 0x00
**Reset:** 0x00
**Property:** Configuration Change Protection

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | IVSEL | CVT | | | | | LVL0RR |
| Access | | R/W | R/W | | | | | R/W |
| Reset | | 0 | 0 | | | | | 0 |

**Bit 6 – IVSEL: Interrupt Vector Select**
If boot section is defined, it will be placed before application section. The actual start address of the application section is determined by the BOOTEND Fuse.

This bit is protected by the Configuration Change Protection mechanism.

| Value | Description |
|---|---|
| 0 | Interrupt vectors are placed at the start of the application section of the Flash. |
| 1 | Interrupt vectors are placed at the start of the boot section of the Flash. |

**Bit 5 – CVT: Compact Vector Table**
This bit is protected by the Configuration Change Protection mechanism.

| Value | Description |
|---|---|
| 0 | Compact Vector Table function is disabled |
| 1 | Compact Vector Table function is enabled |

**Bit 0 – LVL0RR: Round-Robin Priority Enable**
This bit is not protected by the Configuration Change Protection mechanism.

| Value | Description |
|---|---|
| 0 | Priority is fixed for priority level 0 interrupt requests: The lowest interrupt vector address has highest priority. |
| 1 | Round Robin priority scheme is enabled for priority level 0 interrupt requests. |

### 13.5.2 Status

**Name:** STATUS
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | NMIEX | | | | | | LVL1EX | LVL0EX |
| Access | R | | | | | | R | R |
| Reset | 0 | | | | | | 0 | 0 |

**Bit 7 – NMIEX: Non-Maskable Interrupt Executing**
This flag is set if a non-maskable interrupt is executing. The flag is cleared when returning (RETI) from the interrupt handler.

**Bit 1 – LVL1EX: Level 1 Interrupt Executing**
This flag is set when a priority level 1 interrupt is executing, or when the interrupt handler has been interrupted by an NMI. The flag is cleared when returning (RETI) from the interrupt handler.

**Bit 0 – LVL0EX: Level 0 Interrupt Executing**
This flag is set when a priority level 0 interrupt is executing, or when the interrupt handler has been interrupted by a priority level 1 interrupt or an NMI. The flag is cleared when returning (RETI) from the interrupt handler.

### 13.5.3 Interrupt Priority Level 0

**Name:** LVL0PRI
**Offset:** 0x02
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | LVL0PRI[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 – LVL0PRI[7:0]: Interrupt Priority Level 0**
When Round Robin is enabled (LVL0RR bit in CPUINT.CTRLA is '1'), this bit field stores the vector of the last acknowledged priority level 0 (LVL0) interrupt. The stored vector will have the lowest priority next time one or more LVL0 interrupts are pending.

If Round Robin is disabled (LVL0RR in CPUINT.CTRLA is '0'), the vector address based priority scheme (lowest address has highest priority) is governing the priorities of LVL0 interrupt requests.

If a system Reset is asserted, the lowest interrupt vector address will have highest priority within the LVL0.

### 13.5.4 Interrupt Vector with Priority Level 1

**Name:** LVL1VEC
**Offset:** 0x03
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | LVL1VEC[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 – LVL1VEC[7:0]: Interrupt Vector with Priority Level 1**
This bit field contains the address of the single vector with increased priority level 1 (LVL1).

If this bit field has the value 0x00, no vector has LVL1. Consequently, the LVL1 interrupt is disabled.

# 14. EVSYS - Event System

## 14.1 Features

- System for direct peripheral-to-peripheral signaling
- Peripherals can directly produce, use, and react to peripheral events
- Short and guaranteed response time
- Up to six parallel Event channels available; 4 asynchronous- and 2 synchronous
- Channels can be configured to have one triggering peripheral action and multiple peripheral users
- Peripherals can directly trigger and react to Events from other peripherals
- Events can be sent and/or received by most peripherals, and by software
- Works in active mode and standby sleep mode

## 14.2 Overview

The Event System (EVSYS) enables direct peripheral-to-peripheral signaling. It allows a change in one peripheral (the Event Generator) to trigger actions in other peripherals (the Event Users) through Event channels, without using the CPU. It is designed to provide short and predictable response times between peripherals, allowing for autonomous peripheral control and interaction, and also for synchronized timing of actions in several peripheral modules. It is thus a powerful tool for reducing the complexity, size, and execution time of the software.

A change of the Event Generator's state is referred to as an Event, and usually corresponds to one of the peripheral's interrupt conditions. Events can be directly forwarded to other peripherals using the dedicated Event routing network. The routing of each channel is configured in software, including event generation and use.

Only one trigger from an Event generator peripheral can be routed on each channel, but multiple channels can use the same generator source. Multiple peripherals can use events from the same channel.

A channel path can be either asynchronous or synchronous to the main clock. The mode must be selected based on the requirements of the application.

The Event System can directly connect analog and digital converters, analog comparators, I/O port pins, the real-time counter, timer/counters, and the configurable custom logic peripheral. Events can also be generated from software and the peripheral clock.

### 14.2.1 Block Diagram

**Figure 14-1. Block Diagram**

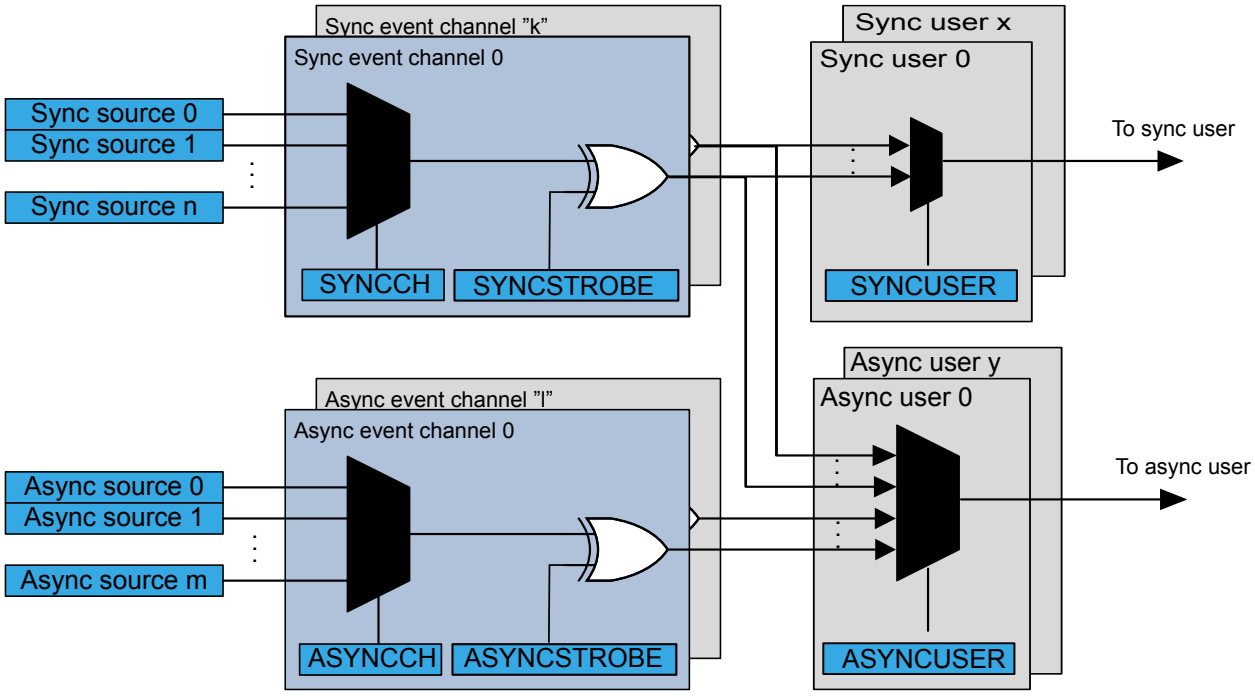
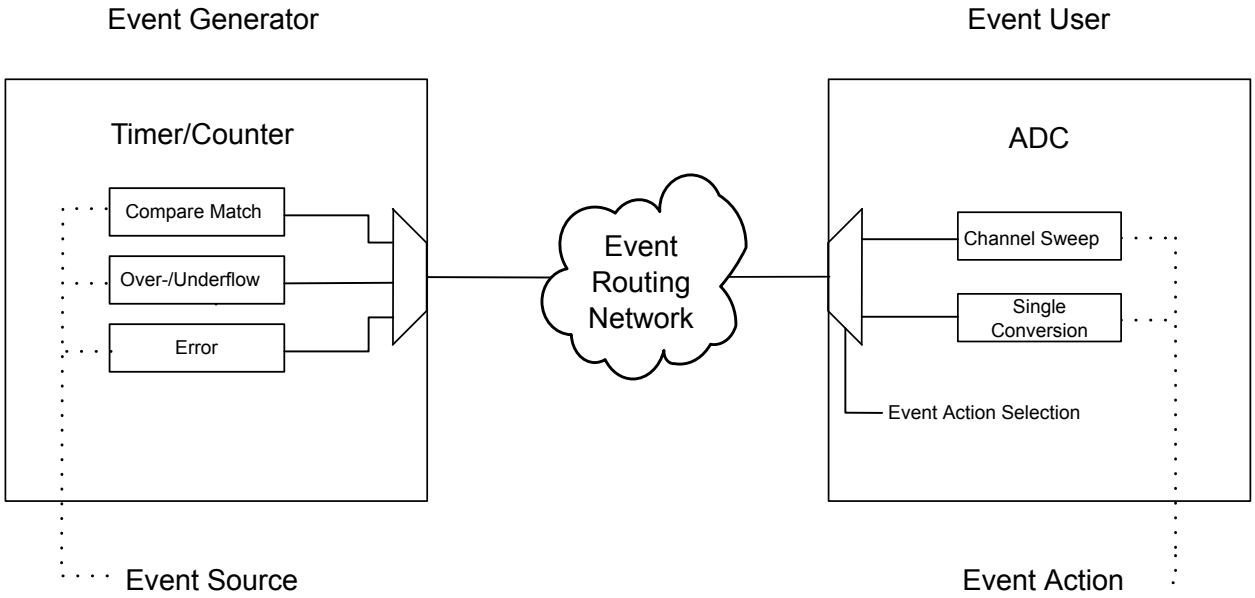



**Figure 14-2. Example of Event Source, Generator, User, and Action**




**Note:**

1. For an overview of peripherals supporting Events, refer the Block Diagram of the device.
2. For a list of Event Generators, refer to the Channel n Generator Selection registers (EVSYS.SYNCCH and EVSYS.ASYNCCH).
3. For a list of Event Users, refer to the User Channel n Input Selection registers (EVSYS.SYNCUSER and EVSYS.ASYNCUSER).

**Related Links**

Block Diagram

SYNCCH0, SYNCCH1
ASYNCCH0, ASYNCCH1, ASYNCCH2, ASYNCCH3
SYNCUSER0, SYNCUSER1
ASYNCUSER0, ASYNCUSER1, ASYNCUSER2, ASYNCUSER3, ASYNCUSER4, ASYNCUSER5, ASYNCUSER6, ASYNCUSER7, ASYNCUSER8, ASYNCUSER9, ASYNCUSER10

### 14.2.2 Signal Description

#### Internal Event Signaling

The Event signaling can either happen synchronously to the main clock (CLK_MAIN), or asynchronously to it.

Depending on the underlying event, the Event signal can be a pulse with a duration of 1 clock cycle, or a level signal (similar to a status flag).

#### Event Output to Pin

| Signal | Type | Description |
|---|---|---|
| EVOUT[2:0] | Digital output | Event output |

**Related Links**

I/O Lines
Block Diagram - CLKCTRL

### 14.2.3 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 14-1. EVSYS System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | Yes | PORTMUX |
| Interrupts | No | - |
| Events | Yes | EVSYS |
| Debug | Yes | UPDI |

**Related Links**

Clocks
Debug Operation

#### 14.2.3.1 Clocks

The EVSYS uses the peripheral clock for I/O registers and software events. When set up correctly, the routing network can be used also in sleep modes without any clock. Software Events will not work in sleep modes where the peripheral clock is halted.

**Related Links**

CLKCTRL - Clock Controller

#### 14.2.3.2 I/O Lines

The EVSYS can output three event channels asynchronously on pins. The output signals are called EVOUT[2:0].

1. Configure which event channel (one of SYNCCH[1:0] or ASYNCCH[3:0]) is output on which the EVOUTn by writing to EVSYS.ASYNCUSER10, EVSYS.ASYNCUSER9 or EVSYS.ASYNCUSER8 respectively.
2. Optional: configure the pin properties using the PORT peripheral.
3. Enable the pin output by writing '1' to the respective EVOUTn bit in the Control A register of the PORTMUX peripheral (PORTMUX.CTRLA).

**Related Links**

PORTMUX - Port Multiplexer

PORT - I/O Pin Configuration

ASYNCUSER0, ASYNCUSER1, ASYNCUSER2, ASYNCUSER3, ASYNCUSER4, ASYNCUSER5, ASYNCUSER6, ASYNCUSER7, ASYNCUSER8, ASYNCUSER9, ASYNCUSER10

## 14.3 Functional Description

### 14.3.1 Initialization

Before enabling Events within the device, the Event Users Multiplexer and Event Channels must be configured.

**Related Links**

Event User Multiplexer Setup

Event System Channel

### 14.3.2 Operation

#### 14.3.2.1 Event User Multiplexer Setup

The Event User multiplexer selects the channel for an Event User. Each Event User has one dedicated Event User multiplexer. Each multiplexer is connected to the supported Event channel outputs, and can be configured to select one of these channels.

Event Users which support asynchronous Events also support synchronous Events. There are also Event Users that support only synchronous Events.

The Event User multiplexers are configured by writing to the corresponding registers:

- Event Users supporting both synchronous and asynchronous Events are configured by writing to the respective asynchronous User Channel Input Selection n register (EVSYS.ASYNCUSERn).
- The users of synchronous-only Events are configured by writing to the respective Synchronous User Channel Input Selection n register (EVSYS.SYNCUSERn).

Default setup of all user multiplexers is Off.

#### 14.3.2.2 Event System Channel

An Event channel can be connected to one of the Event generators. Event channels either support asynchronous generators or synchronous generators.

The source for each asynchronous Event channel is configured by writing to the respective Asynchronous Channel n Input Selection register (EVSYS.ASYNCCHn).

The source for each synchronous Event channel is configured by writing to the respective Synchronous Channel n Input Selection register (EVSYS.SYNCCHn).

#### 14.3.2.3 Event Generators

Each Event channel can receive the Events from several Event generators. For details on Event generation, refer to the documentation of the corresponding peripheral.

For each Event channel, there are several possible Event generators, only one of which can be selected at a time. The Event generator trigger is selected for each channel by writing to the respective channel registers (EVSYS.ASYNCCHn, EVSYS.SYNCCHn). By default, the channels are not connected to any Event generator.

#### 14.3.2.4 Software Event

In a Software Event, the CPU will “strobe” an Event channel by inverting the current value for one system clock cycle.

A Software Event is triggered on a channel by writing a '1' to the respective Strobe bit in the appropriate Channel Strobe register:

- Software Events on asynchronous channel *l* are initiated by writing a '1' to the ASYNCSTROBE[*l*] bit in the Asynchronous Channel Strobe register (EVSYS.ASYNCSTROBE).
- Software Events on synchronous channel *k* are initiated by writing a '1' to the SYNCSTROBE[*k*] bit in the Synchronous Channel Strobe register (EVSYS.SYNCSTROBE).

Software Events are no different to those produced by Event generator peripherals with respect to Event users: when the bit is written to '1', an Event will be generated on the respective channel, and received and processed by the Event User.

### 14.3.3 Interrupts

Not applicable.

### 14.3.4 Sleep Mode Operation

When configured, the Event System will work in all sleep modes. One exception are software Events which require a system clock.

### 14.3.5 Debug Operation

This peripheral is unaffected by entering debug mode.

**Related Links**

UPDI - Unified Program and Debug Interface

### 14.3.6 Synchronization

Asynchronous Events are synchronized and handled by the compatible Event users. Event User peripherals not compatible with asynchronous Events can only be configured to listen to synchronous Event channels.

### 14.3.7 Configuration Change Protection

Not applicable.

**Related Links**

Sequence for Write Operation to Configuration Change Protected I/O Registers

## 14.4 Register Summary - EVSYS

| Offset | Name | Bit Pos. | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | ASYNCSTROBE | 7:0 | ASYNCSTROBE[7:0] | | | | | | | | |
| 0x01 | SYNCSTROBE | 7:0 | SYNCSTROBE[7:0] | | | | | | | | |
| 0x02 | ASYNCCH0 | 7:0 | ASYNCCH[7:0] | | | | | | | | |
| 0x03 | ASYNCCH1 | 7:0 | ASYNCCH[7:0] | | | | | | | | |
| 0x04 | ASYNCCH2 | 7:0 | ASYNCCH[7:0] | | | | | | | | |
| 0x05 | ASYNCCH3 | 7:0 | ASYNCCH[7:0] | | | | | | | | |
| 0x06 ... 0x09 | Reserved | | | | | | | | | | |
| 0x0A | SYNCCH0 | 7:0 | SYNCCH[7:0] | | | | | | | | |
| 0x0B | SYNCCH1 | 7:0 | SYNCCH[7:0] | | | | | | | | |
| 0x0C ... 0x11 | Reserved | | | | | | | | | | |
| 0x12 | ASYNCUSER0 | 7:0 | ASYNCUSER[7:0] | | | | | | | | |
| 0x13 | ASYNCUSER1 | 7:0 | ASYNCUSER[7:0] | | | | | | | | |
| 0x14 | ASYNCUSER2 | 7:0 | ASYNCUSER[7:0] | | | | | | | | |
| 0x15 | ASYNCUSER3 | 7:0 | ASYNCUSER[7:0] | | | | | | | | |
| 0x16 | ASYNCUSER4 | 7:0 | ASYNCUSER[7:0] | | | | | | | | |
| 0x17 | ASYNCUSER5 | 7:0 | ASYNCUSER[7:0] | | | | | | | | |
| 0x18 | ASYNCUSER6 | 7:0 | ASYNCUSER[7:0] | | | | | | | | |
| 0x19 | ASYNCUSER7 | 7:0 | ASYNCUSER[7:0] | | | | | | | | |
| 0x1A | ASYNCUSER8 | 7:0 | ASYNCUSER[7:0] | | | | | | | | |
| 0x1B | ASYNCUSER9 | 7:0 | ASYNCUSER[7:0] | | | | | | | | |
| 0x1C | ASYNCUSER10 | 7:0 | ASYNCUSER[7:0] | | | | | | | | |
| 0x1D ... 0x21 | Reserved | | | | | | | | | | |
| 0x22 | SYNCUSER0 | 7:0 | SYNCUSER[7:0] | | | | | | | | |
| 0x23 | SYNCUSER1 | 7:0 | SYNCUSER[7:0] | | | | | | | | |

## 14.5 Register Description

### 14.5.1 Asynchronous Channel Strobe

**Name:** ASYNCSTROBE
**Offset:** 0x00
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ASYNCSTROBE[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 – ASYNCSTROBE[7:0]: Asynchronous Channel Strobe**
If the strobe register location is written, each Event channel will be inverted for one system clock cycle, i.e. a single Event is generated.

### 14.5.2 Synchronous Channel Strobe

**Name:** SYNCSTROBE
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | SYNCSTROBE[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 – SYNCSTROBE[7:0]: Synchronous Channel Strobe**
If the strobe register location is written, each Event channel will be inverted for one system clock cycle, i.e. a single event is generated.

### 14.5.3 Asynchronous Channel n Generator Selection

**Name:** ASYNCCH0, ASYNCCH1, ASYNCCH2, ASYNCCH3
**Offset:** 0x02 + n*0x01 [n=0..3]
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ASYNCCH[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 – ASYNCCH[7:0]: Asynchronous Channel Generator Selection**

**Table 14-2. Asynchronous Channel Generator selection**

| Value | ASYNCCH0 | ASYNCCH1 | ASYNCCH2 | ASYNCCH3 |
|---|---|---|---|---|
| 0x00 | OFF | OFF | OFF | OFF |
| 0x01 | CCL_LUT0 | | | |
| 0x02 | CCL_LUT1 | | | |
| 0x03 | AC0_OUT | | | |

| Value | ASYNCCH0 | ASYNCCH1 | ASYNCCH2 | ASYNCCH3 |
|---|---|---|---|---|
| 0x04 | TCD0_CMPBCLR | | | |
| 0x05 | TCD0_CMPASET | | | |
| 0x06 | TCD0_CMPBSET | | | |
| 0x07 | TCD0_PROGEV | | | |
| 0x08 | RTC_OVF | | | |
| 0x09 | RTC_CMP | | | |
| 0x0A | PORTA_PIN0 | - | - | PIT_DIV8192 |
| 0x0B | PORTA_PIN1 | - | - | PIT_DIV4096 |
| 0x0C | PORTA_PIN2 | - | - | PIT_DIV2048 |
| 0x0D | PORTA_PIN3 | - | - | PIT_DIV1024 |
| 0x0E | PORTA_PIN4 | - | - | PIT_DIV512 |
| 0x0F | PORTA_PIN5 | - | - | PIT_DIV256 |
| 0x10 | PORTA_PIN6 | - | - | PIT_DIV128 |
| 0x11 | PORTA_PIN7 | - | - | PIT_DIV64 |
| 0x12 | UPDI | - | - | |
| Other | - | - | - | - |

### 14.5.4 Synchronous Channel n Generator Selection

**Name:** SYNCCH0, SYNCCH1
**Offset:** 0x0A + n*0x01 [n=0..1]
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | SYNCCH[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 – SYNCCH[7:0]: Synchronous Channel Generator Selection**

**Table 14-3. Synchronous Channel Generator selection**

| Value | SYNCCH0 | SYNCCH1 |
|---|---|---|
| 0x00 | OFF | |
| 0x01 | TCB0 | |
| 0x02 | TCA0_OVF_LUNF | |
| 0x03 | TCA0_HUNF | |

| Value | SYNCCH0 | SYNCCH1 |
|---|---|---|
| 0x04 | TCA0_CMP0 | |
| 0x05 | TCA0_CMP1 | |
| 0x06 | TCA0_CMP2 | |
| 0x07 | - | - |
| 0x08 | - | - |
| 0x09 | - | - |
| 0x0A | - | - |
| 0x0B | - | - |
| 0x0C | - | - |
| 0x0D | PORTA_PIN0 | - |
| 0x0E | PORTA_PIN1 | - |
| 0x0F | PORTA_PIN2 | - |
| 0x10 | PORTA_PIN3 | |
| 0x11 | PORTA_PIN4 | - |
| 0x12 | PORTA_PIN5 | - |
| 0x13 | PORTA_PIN6 | - |
| 0x14 | PORTA_PIN7 | - |
| Other | - | - |

### 14.5.5 Asynchronous User Channel n Input Selection

**Name:** ASYNCUSER0, ASYNCUSER1, ASYNCUSER2, ASYNCUSER3, ASYNCUSER4, ASYNCUSER5, ASYNCUSER6, ASYNCUSER7, ASYNCUSER8, ASYNCUSER9, ASYNCUSER10
**Offset:** 0x12 + n*0x01 [n=0..10]
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ASYNCUSER[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 – ASYNCUSER[7:0]: Asynchronous User Channel Selection**

**Table 14-4. User Multiplexer Numbers**

| USERn | User Multiplexer | Description |
|---|---|---|
| n=0 | TCB0 | Timer/Counter B 0 |
| n=1 | ADC0 | ADC 0 |
| n=2 | CCL_LUT0EV0 | CCL LUT0 Event 0 |
| n=3 | CCL_LUT1EV0 | CCL LUT1 Event 0 |
| n=4 | CCL_LUT0EV1 | CCL LUT0 Event 1 |
| n=5 | CCL_LUT1EV1 | CCL LUT1 Event 1 |
| n=6 | TCD0_EV0 | Timer Counter D 0 Event 0 |
| n=7 | TCD0_EV1 | Timer Counter D 0 Event 1 |
| n=8 | EVOUT0 | Event OUT 0 |
| n=9 | EVOUT1 | Event OUT 1 |
| n=10 | EVOUT2 | Event OUT 2 |

| Value | Description |
|---|---|
| 0x0 | OFF |
| 0x1 | SYNCCH0 |
| 0x2 | SYNCCH1 |
| 0x3 | ASYNCCH0 |
| 0x4 | ASYNCCH1 |
| 0x5 | ASYNCCH2 |
| 0x6 | ASYNCCH3 |

### 14.5.6 Synchronous User Channel n Input Selection

**Name:** SYNCUSER0, SYNCUSER1
**Offset:** 0x22 + n*0x01 [n=0..1]
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | SYNCUSER[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 – SYNCUSER[7:0]: Synchronous User Channel Selection**

**Table 14-5. User Multiplexer Numbers**

| USERn | User Multiplexer | Description |
|---|---|---|
| n=0 | TCA0 | Timer/Counter A |
| n=1 | USART0 | USART |

| Value | Name |
|---|---|
| 0x0 | OFF |
| 0x1 | SYNCCH0 |
| 0x2 | SYNCCH1 |

# 15. PORTMUX - Port Multiplexer

## 15.1 Overview

The Port Multiplexer (PORTMUX) can either enable or disable functionality of pins, or change between default and alternative pin positions. This depends on the actual pin and property, and is described in detail in the PORTMUX register map.

For available pins and functionalities, refer to the Multiplexed Signals table.

**Related Links**

I/O Multiplexing and Considerations

## 15.2 Register Summary - PORTMUX

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | | | | | | | | EVOUT0 |
| 0x01 | CTRLB | 7:0 | | | | | | SPI0 | | USART0 |

## 15.3 Register Description

### 15.3.1 Control A

**Name:** CTRLA
**Offset:** 0x00
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | EVOUT0 |
| Access | | | | | | | | R/W |
| Reset | | | | | | | | 0 |

**Bit 0 – EVOUT0: Event Output 0**
Write this bit to '1' to enable event output 0.

### 15.3.2 Control B

**Name:** CTRLB
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | SPI0 | | USART0 |
| Access | | | | | | R/W | | R/W |
| Reset | | | | | | 0 | | 0 |

**Bit 2 – SPI0: SPI 0 communication**
Write this bit to '1' to select alternative communication pins for SPI 0.

**Bit 0 – USART0: USART 0 communication**
Write this bit to '1' to select alternative communication pins for USART 0.

# 16. PORT - I/O Pin Configuration

## 16.1 Features

- General purpose input and output pins with individual configuration
- Output driver with configurable inverted I/O and pullup.
- Input with interrupts and events:
  - Sense both edges
  - Sense rising edges
  - Sense falling edges
  - Sense low level
- Asynchronous pin change sensing that can wake the device from all sleep modes
- Efficient and safe access to PORT pins
  - Hardware read-modify-write through dedicated toggle/clear/set registers
  - Mapping of often-used PORT registers into bit-accessible I/O memory space (Virtual Ports)

## 16.2 Overview

The I/O pins of the device are controlled by instances of the PORT peripheral registers. This device has the following instances of the I/O Pin Configuration (PORT): PORTA.

Refer to the I/O Multiplexing table to see which pins are controlled by what instance of PORT. The offsets of the PORT instances and of the corresponding Virtual PORT instances are listed in the Peripherals and Architecture section.

Each of the port pins has a corresponding bit in the Data Direction (PORT.DIR) and Data Output Value (PORT.OUT) registers to enable that pin as an output and to define the output state. For example, pin PA3 is controlled by DIR[3] and OUT[3] of the PORTA instance.

The Data Input Value (PORT.IN) is set as the input value of a PORT pin with resynchronization to the Main Clock. To reduce power consumption, these input synchronizers are not clocked if the Input Sense Configuration bit field (ISC) in PORT.PINnCTRL is INPUT_DISABLE. The value of the pin can always be read, whether the pin is configured as input or output.

The PORT also supports synchronous and asynchronous input sensing with interrupts for selectable pin change conditions. Asynchronous pin-change sensing means that a pin change can wake the device from all sleep modes, including the modes where no clocks are running.

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

The PORT pin configuration also controls input and output selection of other device functions.

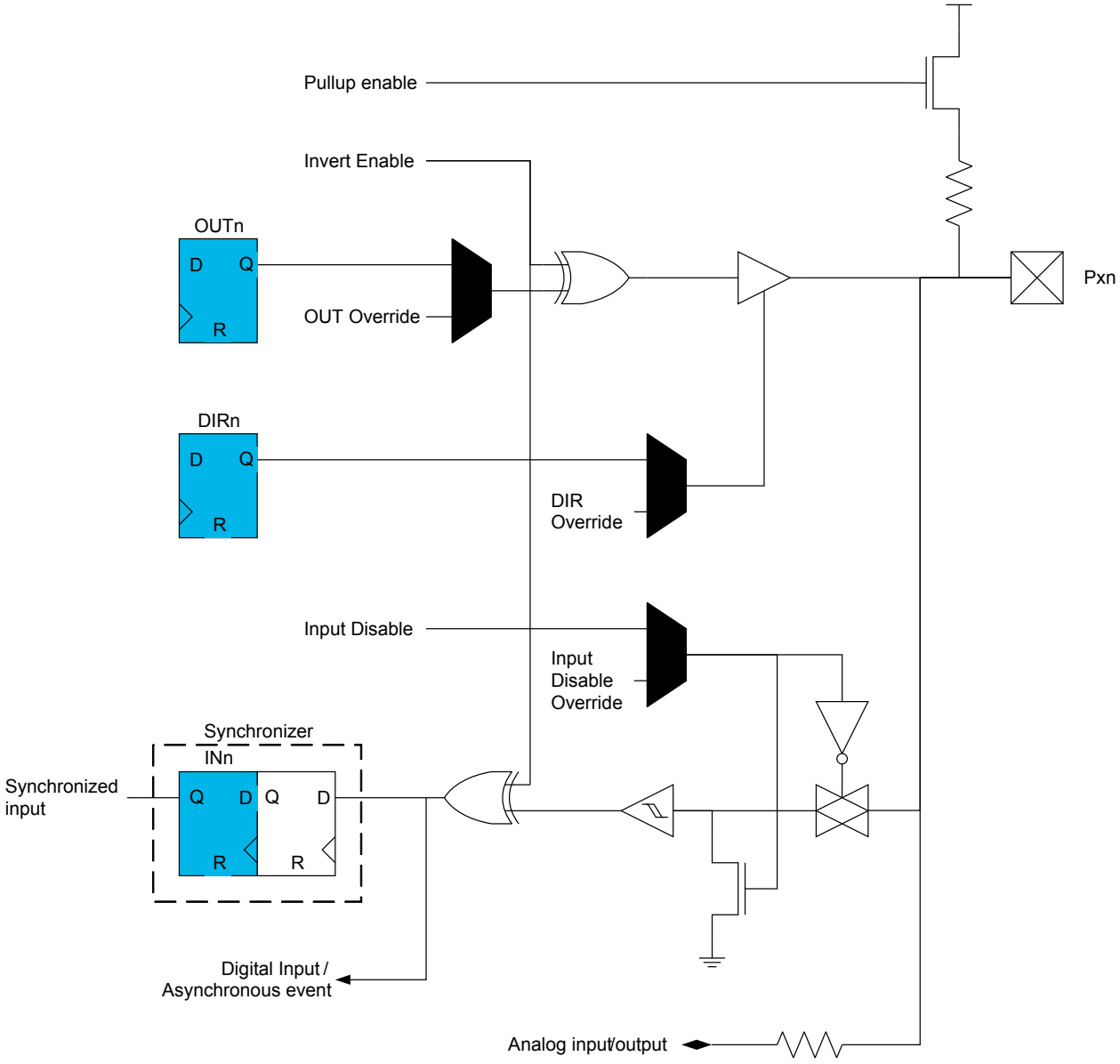
**Related Links**

I/O Multiplexing and Considerations

Peripherals and Architecture

### 16.2.1 Block Diagram

**Figure 16-1. PORT Block Diagram**




### 16.2.2 Signal Description

| Signal | Type | Description |
|---|---|---|
| EXTINT | Digital input | External interrupt - available on all I/O pins |

**Related Links**

I/O Multiplexing and Considerations

### 16.2.3 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 16-1. PORT System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | No | - |
| Interrupts | Yes | CPUINT |
| Events | Yes | EVSYS |
| Debug | No | - |

**Related Links**

Events
Clocks
Interrupts

#### 16.2.3.1 Clocks

This peripheral depends on the peripheral clock.

#### 16.2.3.2 I/O Lines and Connections

Not applicable.

#### 16.2.3.3 Interrupts

Using the interrupts of this peripheral requires the Interrupt Controller to be configured first.

**Related Links**

CPUINT - CPU Interrupt Controller
Interrupts
SREG

#### 16.2.3.4 Events

The events of this peripheral are connected to the Event System.

**Related Links**

EVSYS - Event System

#### 16.2.3.5 Debug Operation

This peripheral is unaffected by entering debug mode.

## 16.3 Functional Description

### 16.3.1 Initialization

After Reset, all standard function device I/O pads are connected to the PORT with outputs tri-stated and input buffers enabled, even if there is no clock running.

For best power consumption, disable the input of unused pins and pins that are used as analog inputs or outputs.

Specific pins, such as those used for connecting a debugger, may be configured differently, as required by their special function.

### 16.3.2 Operation

#### 16.3.2.1 Basic Functions

Each I/O pin Pxn can be controlled by the registers in PORT x. Each pin group x has its own set of PORT registers, the base address of the register set for pin n is at the byte address $\mathrm{PORT} + 0\mathrm{x}10 + n$ . The index within that register set is n.

To use pin number n as an output only, write bit n of the PORT.DIR register to '1'. This can also be done by writing bit n in the PORT.DIRSET register to '1' - this will avoid disturbing the configuration of other pins in that group. The $n^{th}$ bit in the PORT.OUT register must be written to the desired output value.

Similarly, writing a PORT.OUTSET bit to '1' will set the corresponding bit in the PORT.OUT register to '1'. Writing a bit in PORT.OUTCLR to '1' will clear that bit in PORT.OUT to zero. Writing a bit in PORT.OUTTGL or PORT.IN to '1' will toggle that bit in PORT.OUT.

To use pin n as an input, bit n in the PORT.DIR register must be written to '0' to disable the output driver. This can also be done by writing bit n in the PORT.DIRCLR register to '1' - this will avoid disturbing the configuration of other pins in that group. The input value can be read from bit n in register PORT.IN as long as the ISC bit is not set to INPUT_DISABLE.

Writing a bit to '1' in PORT.DIRTGL will toggle that bit in PORT.DIR, and toggle the direction of the corresponding pin.

#### 16.3.2.2 Virtual Ports

The Virtual Port registers map the most frequently used regular Port registers into the bit-accessible I/O space. Writing to the Virtual Port registers has the same effect as writing to the regular registers, but allows for memory-specific instructions, such as bit-manipulation instructions, which are not valid for the extended I/O memory space where the regular Port registers reside.

**Table 16-2. Virtual Port Mapping**

| Regular Port Register | Mapped to Virtual Port Register |
|---|---|
| PORT.DIR | VPORT.DIR |
| PORT.OUT | VPORT.OUT |
| PORT.IN | VPORT.IN |
| PORT.INTFLAG | VPORT.INTFLAG |

**Related Links**

Register Summary - VPORT

I/O Multiplexing and Considerations

Peripherals and Architecture

#### 16.3.2.3 Pin Configuration

The Pin n Configuration register (PORT.PINnCTRL) is used to configure inverted I/O, pullup, and input sensing of a pin.

All input and output on the respective pin n can be inverted by writing a '1' to the Inverted I/O Enable bit (INVEN) in PORT.PINnCTRL.

Toggling the INVEN bit causes an edge on the pin, which can be detected by all peripherals using this pin, and is seen by interrupts or Events if enabled.

Pullup of pin n is enabled by writing a '1' to the Pullup Enable bit (PULLUPEN) in PORT.PINnCTRL.

© 2017 Microchip Technology Inc.

Changes of the signal on a pin can trigger an interrupt. The exact conditions are defined by writing to the Input/Sense bit field (ISC) in PORT.PINnCTRL.

When setting or changing interrupt settings, take these points into account:

- If an INVEN bit is toggled in the same cycle as the interrupt setting, the edge caused by the inversion toggling may not cause an interrupt request.
- If an input is disabled while synchronizing an interrupt, that interrupt may be requested on re-enabling the input, even if it is re-enabled with a different interrupt setting.
- If the interrupt setting is changed while synchronizing an interrupt, that interrupt may not be accepted.
- Only few pins support full asynchronous interrupt detection, see I/O Multiplexing and Considerations. These limitations apply for waking the system from sleep:

| Interrupt Type | Fully Asynchronous Pins | Other Pins |
|---|---|---|
| BOTHEDGES | will wake system | will wake system |
| RISING | will wake system | will not wake system |
| FALLING | will wake system | will not wake system |
| LEVEL | will wake system | will wake system |

**Related Links**

I/O Multiplexing and Considerations

### 16.3.3 Interrupts

**Table 16-3. Available Interrupt Vectors and Sources**

| Offset | Name | Vector Description | Conditions |
|---|---|---|---|
| 0x00 | PORTx | PORT A, B, C interrupt | INTn in PORT.INTFLAGS is raised as configured by ISC bit in PORT.PINnCTRL. |

Each PORT pin n can be configured as an interrupt source. Each interrupt can be individually enabled or disabled by writing to ISC in PORT.PINCTRL.

When an interrupt condition occurs, the corresponding Interrupt Flag is set in the Interrupt Flags register of the peripheral (*peripheral*.INTFLAGS).

An interrupt request is generated when the corresponding interrupt is enabled and the Interrupt Flag is set. The interrupt request remains active until the Interrupt Flag is cleared. See the peripheral's INTFLAGS register for details on how to clear Interrupt Flags.

**Asynchronous Sensing Pin Properties**

**Table 16-4. Behavior Comparison of Fully/Partly Asynchronous Sense Pin**

| Property | Synchronous or Partly Asynchronous Sense Support | Full Asynchronous Sense Support |
|---|---|---|
| Minimum pulse width to trigger interrupt | Minimum one system clock cycle | Less than a system clock cycle |
| Waking the device from sleep | From all interrupt sense configurations from sleep modes with Main Clock running. Only from BOTHEDGES or LEVEL interrupt sense | From all interrupt sense configurations from all sleep modes |

© 2017 Microchip Technology Inc.

| Property | Synchronous or Partly Asynchronous Sense Support | Full Asynchronous Sense Support |
|---|---|---|
| | configuration from sleep modes with Main Clock stopped. | |
| Interrupt "dead time" | No new interrupt for three cycles after the previous | No limitation |
| Minimum Wakeup pulse length | Value on pad must be kept until the system clock has restarted | No limitation |

**Related Links**

AVR CPU

SREG

### 16.3.4 Sleep Mode Operation

With the exception of interrupts and input synchronization, all pin configurations are independent of sleep mode. Peripherals connected to the Ports can be affected by sleep modes, described in the respective peripherals' documentation.

The PORT peripheral will always use the Main Clock. Input synchronization will halt when this clock stops.

### 16.3.5 Synchronization

Not applicable.

### 16.3.6 Configuration Change Protection

Not applicable.

## 16.4 Register Summary - PORT

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | DIR | 7:0 | DIR[7:0] | | | | | | | |
| 0x01 | DIRSET | 7:0 | DIRSET[7:0] | | | | | | | |
| 0x02 | DIRCLR | 7:0 | DIRCLR[7:0] | | | | | | | |
| 0x03 | DIRTGL | 7:0 | DIRTGL[7:0] | | | | | | | |
| 0x04 | OUT | 7:0 | OUT[7:0] | | | | | | | |
| 0x05 | OUTSET | 7:0 | OUTSET[7:0] | | | | | | | |
| 0x06 | OUTCLR | 7:0 | OUTCLR[7:0] | | | | | | | |
| 0x07 | OUTTGL | 7:0 | OUTTGL[7:0] | | | | | | | |
| 0x08 | IN | 7:0 | IN[7:0] | | | | | | | |
| 0x09 | INTFLAGS | 7:0 | INT[7:0] | | | | | | | |
| 0x0A ... 0x0F | Reserved | | | | | | | | | |
| 0x10 | PIN0CTRL | 7:0 | INVEN | | | | PULLUPEN | ISC[2:0] | | |
| 0x11 | PIN1CTRL | 7:0 | INVEN | | | | PULLUPEN | ISC[2:0] | | |
| 0x12 | PIN2CTRL | 7:0 | INVEN | | | | PULLUPEN | ISC[2:0] | | |
| 0x13 | PIN3CTRL | 7:0 | INVEN | | | | PULLUPEN | ISC[2:0] | | |
| 0x14 | PIN4CTRL | 7:0 | INVEN | | | | PULLUPEN | ISC[2:0] | | |
| 0x15 | PIN5CTRL | 7:0 | INVEN | | | | PULLUPEN | ISC[2:0] | | |
| 0x16 | PIN6CTRL | 7:0 | INVEN | | | | PULLUPEN | ISC[2:0] | | |
| 0x17 | PIN7CTRL | 7:0 | INVEN | | | | PULLUPEN | ISC[2:0] | | |

## 16.5 Register Description - Ports

### 16.5.1 Data Direction

**Name:** DIR
**Offset:** 0x00
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DIR[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 – DIR[7:0]: Data Direction**
This bit field selects the data direction for the individual pins n of the Port.
Writing a '1' to PORT.DIR[n] configures and enables pin n as output pin.

Writing a '0' to PORT.DIR[n] configures pin n as input pin. It can be configured by writing to the ISC bit in PORT.PINnCTRL.

### 16.5.2 Data Direction Set

**Name:** DIRSET
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DIRSET[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 – DIRSET[7:0]: Data Direction Set**
This bit field can be used instead of a read-modify-write to set individual pins as output.
Writing a '1' to DIRSET[n] will set the corresponding PORT.DIR[n] bit.

Reading this bit field will always return the value of PORT.DIR.

### 16.5.3 Data Direction Clear

**Name:** DIRCLR
**Offset:** 0x02
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DIRCLR[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 – DIRCLR[7:0]: Data Direction Clear**
This register can be used instead of a read-modify-write to configure individual pins as input.
Writing a '1' to DIRCLR[n] will clear the corresponding bit in PORT.DIR.

Reading this bit field will always return the value of PORT.DIR.

### 16.5.4 Data Direction Toggle

**Name:** DIRTGL
**Offset:** 0x03
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DIRTGL[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 – DIRTGL[7:0]: Data Direction Toggle**
This bit field can be used instead of a read-modify-write to toggle the direction of individual pins.

Writing a '1' to DIRTGL[n] will toggle the corresponding bit in PORT.DIR.

Reading this bit field will always return the value of PORT.DIR.

### 16.5.5 Output Value

**Name:** OUT
**Offset:** 0x04
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OUT[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 – OUT[7:0]: Output Value**
This bit field defines the data output value for the individual pins n of the port.
If OUT[n] is written to '1', pin n is driven high.

If OUT[n] is written to '0', pin n is driven low.

In order to have any effect, the pin direction must be configured as output.

### 16.5.6 Output Value Set

**Name:** OUTSET
**Offset:** 0x05
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OUTSET[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 – OUTSET[7:0]: Output Value Set**
This bit field can be used instead of a read-modify-write to set the output value of individual pins to '1'.
Writing a '1' to OUTSET[n] will set the corresponding bit in PORT.OUT.

Reading this bit field will always return the value of PORT.OUT.

### 16.5.7 Output Value Clear

**Name:** OUTCLR
**Offset:** 0x06
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OUTCLR[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 – OUTCLR[7:0]: Output Value Clear**
This register can be used instead of a read-modify-write to clear the output value of individual pins to '0'. Writing a '1' to OUTCLR[n] will clear the corresponding bit in PORT.OUT.

Reading this bit field will always return the value of PORT.OUT.

### 16.5.8 Output Value Toggle

**Name:** OUTTGL
**Offset:** 0x07
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OUTTGL[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 – OUTTGL[7:0]: Output Value Toggle**
This register can be used instead of a read-modify-write to toggle the output value of individual pins. Writing a '1' to OUTTGL[n] will toggle the corresponding bit in PORT.OUT.

Reading this bit field will always return the value of PORT.OUT.

### 16.5.9 Input Value

**Name:** IN
**Offset:** 0x08
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | IN[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 – IN[7:0]: Input Value**
This register shows the value present on the pins if the digital input driver is enabled. IN[n] shows the value of pin n of the Port. The input is not sampled and cannot be read if the digital input buffers are disabled.
Writing to a bit of PORT.IN will toggle the corresponding bit in PORT.OUT.

### 16.5.10 Interrupt Flags

**Name:** INTFLAGS
**Offset:** 0x09
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | INT[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 – INT[7:0]: Interrupt Pin Flag**
The INT Flag is set when a pin change/state matches the pin's input sense configuration.
Writing a '1' to a flag's bit location will clear the flag.

For enabling and executing the interrupt, refer to ISC bit description in PORT.PINnCTRL.

### 16.5.11 Pin n Control

**Name:** PIN0CTRL, PIN1CTRL, PIN2CTRL, PIN3CTRL, PIN4CTRL, PIN5CTRL, PIN6CTRL, PIN7CTRL
**Offset:** 0x10 + n*0x01 [n=0..7]
**Reset:** 0x00
**Property:** -

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

**Bit 7 – INVEN: Inverted I/O Enable**

| Value | Description |
|---|---|
| 0 | I/O on pin n not inverted |
| 1 | I/O on pin n inverted |

**Bit 3 – PULLUPEN: Pullup Enable**

| Value | Description |
|---|---|
| 0 | Pullup disabled for pin n |
| 1 | Pullup enabled for pin n |

**Bits 2:0 – ISC[2:0]: Input/Sense Configuration**
These bits configure the input and sense configuration of pin n. The sense configuration determines how a port interrupt can be triggered. If the input buffer is disabled, the input cannot be read in the IN register.

| Value | Name | Description |
|---|---|---|
| 0x0 | INTDISABLE | Interrupt disabled but input buffer enabled |
| 0x1 | BOTHEDGES | Sense both edges |
| 0x2 | RISING | Sense rising edge |
| 0x3 | FALLING | Sense falling edge |
| 0x4 | INPUT_DISABLE | Digital input buffer disabled |
| 0x5 | LEVEL | Sense low level |
| other | - | Reserved |

## 16.6 Register Summary - VPORT

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | DIR | 7:0 | DIR[7:0] | | | | | | | |
| 0x01 | OUT | 7:0 | OUT[7:0] | | | | | | | |
| 0x02 | IN | 7:0 | IN[7:0] | | | | | | | |
| 0x03 | INTFLAGS | 7:0 | INT[7:0] | | | | | | | |

## 16.7 Register Description - Virtual Ports

### 16.7.1 Data Direction

Writing to the Virtual Port registers has the same effect as writing to the regular registers, but allows for memory-specific instructions, such as bit-manipulation instructions, which are not valid for the extended I/O memory space where the regular Port registers reside.

**Name:** DIR
**Offset:** 0x00
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DIR[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 – DIR[7:0]: Data Direction**
This bit field selects the data direction for the individual pins in the Port.

### 16.7.2 Output Value

Writing to the Virtual Port registers has the same effect as writing to the regular registers, but allows for memory-specific instructions, such as bit-manipulation instructions, which are not valid for the extended I/O memory space where the regular Port registers reside.

**Name:** OUT
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | OUT[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 – OUT[7:0]: Output Value**
This bit field selects the data output value for the individual pins in the Port.

### 16.7.3 Input Value

Writing to the Virtual Port registers has the same effect as writing to the regular registers, but allows for memory-specific instructions, such as bit-manipulation instructions, which are not valid for the extended I/O memory space where the regular Port registers reside.

**Name:** IN
**Offset:** 0x02
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | IN[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 – IN[7:0]: Input Value**
This bit field holds the value present on the pins if the digital input buffer is enabled.
Writing to a bit of VPORT.IN will toggle the corresponding bit in VPORT.OUT.

### 16.7.4 Interrupt Flag

Writing to the Virtual Port registers has the same effect as writing to the regular registers, but allows for memory-specific instructions, such as bit-manipulation instructions, which are not valid for the extended I/O memory space where the regular Port registers reside.w

**Name:** INTFLAGS
**Offset:** 0x03
**Reset:** 0x00
**Property:** -

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

**Bits 7:0 – INT[7:0]: Interrupt Pin Flag**
The INT flag is set when a pin change/state matches the pin's input sense configuration, and the pin is configured as source for port interrupt.
Writing a '1' to this flag's bit location will clear the flag.

For enabling and executing the interrupt, refer to PORT_PINnCTRL.ISC.

# 17. BOD - Brownout Detector

## 17.1 Features

- Brownout detection - monitors the power supply to avoid operation below a programmable level
- Three modes:
  - Enabled
  - Sampled
  - Disabled
- Separate selection of mode for Active mode and sleep modes
- Voltage level monitor (VLM) with interrupt
- Programmable VLM level relative to the BOD level

## 17.2 Overview

The Brownout Detector (BOD) monitors the power supply and compares the voltage with two programmable brownout threshold levels: the brownout threshold level defines when to generate a Reset. A voltage level monitor (VLM) monitors the power supply and compares it to a threshold higher than the BOD threshold. The VLM can then generate an interrupt request as an "early warning" when the supply voltage is about to drop below the VLM threshold. The VLM threshold level is expressed in percentage above the BOD threshold level.

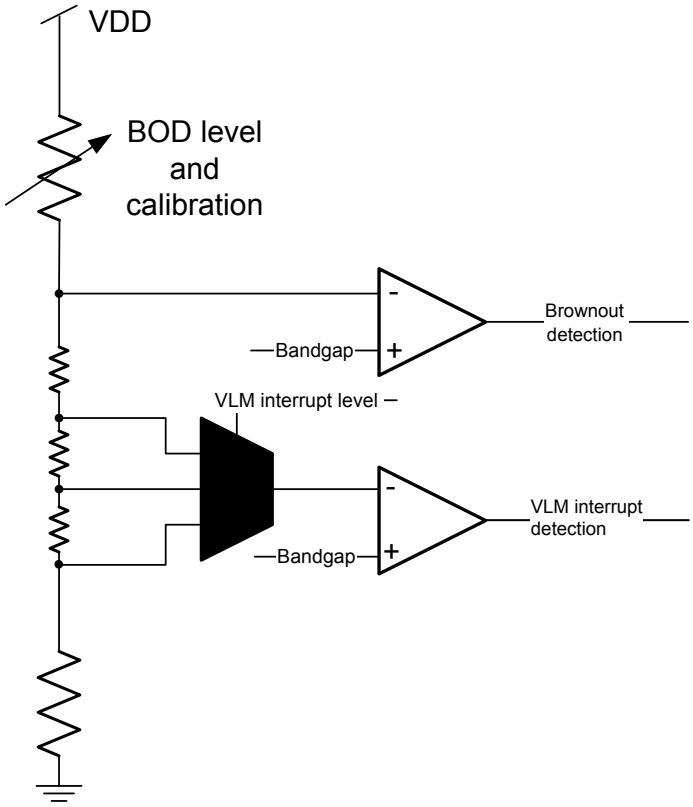
The BOD is mainly controlled by fuses. The mode used in Standby sleep mode and Power Down sleep mode can be altered in normal program execution. The VLM part of the BOD is controlled by I/O registers as well.

When activated, The BOD can operate in Enabled mode, where the BOD is continuously active, and in Sampled mode, where the BOD is activated briefly at a given period to check the supply voltage level.

© 2017 Microchip Technology Inc.

### 17.2.1 Block Diagram

**Figure 17-1. BOD Block Diagram**




### 17.2.2 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 17-1. BOD System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | No | - |
| Interrupts | Yes | CPUINT |
| Events | Yes | EVSYS |
| Debug | Yes | UPDI |

**Related Links**

Clocks

Debug Operation

Interrupts

Events

#### 17.2.2.1 Clocks

The BOD uses the 32KHz oscillator (OSCULP32K) as clock source for CLK_BOD.

#### 17.2.2.2 I/O Lines and Connections

Not applicable.

#### 17.2.2.3 Interrupts

Using the interrupts of this peripheral requires the Interrupt Controller to be configured first.

**Related Links**

CPUINT - CPU Interrupt Controller

SREG

Interrupts

#### 17.2.2.4 Events

Not applicable.

#### 17.2.2.5 Debug Operation

This peripheral is unaffected by entering debug mode.

The VLM interrupt will not be executed if the CPU is halted in debug mode.

If the peripheral is configured to require periodical service by the CPU through interrupts or similar, improper operation or data loss may result during halted debugging.

## 17.3 Functional Description

### 17.3.1 Initialization

The BOD settings are loaded from fuses during reset. The BOD level and operating mode in Active and Idle sleep mode are set by fuses and cannot be changed by the CPU. The operating mode in Standby and Power Down sleep mode is loaded from fuses and can be changed by software.

The Voltage Level Monitor function can be enabled by writing a '1' to the VLM Interrupt Enable bit (VLMIE) in the Interrupt Control register (BOD.INTCTRL). The VLM interrupt is configured by writing the VLM Configuration bits (VLMCFG) in BOD.INTCTRL. An interrupt is requested when the supply voltage crosses the VLM threshold either from above, from below, or from any direction.

The VLM functionality will follow the BOD mode. If the BOD is turned off, the VLM will not be enabled, even if the VLMIE is '1'. If the BOD is using sampled mode, the VLM will also be sampled. When enabling VLM interrupt, the interrupt flag will always be set if VLMCFG equals 0x2 and may be set if VLMCFG is configured to 0x0 or 0x1.

The VLM threshold is defined by writing the VLM Level bits (VLMLVL) in the Control A register (BOD.VLMCTRLA).

If the BOD/VLM is enabled in sampled mode, only VLMCFG=0x1 (crossing threshold from above) in BOD.INTCTRL will trigger an interrupt.

### 17.3.2 Interrupts

**Table 17-2. Available Interrupt Vectors and Sources**

| Offset | Name | Vector Description | Conditions |
|---|---|---|---|
| 0x00 | VLM | Voltage Level Monitor | Supply voltage crossing the VLM threshold as configured by VLMCFG in BOD.INTCTRL |

When an interrupt condition occurs, the corresponding Interrupt Flag is set in the Interrupt Flags register of the peripheral (*peripheral*.INTFLAGS).

An interrupt source is enabled or disabled by writing to the corresponding enable bit in the peripheral's Interrupt Control register (*peripheral*.INTCTRL).

An interrupt request is generated when the corresponding interrupt source is enabled and the Interrupt Flag is set. The interrupt request remains active until the Interrupt Flag is cleared. See the peripheral's INTFLAGS register for details on how to clear Interrupt Flags.

**Related Links**

AVR CPU

SREG

### 17.3.3 Sleep Mode Operation

There are two separate fuses defining the BOD configuration in different sleep modes: One fuse defines the mode used in Active mode and Idle sleep mode (ACTIVE in FUSE.BODCFG), and is written to the ACTIVE bits in the Control A register (BOD.CTRLA). The second fuse (SLEEP in FUSE.BODCFG) selects the mode used in Standby sleep mode and Power Down sleep mode, and is loaded into the SLEEP bits in the Control A register (BOD.CTRLA).

The operating mode in Active mode and Idle sleep mode (i.e. ACTIVE in BOD.CTRLA) cannot be altered by software. The operating mode in Standby sleep mode and Power Down sleep mode can be altered by writing to the SLEEP bits in the Control A register (BOD.CTRLA).

When the device is going into Standby sleep mode or Power Down sleep mode, the BOD will change operation mode as defined by SLEEP in BOD.CTRLA. When the device is waking up from Standby or Power Down sleep mode, the BOD will operate in the mode defined by the ACTIVE bit field in BOD.CTRLA.

### 17.3.4 Synchronization

Not applicable.

### 17.3.5 Configuration Change Protection

This peripheral has registers that are under Configuration Change Protection (CCP). In order to write to these, a certain key must be written to the CPU.CCP register first, followed by a write access to the protected bits within four CPU instructions.

It is possible to try writing to these registers any time, but the values are not altered.

The following registers are under CCP:

**Table 17-3. Registers under Configuration Change Protection**

| Register | Key |
|---|---|
| SLEEP in BOD.CTRLA | IOREG |

**Related Links**

Sequence for Write Operation to Configuration Change Protected I/O Registers

## 17.4 Register Summary - BOD

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | | | | SAMPFREQ | ACTIVE[1:0] | | SLEEP[1:0] | |
| 0x01 | CTRLB | 7:0 | | | | | | LVL[2:0] | | |
| 0x02<br>...<br>0x07 | Reserved | | | | | | | | | |
| 0x08 | VLMCTRLA | 7:0 | | | | | | | VLMLVL[1:0] | |
| 0x09 | INTCTRL | 7:0 | | | | | | VLMCFG[1:0] | | VLMIE |
| 0x0A | INTFLAGS | 7:0 | | | | | | | | VLMIF |
| 0x0B | STATUS | 7:0 | | | | | | | | VLMS |

## 17.5 Register Description

### 17.5.1 Control A

**Name:** CTRLA
**Offset:** 0x00
**Reset:** Loaded from fuse
**Property:** Configuration Change Protection

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | SAMPFREQ | ACTIVE[1:0] | | SLEEP[1:0] | |
| Access | R | R | R | R | R | R | R/W | R/W |
| Reset | 0 | 0 | 0 | x | x | x | x | x |

**Bit 4 – SAMPFREQ: Sample Frequency**
This bit selects the BOD sample frequency.

The Reset value is loaded from the SAMPFREQ bit in FUSE.BODCFG.This bit is under Configuration Change Protection (CCP).

| Value | Description |
|---|---|
| 0x0 | Sample frequency is 1kHz |
| 0x1 | Sample frequency is 125Hz |

**Bits 3:2 – ACTIVE[1:0]: Active**
These bits selects the BOD operation mode when the device is in active or idle mode.

The Reset value is loaded from the ACTIVE bits in FUSE.BODCFG.These bits are not under Configuration Change Protection (CCP).

| Value | Description |
|---|---|
| 0x0 | Disabled |
| 0x1 | Enabled |
| 0x2 | Sampled |
| 0x3 | Enabled with wake-up halted until BOD is ready |

**Bits 1:0 – SLEEP[1:0]: Sleep**
These bits select the BOD operation mode when the device is in standby or power-down sleep mode. These bits are under Configuration Change Protection (CCP).

The Reset value is loaded from the SLEEP bits in FUSE.BODCFG.

| Value | Description |
|---|---|
| 0x0 | Disabled |
| 0x1 | Enabled |
| 0x2 | Sampled |
| 0x3 | Reserved |

### 17.5.2 Control B

**Name:** CTRLB
**Offset:** 0x01
**Reset:** Loaded from fuse
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | LVL[2:0] | | |
| Access | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | x | x | x |

**Bits 2:0 – LVL[2:0]: BOD Level**
These bits selects the BOD threshold level.

The Reset value is loaded from the BOD Level bits (LVL) in the BOD Configuration fuse (FUSE.BODCFG).

| Value | Name | Description |
|---|---|---|
| 0x0 | BODLEVEL0 | 1.8V |
| 0x1 | BODLEVEL1 | 2.15V |
| 0x2 | BODLEVEL2 | 2.60V |
| 0x3 | BODLEVEL3 | 2.95V |
| 0x4 | BODLEVEL4 | 3.30V |
| 0x5 | BODLEVEL5 | 3.70V |
| 0x6 | BODLEVEL6 | 4.00V |
| 0x7 | BODLEVEL7 | 4.30V |

### 17.5.3 VLM Control A

**Name:** VLMCTRLA
**Offset:** 0x08
**Reset:** 0x00
**Property:** -

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

**Bits 1:0 – VLMLVL[1:0]: VLM Level**
These bit select the Voltage Level Monitor threshold relative to the BOD threshold (LVL in BOD.CTRLB).

| Value | Description |
|---|---|
| 0x0 | VLM threshold 5% above BOD threshold |
| 0x1 | VLM threshold 15% above BOD threshold |
| 0x2 | VLM threshold 25% above BOD threshold |
| other | Reserved |

### 17.5.4 Interrupt Control

**Name:** INTCTRL
**Offset:** 0x09
**Reset:** 0x00
**Property:** -

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

**Bits 2:1 – VLMCFG[1:0]: VLM Configuration**
These bits select which incidents will trigger a VLM interrupt.

| Value | Description |
|---|---|
| 0x0 | Voltage crosses VLM threshold from above |
| 0x1 | Voltage crosses VLM threshold from below |
| 0x2 | Either direction is triggering an interrupt request |
| Other | Reserved |

**Bit 0 – VLMIE: VLM Interrupt Enable**
Writing a '1' to this bit enables the Voltage Level Monitor (VLM) interrupt.

### 17.5.5 VLM Interrupt Flags

**Name:** INTFLAGS
**Offset:** 0x0A
**Reset:** 0x00
**Property:** -

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

**Bit 0 – VLMIF: VLM Interrupt Flag**
This flag is set when a trigger from the VLM is given, as configured by the VLMCFG bit in the BOD.INTCTRL register. The flag is only updated when the BOD is enabled.

### 17.5.6 VLM Status

**Name:** STATUS
**Offset:** 0x0B
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | VLMS |
| Access | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bit 0 – VLMS: VLM Status**
This bit is only valid when the bod is enabled.

| Value | Description |
|---|---|
| 0 | The voltage is above the VLM threshold level |
| 1 | The voltage is below the VLM threshold level |

# 18. VREF - Voltage Reference

## 18.1 Features

- programmable voltage reference sources
  - One for each ADC peripheral
  - One for each AC and DAC peripheral
- Each reference source supports five different voltages:
  - 0.55V
  - 1.1V
  - 1.5V
  - 2.5V
  - 4.3V

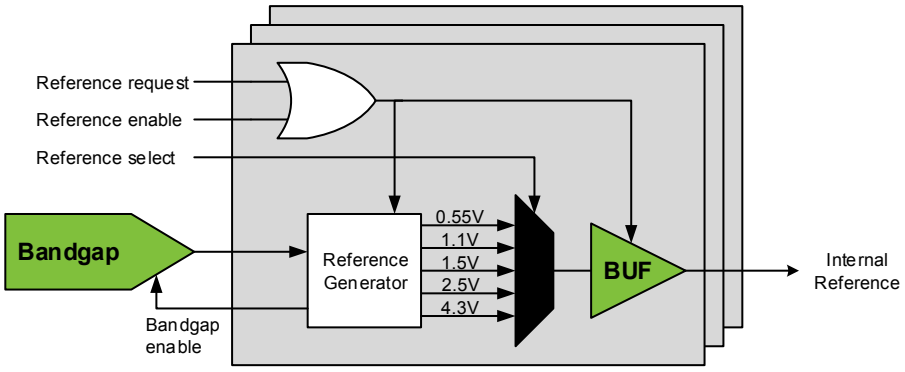
## 18.2 Overview

The Voltage Reference peripheral (VREF) provides control registers for the voltage reference sources used by several peripherals. The user can select the reference voltages for the ADC0 by writing to the ADC0 Reference Select bit field (ADC0REFSEL) in the Control A register (VREF.CTRLA), and for both AC0 and DAC0 by writing to the DAC0 and AC0 Reference Select bit field DAC0REFSEL in VREF.CTRLA.

A voltage reference source is enabled automatically when requested by a peripheral. The user can enable the reference voltage sources (and thus, override the automatic disabling of unused sources) by writing to the respective Force Enable bit (ADC0REFEN, DAC0REFEN) in the Control B register (VREF.CTRLB). This may be desirable to decrease start-up time, at the cost of increased power consumption.

### 18.2.1 Block Diagram

**Figure 18-1. Block Diagram VREF**




## 18.3 Functional Description

### 18.3.1 Initialization

The default configuration will enable the respective source when the ADC0, AC0 or DAC0 is requesting a reference voltage. The default reference voltages are 0.55V, but can be configured by writing to the

respective Reference Select bit field (ADC0REFSEL, DAC0REFSEL) in the Control A register (VREF.CTRLA).

## 18.4 Register Summary - VREF

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | | ADC0REFSEL[2:0] | | | | DAC0REFSEL[2:0] | | |
| 0x01 | CTRLB | 7:0 | | | | | | | ADC0REFEN | DAC0REFEN |

## 18.5 Register Description

### 18.5.1 Control A

**Name:** CTRLA
**Offset:** 0x00
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | ADC0REFSEL[2:0] | | | | DAC0REFSEL[2:0] | | |
| Access | | R/W | R/W | R/W | | R/W | R/W | R/W |
| Reset | | 0 | 0 | 0 | | 0 | 0 | 0 |

**Bits 6:4 – ADC0REFSEL[2:0]: ADC0 Reference Select**
These bits select the reference voltage for the ADC0.

| Value | Description |
|---|---|
| 0x0 | 0.55V |
| 0x1 | 1.1V |
| 0x2 | 2.5V |
| 0x3 | 4.3V |
| 0x4 | 1.5V |
| other | Reserved |

**Bits 2:0 – DAC0REFSEL[2:0]: DAC0 and AC0 Reference Select**
These bits select reference voltage for the DAC0 and AC0.

| Value | Description |
|---|---|
| 0x0 | 0.55V |
| 0x1 | 1.1V |
| 0x2 | 2.5V |
| 0x3 | 4.3V |
| 0x4 | 1.5V |
| other | Reserved |

### 18.5.2 Control B

**Name:** CTRLB
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | ADC0REFEN | DAC0REFEN |
| Access | | | | | | | R/W | R/W |
| Reset | | | | | | | 0 | 0 |

**Bit 1 – ADC0REFEN: ADC0 Reference Force Enable**
Writing a '1' to this bit forces the voltage reference for the ADC0 to be running, even if it not requested.

Writing a '0' to this bit allows to automatic enable/disable of the reference source by the peripheral.

**Bit 0 – DAC0REFEN: DAC0 and AC0 Reference Force Enable**
Writing a '1' to this bit forces the voltage reference for the DAC0 and AC0 to be running, even if it not requested.

Writing a '0' to this bit allows to automatic enable/disable of the reference source by the peripheral.

# 19. WDT - Watchdog Timer

## 19.1 Features

- Issues a System Reset if the Watchdog Timer is not cleared before its timeout period
- Operating asynchronously from system clock using an independent oscillator
- Using the 1KHz output of the 32KHz Ultra Low Power oscillator (OSCULP32K)
- 11 selectable timeout periods, from 8ms to 8s
- Two operation modes:
  - Normal Mode
  - Window Mode
- Configuration lock to prevent unwanted changes
- Closed period timer activation after first WDT instruction for easy setup

## 19.2 Overview

The Watchdog Timer (WDT) is a system function for monitoring correct program operation. It allows to recover from situations such as runaway or deadlocked code, by issuing a Reset. When enabled, the WDT is a constantly running timer configured to a predefined timeout period. If the WDT is not reset within the timeout period, it will issue a System Reset. The WDT is reset by executing the `WDR` (Watchdog Timer Reset) instruction from software.

The WDT has two modes of operation, Normal mode and Window mode. The settings in the Control A register (WDT.CTRLA) determine the mode of operation.

A window mode allows to define a time slot or "window" inside the timeout period during which the WDT must be reset. If the WDT is reset outside this window, either too early or too late, a System Reset will be issued. Compared to the normal mode, the window mode can also catch situations where a code error causes constant `WDR` execution.

When enabled, the WDT will run in active mode and all sleep modes. It is asynchronous, i.e. running from a CPU-independent clock source. For this reason it will continue to operate and be able to issue a System Reset even if the main clock fails.

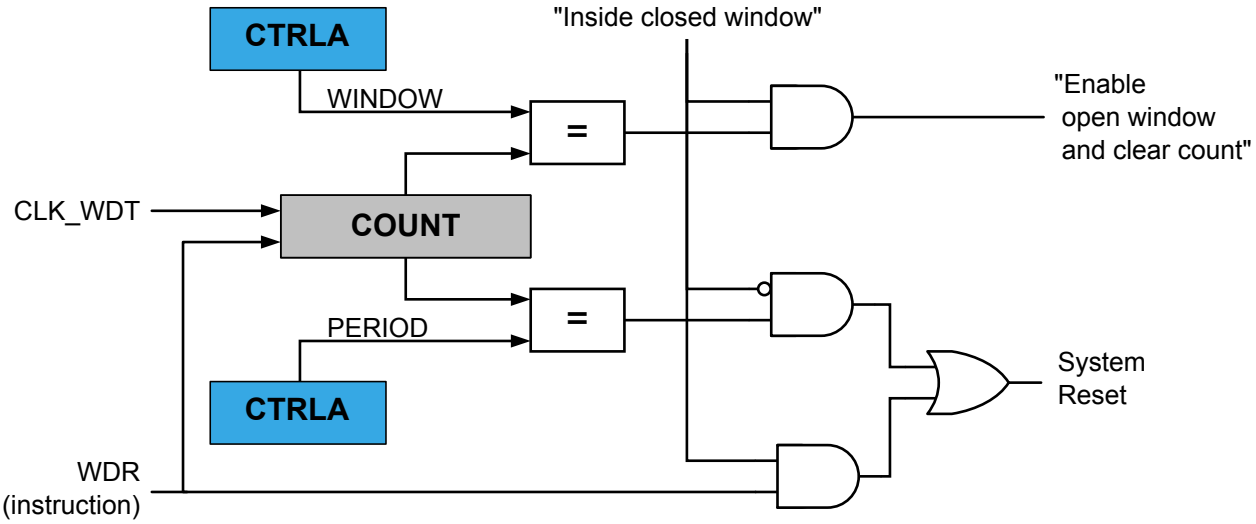
The Configuration Change Protection mechanism ensures that the WDT settings cannot be changed by accident. For increased safety, a configuration for locking the WDT settings is also available.

**Related Links**

CCP - Configuration Change Protection

### 19.2.1 Block Diagram

**Figure 19-1. WDT Block Diagram**




### 19.2.2 Signal Description

Not applicable.

### 19.2.3 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 19-1. WDT System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | No | - |
| Interrupts | No | - |
| Events | No | - |
| Debug | Yes | UPDI |

**Related Links**

Clocks

Debug Operation

#### 19.2.3.1 Clocks

A 1KHz oscillator clock (CLK_WDT_OSC) is sourced from the internal ultra-low-power oscillator, OSCULP32K. Due to the ultra-low-power design, the oscillator is not very accurate, and so the exact time-out period may vary from device to device. This variation must be kept in mind when designing software that uses the WDT to ensure that the time-out periods used are valid for all devices.

The counter clock CLK_WDT_OSC is asynchronous to the system clock. Due to this asynchronicity, writing to WDT control register will require synchronization between the clock domains.

#### 19.2.3.2 I/O Lines and Connections

Not applicable.

#### 19.2.3.3 Interrupts

Not applicable.

#### 19.2.3.4 Events

Not applicable.

#### 19.2.3.5 Debug Operation

When run-time debugging, this peripheral will continue normal operation. Halting the CPU in debugging mode will halt normal operation of the peripheral.

When halting the CPU in debug mode, the WDT counter is reset.

when starting the CPU again and the WDT was operating in window mode, the first closed window timeout period will be disabled, and a normal mode timeout period is executed.

**Related Links**

Window Mode

## 19.3 Functional Description

### 19.3.1 Initialization

- The WDT is enabled when a non-zero value is written to the Period bits (PERIOD) in the Control A register (WDT.CTRLA).
- Optional: Write a non-zero value to the Window bits (WINDOW) in WDT.CTRLA to enable window mode operation.

All bits in the Control A register and the Lock bit (LOCK) in the Status register (WDT.STATUS) are write protected by the Configuration Change Protection mechanism.

The Reset value of WDT.CTRLA is defined by a fuse (FUSE.WDTCFG), so the WDT can be enabled at boot time. If this is the case, the LOCK bit in WDT.STATUS is set at boot time.

**Related Links**

Register Summary - WDT

### 19.3.2 Operation

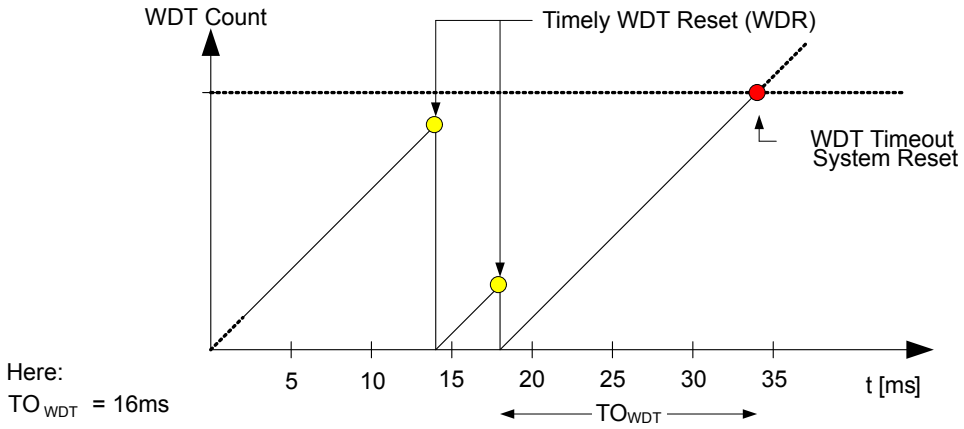
#### 19.3.2.1 Normal Mode

In normal mode operation, a single timeout period is set for the WDT. If the WDT is not reset from software using the `WDR` any time before the timeout occurs, the WDT will issue a System Reset.

A new WDT timeout period will be started each time the WDT is reset by `WDR`.

There are 11 possible WDT timeout periods ($TO_{WDT}$), selectable from 8ms to 8s by writing to the Period bit field (PERIOD) in the Control A register (WDT.CTRLA).

**Figure 19-2. Normal Mode Operation**




Normal mode is enabled as long as the WINDOW bit field in the Control A register (WDT.CTRLA) is 0x0.

**Related Links**

Register Summary - WDT

#### 19.3.2.2 Window Mode

In window mode operation, the WDT uses two different timeout periods, a "closed" window timeout period (TOWDTW) and the normal timeout period (TOWDT):

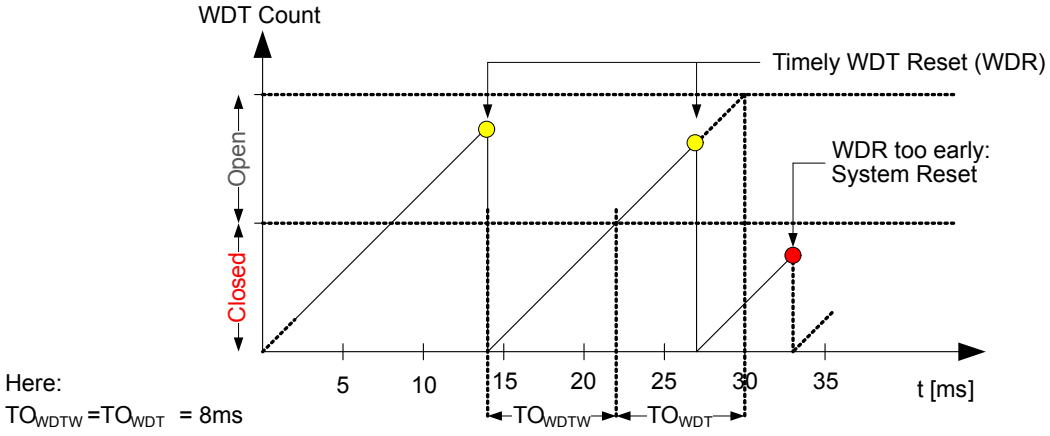
The closed window timeout period defines a duration from 8ms to 8s where the WDT cannot be reset. If the WDT is reset during this period, the WDT will issue a System Reset.

The normal WDT timeout period, which is also 8ms to 8s, defines the duration of the "open" period during which the WDT can (and should) be reset. The open period will always follow the closed period, so the total duration of the timeout period is the sum of the closed window and the open window timeout periods.

When enabling window mode or when going out of debug mode, the first closed period is activated after the first `WDR` instruction.

If a second `WDR` is issued while a previous `WDR` is being synchronized, the second one will be ignored.

**Figure 19-3. Window Mode Operation**




The window mode is enabled by writing a non-zero value to the WINDOW bit field in the Control A register (WDT.CTRLA), and disabled by writing WINDOW=0x0.

#### 19.3.2.3 Configuration Protection and Lock

The WDT provides two security mechanisms to avoid unintentional changes to the WDT settings:

The first mechanism is the Configuration Change Protection mechanism, employing a timed write procedure for changing the WDT control registers.

The second mechanism locks the configuration by writing a '1' to the LOCK bit in the Status register (WDT.STATUS). When this bit is '1', the Control A register (WDT.CTRLA) cannot be changed. Consequently, the WDT cannot be disabled from software.

LOCK in WDT.STATUS can only be written to '1'. It can only be cleared in debug mode.

If the WDT configuration is loaded from fuses, LOCK is automatically set in WDT.STATUS.

**Related Links**

CCP - Configuration Change Protection

### 19.3.3 Events

Not applicable.

### 19.3.4 Interrupts

Not applicable.

### 19.3.5 Sleep Mode Operation

The WDT will continue to operate in any sleep mode where the source clock is active.

### 19.3.6 Synchronization

Due to asynchronicity between the main clock domain and the peripheral clock domain, the Control A register (WDT.CTRLA) is synchronized when written. The Synchronization Busy flag (SYNCBUSY) in the Status register (WDT.STATUS) indicates if there is an ongoing synchronization.

Writing to WDT.CTRLA while SYNCBUSY=1 is not allowed.

The following registers are synchronized when written:

- PERIOD bits in Control A register (WDT.CTRLA)
- Window Period bits (WINDOW) in WDT.CTRLA.

The WDR instruction will need 2 to 3 cycles of the WDT clock in order to be synchronized. Issuing a new WDR instruction while a WDR instruction is being synchronized will be ignored

### 19.3.7 Configuration Change Protection

This peripheral has registers that are under Configuration Change Protection (CCP). In order to write to these, a certain key must be written to the CPU.CCP register first, followed by a write access to the protected bits within four CPU instructions.

It is possible to try writing to these registers any time, but the values are not altered.

The following registers are under CCP:

**Table 19-2. WDT - Registers under Configuration Change Protection**

| Register | Key |
|---|---|
| WDT.CTRLA | IOREG |
| LOCK bit in WDT.STATUS | IOREG |

List of bits/registers protected by CCP.

- Period bits in Control A register (CTRLA.PERIOD)

- Window Period bits in Control A register (CTRLA.WINDOW)
- Lock bit in Status register (STATUS.LOCK)

**Related Links**

CCP - Configuration Change Protection

Sequence for Write Operation to Configuration Change Protected I/O Registers

CCP

## 19.4 Register Summary - WDT

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | WINDOW[3:0] | | | | PERIOD[3:0] | | | |
| 0x01 | STATUS | 7:0 | LOCK | | | | | | | SYNCBUSY |

## 19.5 Register Description

### 19.5.1 Control A

**Name:** CTRLA
**Offset:** 0x00
**Reset:** From FUSE.WDTCFG
**Property:** Configuration Change Protection

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | WINDOW[3:0] | | | | PERIOD[3:0] | | | |
| 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 7:4 – WINDOW[3:0]: Window**
Writing a non-zero value to these bits enables the window mode, and selects the according duration of the closed period.

The bits are optionally lock protected:

- If LOCK bit in WDT.STATUS is '1', all bits are change protected (Access = R)
- If LOCK bit in WDT.STATUS is '0', all bits can be changed (Access = R/W)

| Value | Name | Description |
|---|---|---|
| 0x0 | OFF | - |
| 0x1 | 8CLK | 0.008s |
| 0x2 | 16CLK | 0.016s |
| 0x3 | 32CLK | 0.032s |
| 0x4 | 64CLK | 0.064s |
| 0x5 | 128CLK | 0.128s |
| 0x6 | 256CLK | 0.256s |
| 0x7 | 512CLK | 0.512s |
| 0x8 | 1KCLK | 1.024s |
| 0x9 | 2KCLK | 2.048s |
| 0xA | 4KCLK | 4.096s |
| 0xB | 8KCLK | 8.192s |
| other | - | Reserved |

**Bits 3:0 – PERIOD[3:0]: Period**
Writing a non-zero value to this bit enables the WDT, and selects the according timeout period in normal mode. In window mode, these bits select the duration of the open window.

The bits are optionally lock protected:

- If LOCK in WDT.STATUS is '1', all bits are change protected (Access = R)
- If LOCK in WDT.STATUS is '0', all bits can be changed (Access = R/W)

| Value | Name | Description |
|---|---|---|
| 0x0 | OFF | - |
| 0x1 | 8CLK | 0.008s |
| 0x2 | 16CLK | 0.016s |
| 0x3 | 32CLK | 0.032s |
| 0x4 | 64CLK | 0.064s |
| 0x5 | 128CLK | 0.128s |
| 0x6 | 256CLK | 0.256s |
| 0x7 | 512CLK | 0.512s |
| 0x8 | 1KCLK | 1.0s |
| 0x9 | 2KCLK | 2.0s |
| 0xA | 4KCLK | 4.1s |
| 0xB | 8KCLK | 8.2s |
| other | - | Reserved |

### 19.5.2 Status

**Name:** STATUS
**Offset:** 0x01
**Reset:** 0x00
**Property:** Configuration Change Protection

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

**Bit 7 – LOCK: Lock**
Writing this bit to '1' write protects the WDT.CTRLA register.

It is only possible to write this bit to '1'. This bit can only be cleared in debug.

If the PERIOD bits in WDT.CTRLA are different from zero after boot code, Lock will automatically be set.

This bit is under Configuration Change Protection (CCP).

**Bit 0 – SYNCBUSY: Synchronization Busy**
This bit is set after writing to the WDT.CTRLA register while the data is being synchronized from the system clock domain to the WDT clock domain.

This bit is cleared by system after the synchronization is finished.

This bit is not under Configuration Change Protection (CCP).

**Related Links**

Synchronization

Configuration Change Protection

# 20. TCA - 16-bit Timer/Counter Type A

## 20.1 Features

- 16-bit timer/counter
- Three compare channels
- Double buffered timer period setting
- Double buffered compare channels
- Waveform generation:
  - Frequency generation
  - Single-slope PWM (pulse width modulation)
  - Dual-slope PWM
- Count on event
- Timer overflow interrupts/events
- One compare match per compare channel
- Two 8-bit timer/counters in Split Mode

## 20.2 Overview

The flexible 16-bit PWM Timer/Counter type A (TCA) provides accurate program execution timing, frequency and waveform generation, and command execution.

A TCA consists of a base counter and a set of compare channels. The base counter can be used to count clock cycles or events, or let events control how it counts clock cycles. It has direction control and period setting that can be used for timing. The compare channels can be used together with the base counter to do compare match control, frequency generation, and pulse width waveform modulation.

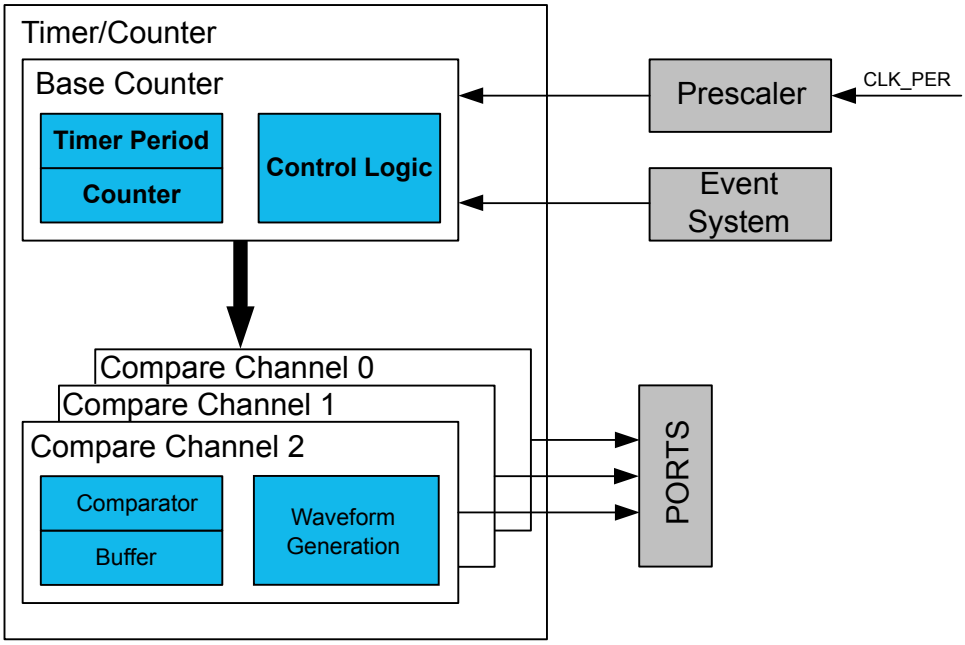
Depending on the mode of operation, the counter is cleared, reloaded, incremented, or decremented at each timer/counter clock or event input.

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

By default, the TCA is a 16-bit timer/counter. The timer/Counter has a Split mode feature that splits it into two 8-bit timer/counters with three compare channels each.

A block diagram of the 16-bit timer/counter with closely related peripheral modules (in grey) is shown below.

**Figure 20-1. 16-bit Timer/counter and Closely Related Peripherals**

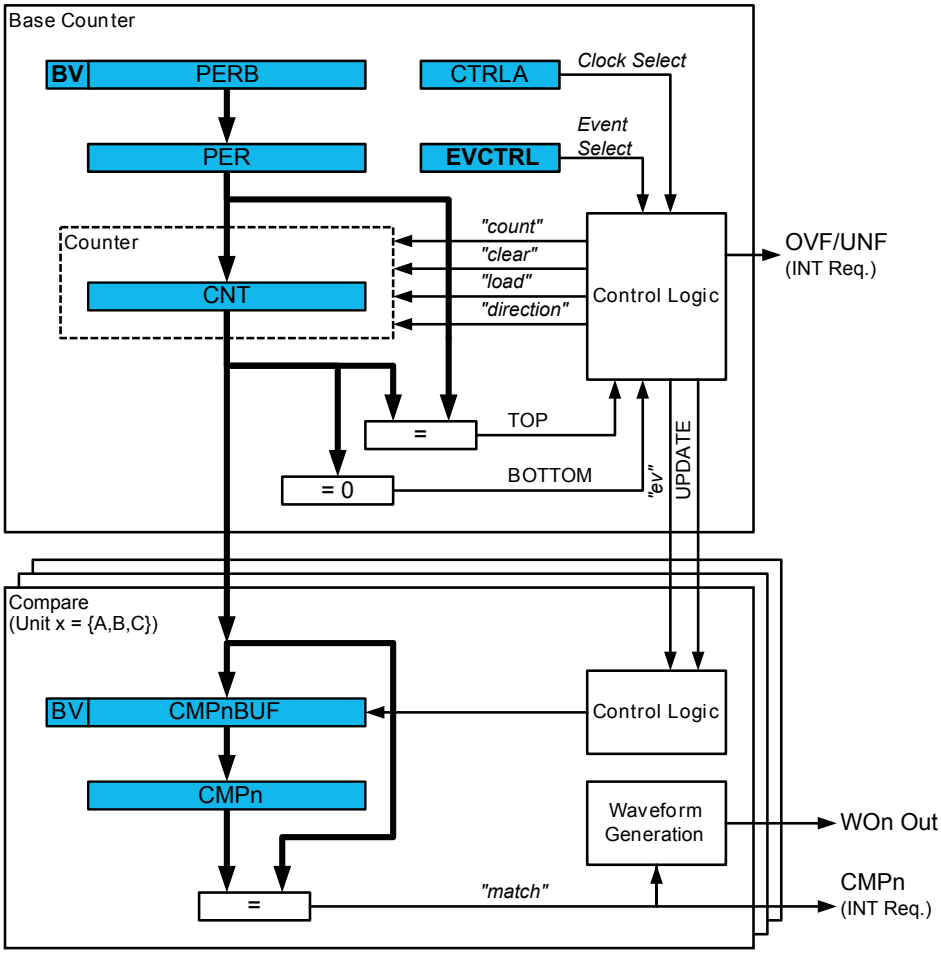



This device provides one instance of the TCA peripheral, TCA0.

### 20.2.1 Block Diagram

The below figure shows a detailed block diagram of the timer/counter.

**Figure 20-2. Timer/Counter Block Diagram**



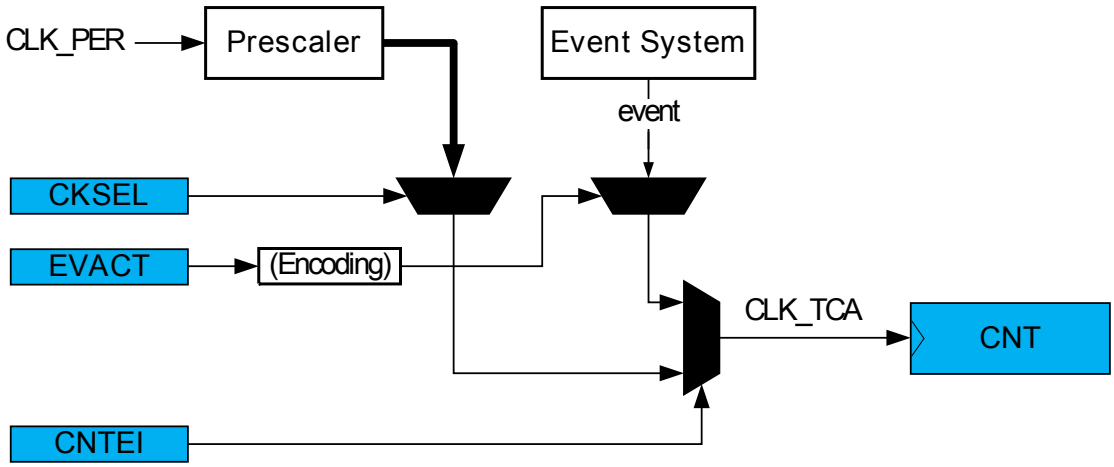

The counter register (CNT), period registers with buffer (PER and PERBUF), and compare registers with buffers (CMPx and CMPBUFx) are 16-bit registers. All buffer registers have a buffer valid (BV) flag that indicates when the buffer contains a new value.

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

The counter value is also compared to the CMPx registers. These comparisons can be used to generate interrupt requests. The waveform generator modes use these comparisons to set the waveform period or pulse width.

A prescaled peripheral clock and events from the event system can be used to control the counter.

**Figure 20-3. Timer/Counter Clock Logic**




### 20.2.2 Signal Description

| Signal | Description | Type |
|---|---|---|
| WO[2:0] | Digital output | Waveform output |
| WO[5:3] | Digital output | Waveform output - Split Mode only |

### 20.2.3 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 20-1. TCA System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | Yes | WO[5:0] |
| Interrupts | Yes | CPUINT |
| Events | Yes | EVSYS |
| Debug | Yes | UPDI |

**Related Links**

Clocks
Debug Operation
Interrupts
Events

#### 20.2.3.1 Clocks

This peripheral uses the system clock CLK_PER, and has its own prescaler.

**Related Links**

CLKCTRL - Clock Controller

#### 20.2.3.2 I/O Lines and Connections

Using the I/O lines of the peripheral requires configuration of the I/O pins.

**Related Links**

I/O Multiplexing and Considerations

PORT - I/O Pin Configuration

#### 20.2.3.3 Interrupts

Using the interrupts of this peripheral requires the Interrupt Controller to be configured first.

**Related Links**

CPUINT - CPU Interrupt Controller

SREG

Interrupts

#### 20.2.3.4 Events

The events of this peripheral are connected to the Event System.

**Related Links**

EVSYS - Event System

#### 20.2.3.5 Debug Operation

When run-time debugging, this peripheral will continue normal operation. Halting the CPU in debugging mode will halt normal operation of the peripheral.

This peripheral can be forced to operate with halted CPU by writing a '1' to the Debug Run bit (DBGRUN) in the Debug Control register of the peripheral (*peripheral*.DBGCTRL).

**Related Links**

UPDI - Unified Program and Debug Interface

## 20.3 Functional Description

### 20.3.1 Definitions

The following definitions are used throughout the documentation:

**Table 20-2. Timer/Counter Definitions**

| Name | Description |
|---|---|
| BOTTOM | The counter reaches BOTTOM when it becomes zero. |
| MAX | The counter reaches MAXimum when it becomes all ones. |
| TOP | The counter reaches TOP when it becomes equal to the highest value in the count sequence. |
| UPDATE | The update condition is met when the timer/counter reaches BOTTOM or TOP, depending on the waveform generator mode. |
| CNT | Counter register value |
| CMP | Compare register value |

In general, the term “timer” is used when the timer/counter is counting periodic clock ticks. The term “counter” is used when the input signal has sporadic or irregular ticks.

### 20.3.2 Initialization

To start using the timer/counter in a basic mode, follow these steps:

- Write a TOP value to the Period register (TCA.PER)
- Enable the peripheral by writing a '1' to the ENABLE bit in the Control A register (TCA.CTRLA). The counter will start counting clock ticks according to the prescaler setting in the Clock Select bit field (CLKSEL) in TCA.CTRLA.
- Optional: By writing a '1' to the Enable Count on Event Input bit (CNTEI) in the Event Control register (TCA.EVCTRL), Event inputs are counted instead of clock ticks.
- The counter value can be read from the Counter bit field (CNT) in the Counter register (TCA.CNT).

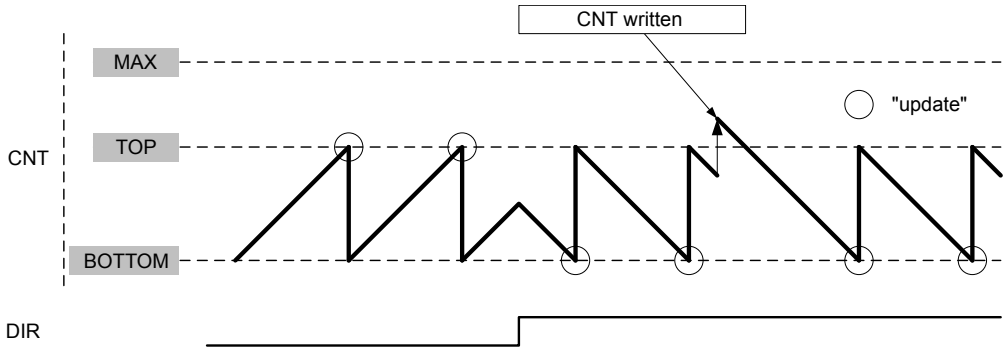
### 20.3.3 Operation

#### 20.3.3.1 Normal Operation

In normal operation, the counter is counting clock ticks in the direction selected by the Direction bit (DIR) in the Control E register (TCA.CTRLE), until it reaches TOP or BOTTOM. The clock ticks are from the peripheral clock CLK_PER, optionally prescaled, depending on the Clock Select bit field (CLKSEL) in the Control A register (TCA.CTRLA).

When up-counting and TOP is reached, the counter will wrap to zero at the next clock tick. When down-counting, the counter is reloaded with the Period register value (TCA.PER) when BOTTOM is reached.

**Figure 20-4. Normal Operation**



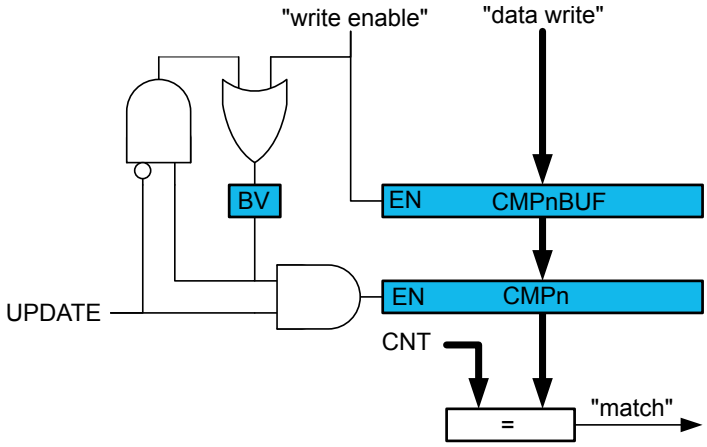

It is possible to change the Counter value in the Counter register (TCA.CNT) when the counter is running. The write access to TCA.CNT has higher priority than count, clear, or reload, and will be immediate. The direction of the counter can also be changed during normal operation by writing to DIR in TCA.CTRLE.

#### 20.3.3.2 Double Buffering

The Period register value (TCA.PER) and the Compare n register values (TCA.CMPn) are all double buffered (TCA.PERBUF and TCA.CMPnBUF).

Each buffer register has a Buffer Valid flag (PERBV, CMPnBV) in the Control F register (TCA.CTRLF), which indicates that the buffer register contains a valid, i.e. new, value that can be copied into the corresponding Period or Compare register. When the Period register and Compare n registers are used for a Compare operation, the BV flag is set when data is written to the buffer register, and cleared on an UPDATE condition. This is shown for a Compare register in below.

**Figure 20-5. Period and Compare Double Buffering**



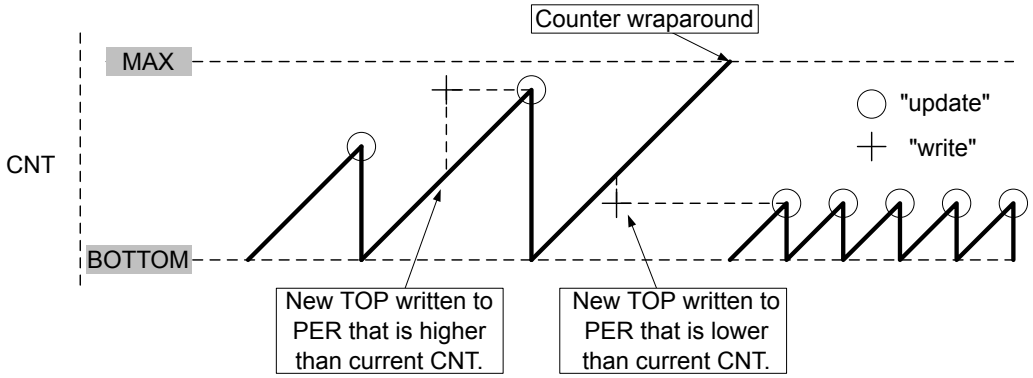

Both the TCA.CMPn and TCA.CMPnBUF registers are available as I/O registers. This allows initialization and bypassing of the buffer register and the double buffering function.

#### 20.3.3.3 Changing the Period

The Counter period is changed by writing a new TOP value to the Period register (TCA.PER).

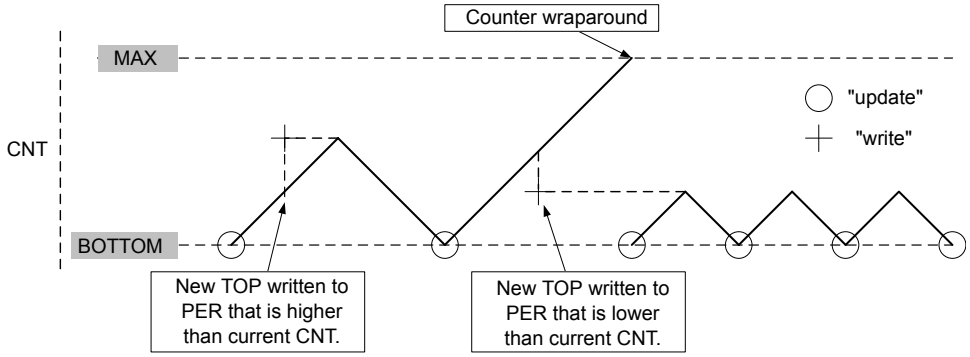
**No Buffering:** If double buffering is not used, any period update is immediate.

**Figure 20-6. Changing the Period without Buffering**



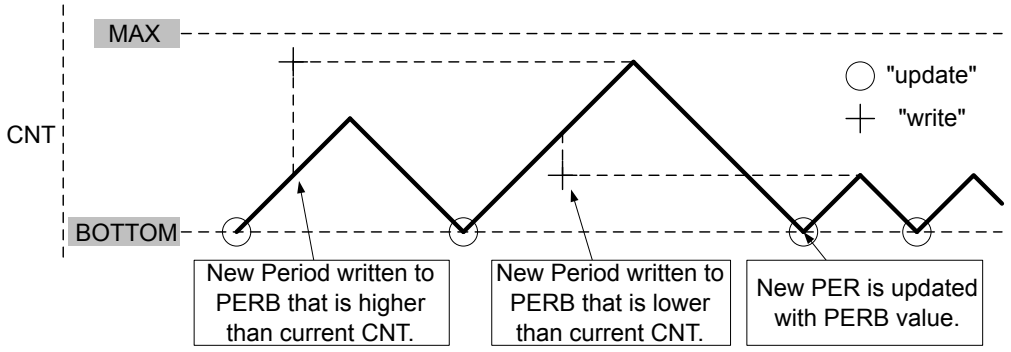

A counter wraparound can occur in any mode of operation when up-counting without buffering. This is due to the fact that the registers TCA.CNT and TCA.PER are continuously compared: if a new TOP value is written to TCA.PER that is lower than current TCA.CNT, the counter will wrap first before a compare match happened.

**Figure 20-7. Unbuffered Dual-slope Operation**



**With Buffering:** When double buffering is used, the buffer can be written at any time and still maintain correct operation. The TCA.PER is always updated on the UPDATE condition, as shown for dual-slope operation in the figure below. This prevents wraparound and the generation of odd waveforms.

**Figure 20-8. Changing the Period Using Buffering**




#### 20.3.3.4 Compare Channel

Each Compare Channel n continuously compares the counter value (TCA.CNT) with the Compare n register (TCA.CMPn). If TCA.CNT equals TCA.CMPn, the comparator n signals a match. The match will set the Compare Channel's interrupt flag at the next timer clock cycle, and the optional interrupt is generated.

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

**Waveform Generation**

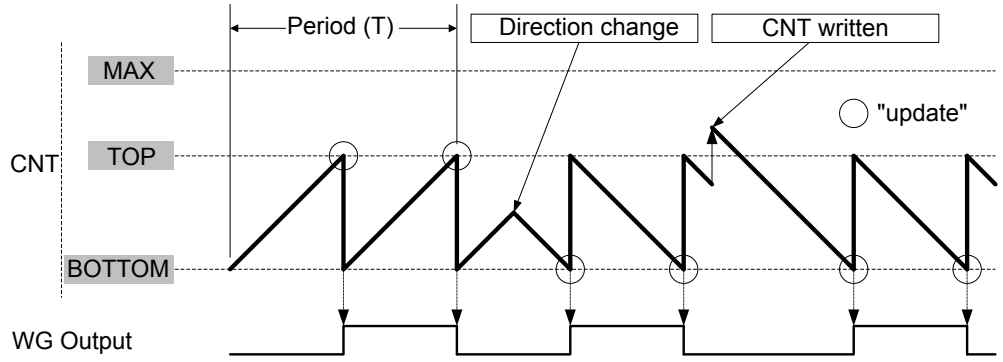
The compare channels can be used for waveform generation on the corresponding port pins. To make the waveform visible on the connected port pin, the following requirements must be fulfilled:

1. A waveform generation mode must be selected by writing the WGMODE bit field in TCA.CTRLB.
2. The TCA is counting clock tick, not Events (CNTEI=0 in TCA.EVCTRL)
3. The compare channels used must be enabled (CMPnEN=1 in TCA.CTRLB). This will override the corresponding PORT pin output register. An alternative pin can be selected by writing to the respective TCA Waveform Output n bit (TCA0n) in the Control C register of the Port Multiplexer (PORTMUX.CTRLC).
4. The direction for the associated Port pin n must be configured as output (PORT.DIR[n]=1).
5. Optional: Enable inverted waveform output for the associated Port pin n (INVEN=1 in PORT.PINn).

**Frequency (FRQ) Waveform Generation**

For frequency generation, the period time (T) is controlled by a TCA.CMPn register instead of the Period register (TCA.PER). The waveform generation output WG is toggled on each compare match between the TCA.CNT and TCA.CMPn registers.

**Figure 20-9. Frequency Waveform Generation**




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

$$f_{\mathrm{FRQ}} = \frac{\mathrm{f_{CLK_PER}}}{2N(\mathrm{CMPn}+1)}$$

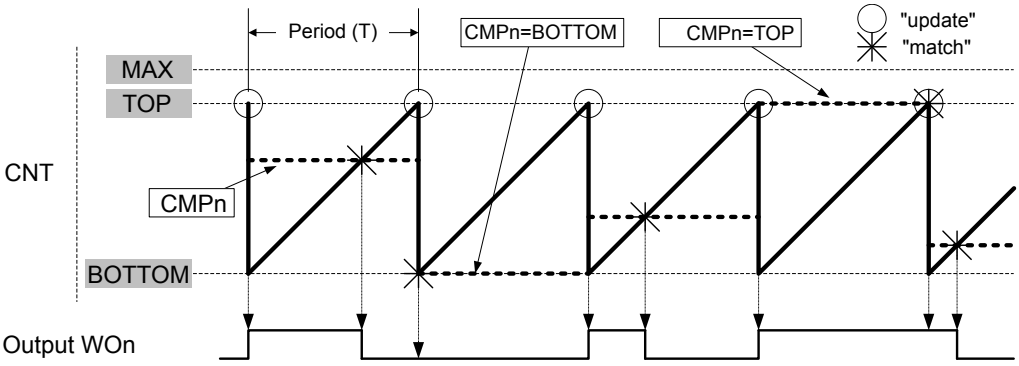
where *N* represents the prescaler divider used (CLKSEL in TCA.CTRLA), and $f_{CLK_PER}$ is the system clock for the peripherals.

The maximum frequency of the waveform generated is half of the peripheral clock frequency ($f_{CLK_PER}/2$) when TCA.CMPn is written to zero (0x0000) and no prescaling is used (*N*=1, CLKSEL=0x0 in TCA.CTRLA).

#### Single-Slope PWM Generation

For single-slope Pulse Width Modulation (PWM) generation, the period (T) is controlled by TCA.PER, while the values of TCA.CMPn control the duty cycle of the WG output. The figure below shows how the counter counts from BOTTOM to TOP and then restarts from BOTTOM. The waveform generator (WO) output is set at TOP, and cleared on the compare match between the TCA.CNT and TCA.CMPn registers.

**Figure 20-10. Single-Slope Pulse Width Modulation**




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

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

$$R_{\mathrm{PWM_SS}} = \frac{\log(\mathrm{PER}+1)}{\log(2)}$$

The single-slope PWM frequency ($f_{PWM_SS}$) depends on the period setting (TCA_PER), the system's peripheral clock frequency $f_{CLK_PER}$, the TCA prescaler (CLKSEL in TCA.CTRLA). It is calculated by the following equation:

$$f_{\mathrm{PWM_SS}} = \frac{f_{\mathrm{CLK_PER}}}{N(\mathrm{PER}+1)}$$

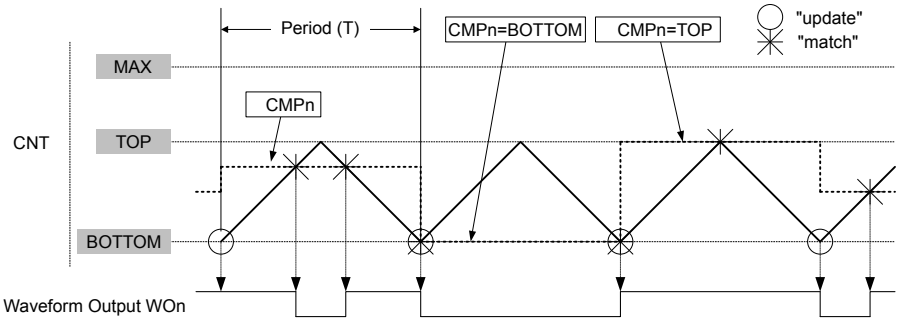
where *N* represents the prescaler divider used.

**Dual-slope PWM**

For dual-slope PWM generation, the period (T) is controlled by TCA.PER, while the values of TCA.CMPn control the duty cycle of the WG output.

The figure below shows how for dual-slope PWM the counter counts repeatedly from BOTTOM to TOP and then from TOP to BOTTOM. The waveform generator output is set on BOTTOM, cleared on compare match when up-counting, and set on compare match when down-counting.

**Figure 20-11. Dual-slope Pulse Width Modulation**




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

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

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

$$R_{\mathrm{PWM_DS}} = \frac{\log(\mathrm{PER}+1)}{\log(2)}$$

The PWM frequency depends on the period setting (TCA.PER), the peripheral clock frequency ($f_{CLK_PER}$), and the prescaler divider used (CLKSEL in TCA.CTRLA). It is calculated by the following equation:

$$f_{\mathrm{PWM_DS}} = \frac{f_{\mathrm{CLK_PER}}}{2N \cdot \mathrm{PER}}$$

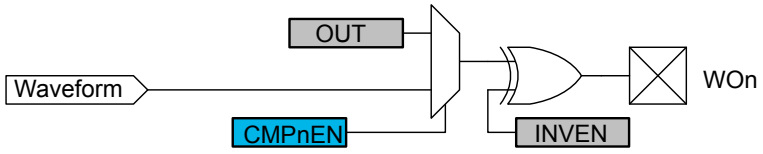
*N* represents the prescaler divider used.

**Port Override for Waveform Generation**

To make the waveform generation available on the Port pins, the corresponding Port pin direction must be set as output (PORT.DIR[n]=1). The TCA will override the port pin values when the compare channel is enabled (CMPnEN=1 in TCA.CTRLB) and a waveform generation mode is selected.

The figure below shows the port override for TCA. The timer/counter compare channel will override the port pin output value (OUT) on the corresponding port pin. Enabling inverted I/O on the port pin (INVEN=1 in PORT.PINn) inverts the corresponding WG output.

**Figure 20-12. Port Override for Timer/Counter Type A**



#### 20.3.3.5 Timer/Counter Commands

A set of commands can be issued by software to immediately change the state of the peripheral. These commands give direct control of the UPDATE, RESTART, and RESET signals. A command is issued by writing the respective value to the Command bit field (CMD) in the Control E register (TCA.CTRLESET).

An `UPDATE` command has the same effect as when an update condition occurs, except that the `UPDATE` command is not affected by the state of the Lock Update bit (LUPD) in the Control E register (TCA.CTRLE).

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

A `RESET` command will set all timer/counter registers to their initial values. A `RESET` can be issued only when the timer/counter is not running (ENABLE=0 in TCA.CTRLA).

#### 20.3.3.6 Split Mode - Two 8-Bit Timer/Counters

**Split Mode Overview**

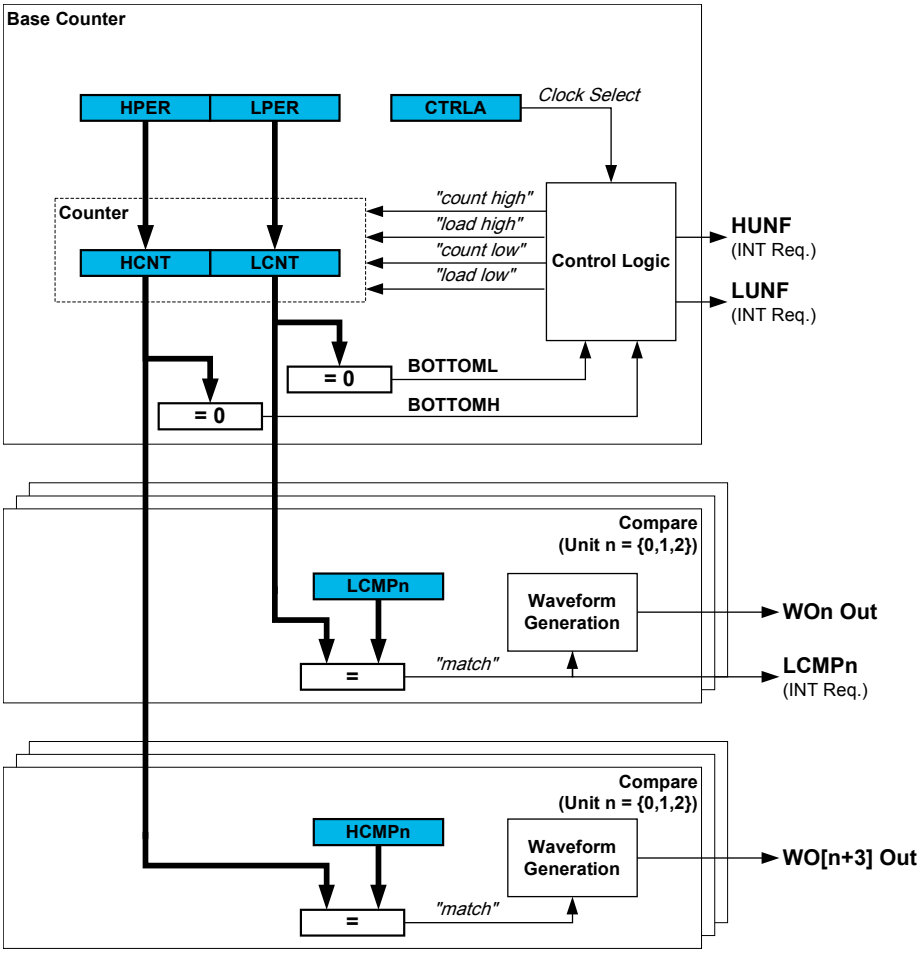
To double the number of timers and PWM channels in the TCA, a Split Mode is provided. In this Split Mode, the 16-bit timer/counter acts as two separate 8-bit timers, which each have three compare channels for PWM generation. The split mode will only work with single slope down count. Split Mode does not support Event action controlled operation.

**Split Mode Differences to Normal Mode**

- Count
  - Down-count only
  - Timer/Counter counter high and low byte are independent (TCA.LCNT, TCA.HCNT)
- Waveform generation
  - Single Slope PWM only (WGMODE=SINGLESLOPE in TCA.CTRLB)
- Interrupt
  - No change for low byte Timer/Counter (TCA.LCNT)
  - Underflow interrupt for high byte Timer/Counter (TCA.HCNT)
  - No compare interrupt or flag for High-byte Compare n registers (TCA.HCMPn)
- Event actions: Not compatible
- Buffer registers and Buffer Valid flags: Unused
- Register access: Byte access to all registers.
- Temp register: Unused, 16-bit register of the Normal Mode are accessed as 8-bit 'TCA_H' and 'TCA_L', respectively.

### Block Diagram

**Figure 20-13. Timer/Counter Block Diagram Split Mode**




### Split Mode Initialization

When shifting between Normal Mode and Split Mode, the functionality of some registers and bits change, but their values do not. For this reason, disabling the peripheral (ENABLE=0 in TCA.CTRLA) and doing a Hard Reset (CMD=RESET in TCA.CTRLESET) is recommended when changing the mode to avoid unexpected behavior.

To start using the timer/counter in basic Split Mode after a Hard Reset, follow these steps:

- Enable Split Mode by writing a '1' to the Split Mode Enable bit in the Control D register (SPLITM in TCA.CTRLD).
- Write a TOP value to the Period registers (TCA.PER)
- Enable the peripheral by writing a '1' to the ENABLE bit in the Control A register (TCA.CTRLA). The counter will start counting clock ticks according to the prescaler setting in the Clock Select bit field (CLKSEL) in TCA.CTRLA.
- The counter values can be read from the Counter bit field in the Counter registers (TCA.CNT).

Activating Split Mode results in changes to the functionality of some registers and register bits. The modifications are described in a separate register map.

### Related Links

Register Summary - TCA in Split Mode (CTRLD.SPLITM=1)

### 20.3.4 Events

The peripheral can take the following actions on an input Event:

- The counter counts positive edges of the Event signal.
- The counter counts both positive and negative edges of the Event signal.
- The counter counts prescaled clock cycles as long as the Event signal is high.
- The counter counts prescaled clock cycles. Event signal controls the direction of counting. Up-count when Event signal is low, down-count when Event signal is high.

The specific action is selected by writing to the Event Action bits (EVACT) in the Event Control register (TCA.EVCTRL). Events as input are enabled by writing a '1' to the Enable Count on Event Input bit (CNTEI in TCA.EVCTRL).

Event controlled operation is not available in Split Mode.

### 20.3.5 Interrupts

**Table 20-3. Available Interrupt Vectors and Sources in Normal Mode**

| Offset | Name | Vector Description | Conditions |
|---|---|---|---|
| 0x00 | OVF | Overflow and Compare match interrupt | The counter has reached its top value and wrapped to zero. |
| 0x04 | CMP0 | Compare channel 0 interrupt | Match between the counter value and the Compare 0 register. |
| 0x06 | CMP1 | Compare channel 1 interrupt | Match between the counter value and the Compare 1 register. |
| 0x08 | CMP2 | Compare channel 2 interrupt | Match between the counter value and the Compare 2 register. |

**Table 20-4. Available Interrupt Vectors and Sources in Split Mode**

| Offset | Name | Vector Description | Conditions |
|---|---|---|---|
| 0x00 | LUNF | Low-byte Underflow interrupt | Low-byte timer reaches BOTTOM. |
| 0x02 | HUNF | High-byte Underflow interrupt | High-byte timer reaches BOTTOM. |
| 0x04 | LCMP0 | Compare channel 0 interrupt | Match between the counter value and the low-byte of Compare 0 register. |
| 0x06 | LCMP1 | Compare channel 1 interrupt | Match between the counter value and the low-byte of Compare 1 register. |
| 0x08 | LCMP2 | Compare channel 2 interrupt | Match between the counter value and the low-byte of the Compare 2 register. |

When an interrupt condition occurs, the corresponding Interrupt Flag is set in the Interrupt Flags register of the peripheral (*peripheral*.INTFLAGS).

An interrupt source is enabled or disabled by writing to the corresponding enable bit in the peripheral's Interrupt Control register (*peripheral*.INTCTRL).

An interrupt request is generated when the corresponding interrupt source is enabled and the Interrupt Flag is set. The interrupt request remains active until the Interrupt Flag is cleared. See the peripheral's INTFLAGS register for details on how to clear Interrupt Flags.

© 2017 Microchip Technology Inc.

**Related Links**

AVR CPU

SREG

### 20.3.6 Sleep Mode Operation

The timer/counter will halt operation in all sleep modes.

### 20.3.7 Configuration Change Protection

Not applicable.

## 20.4 Register Summary - TCA in Normal Mode (CTRLD.SPLITM=0)

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | | | | | CLKSEL[2:0] | | | ENABLE |
| 0x01 | CTRLB | 7:0 | | CMPnEN2 | CMPnEN1 | CMPnEN0 | ALUPD | WGMODE[2:0] | | |
| 0x02 | CTRLC | 7:0 | | | | | | CMP2OV | CMP1OV | CMP0OV |
| 0x03 | CTRLD | 7:0 | | | | | | | | SPLITM |
| 0x04 | CTRLECLR | 7:0 | | | | | CMD[1:0] | | LUPD | DIR |
| 0x05 | CTRLESET | 7:0 | | | | | CMD[1:0] | | LUPD | DIR |
| 0x06 | CTRLFCLR | 7:0 | | | | | CMP2BV | CMP1BV | CMP0BV | PERBV |
| 0x07 | CTRLFSET | 7:0 | | | | | CMP2BV | CMP1BV | CMP0BV | PERBV |
| 0x08 | Reserved | | | | | | | | | |
| 0x09 | EVCTRL | 7:0 | | | | | | EVACT[1:0] | | CNTEI |
| 0x0A | INTCTRL | 7:0 | | CMP2 | CMP1 | CMP0 | | | | OVF |
| 0x0B | INTFLAGS | 7:0 | | CMP2 | CMP1 | CMP0 | | | | OVF |
| 0x0C ... 0x0D | Reserved | | | | | | | | | |
| 0x0E | DBGCTRL | 7:0 | | | | | | | | DBGRUN |
| 0x0F | TEMP | 7:0 | TEMP[7:0] | | | | | | | |
| 0x10 ... 0x1F | Reserved | | | | | | | | | |
| 0x20 | CNT | 7:0 | CNT[7:0] | | | | | | | |
| 0x21 | | 15:8 | CNT[15:8] | | | | | | | |
| 0x22 ... 0x25 | Reserved | | | | | | | | | |
| 0x26 | PER | 7:0 | PER[7:0] | | | | | | | |
| 0x27 | | 15:8 | PER[15:8] | | | | | | | |
| 0x28 | CMP0 | 7:0 | CMP[7:0] | | | | | | | |
| 0x29 | | 15:8 | CMP[15:8] | | | | | | | |
| 0x2A | CMP1 | 7:0 | CMP[7:0] | | | | | | | |
| 0x2B | | 15:8 | CMP[15:8] | | | | | | | |
| 0x2C | CMP2 | 7:0 | CMP[7:0] | | | | | | | |
| 0x2D | | 15:8 | CMP[15:8] | | | | | | | |
| 0x2E ... 0x35 | Reserved | | | | | | | | | |
| 0x36 | PERBUF | 7:0 | PERBUF[7:0] | | | | | | | |
| 0x37 | | 15:8 | PERBUF[15:8] | | | | | | | |
| 0x38 | CMP0BUF | 7:0 | CMPBUF[7:0] | | | | | | | |
| 0x39 | | 15:8 | CMPBUF[15:8] | | | | | | | |
| 0x3A | CMP1BUF | 7:0 | CMPBUF[7:0] | | | | | | | |
| 0x3B | | 15:8 | CMPBUF[15:8] | | | | | | | |
| 0x3C | CMP2BUF | 7:0 | CMPBUF[7:0] | | | | | | | |
| 0x3D | | 15:8 | CMPBUF[15:8] | | | | | | | |

## 20.5 Register Description - Normal Mode

### 20.5.1 Control A

**Name:** CTRLA
**Offset:** 0x00
**Reset:** 0x00
**Property:** -

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

**Bits 3:1 – CLKSEL[2:0]: Clock Select**
These bits select the clock frequency for the timer/counter.

| Value | Name | Description |
|---|---|---|
| 0x0 | DIV1 | $f_{TCA} = f_{CLK_PER}/1$ |
| 0x1 | DIV2 | $f_{TCA} = f_{CLK_PER}/2$ |
| 0x2 | DIV4 | $f_{TCA} = f_{CLK_PER}/4$ |
| 0x3 | DIV8 | $f_{TCA} = f_{CLK_PER}/8$ |
| 0x4 | DIV16 | $f_{TCA} = f_{CLK_PER}/16$ |
| 0x5 | DIV64 | $f_{TCA} = f_{CLK_PER}/64$ |
| 0x6 | DIV256 | $f_{TCA} = f_{CLK_PER}/256$ |
| 0x7 | DIV1024 | $f_{TCA} = f_{CLK_PER}/1024$ |

**Bit 0 – ENABLE: Enable**

| Value | Description |
|---|---|
| 0 | The peripheral is disabled |
| 1 | The peripheral is enabled |

### 20.5.2 Control B - Normal Mode

**Name:** CTRLB
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | CMPnEN2 | CMPnEN1 | CMPnEN0 | ALUPD | WGMODE[2:0] | | |
| Access | | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 4, 5, 6 – CMPnEN: Compare n Enable**
In the FRQ or PWM waveform generation mode, these bits will override the PORT output register for the corresponding pin.

| Value | Description |
|---|---|
| 0 | PORT output settings for the pin with WOn output respected. |
| 1 | PORT output settings for pin with WOn output overridden in FRQ or PWM waveform generation mode. |

**Bit 3 – ALUPD: Auto Lock Update**
The Auto Lock Update feature controls the Lock Update (LUPD) bit in the TCA.CTRLE register. When ALUPD is written to '1', LUPD will be set to '1' until the Buffer Valid (CMPnBV) bits of all enabled compare channels are '1'. This condition will clear LUPD.

It will remain cleared until the next UPDATE condition, where the buffer values will be transferred to the CMPn registers and LUPD will be set to '1' again. This makes sure that CMPnBUF register values are not transferred to the CMPn registers until all enabled compare buffers are written.

| Value | Description |
|---|---|
| 0 | LUPD in TCA.CTRLE not altered by system. |
| 1 | LUPD in TCA.CTRLE set and cleared automatically. |

**Bits 2:0 – WGMODE[2:0]: Waveform Generation Mode**
These bits select the waveform generation mode, and control the counting sequence of the counter, TOP value, UPDATE condition, interrupt condition, and type of waveform that is generated.

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

**Table 20-5. Timer Waveform Generation Mode**

| WGMODE[2:0] | Group Configuration | Mode of Operation | Top | Update | OVF |
|---|---|---|---|---|---|
| 000 | NORMAL | Normal | PER | TOP | TOP |
| 001 | FRQ | Frequency | CMP0 | TOP | TOP |
| 010 | - | Reserved | - | - | - |
| 011 | SINGLESLOPE | Single-slope PWM | PER | BOTTOM | BOTTOM |
| 100 | - | Reserved | - | - | - |
| 101 | DSTOP | Dual-slope PWM | PER | BOTTOM | TOP |
| 110 | DSBOTH | Dual-slope PWM | PER | BOTTOM | TOP and BOTTOM |
| 111 | DSBOTTOM | Dual-slope PWM | PER | BOTTOM | BOTTOM |

| Value | Name | Description |
|---|---|---|
| 0x0 | NORMAL | Normal operation mode |
| 0x1 | FRQ | Frequency mode |
| 0x3 | SINGLESLOPE | Single-slope PWM mode |
| 0x5 | DSTOP | Dual-slope PWM mode |
| 0x6 | DSBOTH | Dual-slope PWM mode |
| 0x7 | DSBOTTOM | Dual-slope PWM mode |
| Other | - | Reserved |

### 20.5.3 Control C - Normal Mode

**Name:** CTRLC
**Offset:** 0x02
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | CMP2OV | CMP1OV | CMP0OV |
| Access | | | | | | R/W | R/W | R/W |
| Reset | | | | | | 0 | 0 | 0 |

**Bit 2 – CMP2OV: Compare Output Value 2**
See CMP0OV.

**Bit 1 – CMP1OV: Compare Output Value 1**
See CMP0OV.

**Bit 0 – CMP0OV: Compare Output Value 0**
The CMPnOV bits allow direct access to the waveform generator's output compare value when the timer/counter is not enabled. This is used to set or clear the WG output value when the timer/counter is not running.

### 20.5.4 Control D

**Name:** CTRLD
**Offset:** 0x03
**Reset:** 0x00
**Property:** -

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

**Bit 0 – SPLITM: Enable Split Mode**
This bit sets the timer/counter in split mode operation. It will then work as two 8-bit timer/counters. The register map will change compared to normal 16-bit mode.

### 20.5.5 Control Register E Clear - Normal Mode

The individual status bit can be cleared by writing a one to its bit location. This allows each bit to be cleared without use of a read-modify-write operation on a single register.
Each Status bit can be read out either by reading TCA.CTRLESET or TCA.CTRLECLR.

**Name:** CTRLECLR
**Offset:** 0x04
**Reset:** 0x00
**Property:** -

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

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

| Value | Name | Description |
|---|---|---|
| 0x0 | NONE | No command |
| 0x1 | UPDATE | Force update |
| 0x2 | RESTART | Force restart |
| 0x3 | RESET | Force hard Reset (ignored if TC is enabled) |

**Bit 1 – LUPD: Lock Update**
Lock update can be used to ensure that all buffers are valid before an update is performed.

| Value | Description |
|---|---|
| 0 | The buffered registers are updated as soon as an UPDATE condition has occurred. |
| 1 | No update of the buffered registers is performed, even though an UPDATE condition has occurred. |

**Bit 0 – DIR: Counter Direction**
Normally this bit is controlled in hardware by the waveform generation mode or by event actions, but this bit can also be changed from software.

| Value | Description |
|---|---|
| 0 | The counter is counting up (incrementing). |
| 1 | The counter is counting down (decrementing). |

### 20.5.6 Control Register E Set - Normal Mode

The individual status bit can be set by writing a '1' to its bit location. This allows each bit to be set without use of a read-modify-write operation on a single register.
Each Status bit can be read out either by reading TCA.CTRLESET or TCA.CTRLECLR.

**Name:** CTRLESET
**Offset:** 0x05
**Reset:** 0x00
**Property:** -

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

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

| Value | Name | Description |
|---|---|---|
| 0x0 | NONE | No command |
| 0x1 | UPDATE | Force update |
| 0x2 | RESTART | Force restart |
| 0x3 | RESET | Force hard Reset (ignored if TC is enabled) |

**Bit 1 – LUPD: Lock Update**
Locking the update ensures that all buffers are valid before an update is performed.

| Value | Description |
|---|---|
| 0 | The buffered registers are updated as soon as an UPDATE condition has occurred. |
| 1 | No update of the buffered registers is performed, even though an UPDATE condition has occurred. |

**Bit 0 – DIR: Counter Direction**
Normally this bit is controlled in hardware by the waveform generation mode or by event actions, but this bit can also be changed from software.

| Value | Description |
|---|---|
| 0 | The counter is counting up (incrementing). |
| 1 | The counter is counting down (decrementing). |

### 20.5.7 Control Register F Clear

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

**Name:** CTRLFCLR
**Offset:** 0x06
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | CMP2BV | CMP1BV | CMP0BV | PERBV |
| Access | | | | | R/W | R/W | R/W | R/W |
| Reset | | | | | 0 | 0 | 0 | 0 |

**Bit 3 – CMP2BV: Compare 2 Buffer Valid**
See CMP0BV.

**Bit 2 – CMP1BV: Compare 1 Buffer Valid**
See CMP0BV.

**Bit 1 – CMP0BV: Compare 0 Buffer Valid**
The CMPnBV bits are set when a new value is written to the corresponding CMPnBUF register. These bits are automatically cleared on an UPDATE condition.

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

### 20.5.8 Control Register F Set

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

**Name:** CTRLFSET
**Offset:** 0x07
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | CMP2BV | CMP1BV | CMP0BV | PERBV |
| Access | | | | | R/W | R/W | R/W | R/W |
| Reset | | | | | 0 | 0 | 0 | 0 |

**Bit 3 – CMP2BV: Compare 2 Buffer Valid**
See CMP0BV.

**Bit 2 – CMP1BV: Compare 1 Buffer Valid**
See CMP0BV.

**Bit 1 – CMP0BV: Compare 0 Buffer Valid**
The CMPnBV bits are set when a new value is written to the corresponding CMPnBUF register. These bits are automatically cleared on an UPDATE condition.

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

### 20.5.9 Event Control

**Name:** EVCTRL
**Offset:** 0x09
**Reset:** 0x00
**Property:** -

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

**Bits 2:1 – EVACT[1:0]: Event Action**
These bits define on what type of event action the counter will increment or decrement.

| Value | Name | Description |
|---|---|---|
| 0x0 | EVACT_POSEDGE | Count on positive edge event |
| 0x1 | EVACT_ANYEDGE | Count on any edge event |
| 0x2 | EVACT_HIGHLVL | Count on prescaled clock while event line is 1. |
| 0x3 | EVACT_UPDOWN | Count on prescaled clock. Event controls count direction. Up-count when event line is 0, down-count when event line is 1. |

**Bit 0 – CNTEI: Enable Count on Event Input**

| Value | Description |
|---|---|
| 0 | Counting on Event input is disabled. |
| 1 | Counting on Event input is enabled according to EVACT bit field. |

### 20.5.10 Interrupt Control Register - Normal Mode

**Name:** INTCTRL
**Offset:** 0x0A
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | CMP2 | CMP1 | CMP0 | | | | OVF |
| Access | | R/W | R/W | R/W | | | | R/W |
| Reset | | 0 | 0 | 0 | | | | 0 |

**Bit 6 – CMP2: Compare Channel 2 Interrupt Enable**
See CMP0.

**Bit 5 – CMP1: Compare Channel 1 Interrupt Enable**
See CMP0.

**Bit 4 – CMP0: Compare Channel 0 Interrupt Enable**
Writing CMPn bit to '1' enables compare interrupt from channel n.

**Bit 0 – OVF: Timer Overflow/Underflow Interrupt Enable**
Writing OVF bit to '1' enables overflow interrupt.

### 20.5.11 Interrupt Flag Register - Normal Mode

The individual status bit can be cleared by writing a '1'e to its bit location. This allows each bit to be set without use of a read-modify-write operation on a single register.

**Name:** INTFLAGS
**Offset:** 0x0B
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | CMP2 | CMP1 | CMP0 | | | | OVF |
| Access | | R/W | R/W | R/W | | | | R/W |
| Reset | | 0 | 0 | 0 | | | | 0 |

**Bit 6 – CMP2: Compare Channel 2 Interrupt Flag**
See CMP0 flag description.

**Bit 5 – CMP1: Compare Channel 1 Interrupt Flag**
See CMP0 flag description.

**Bit 4 – CMP0: Compare Channel 0 Interrupt Flag**
The compare interrupt flag (CMPn) is set on a compare match on the corresponding compare channel.

For all modes of operation, the CMPn flag will be set when a compare match occurs between the count register (CNT) and the corresponding compare register (CMPn). The CMPn flag will not be cleared automatically and has to be cleared by software. This is done by writing a one to its bit location.

**Bit 0 – OVF: Overflow/Underflow Interrupt Flag**
This flag is set either on a TOP (overflow) or BOTTOM (underflow) condition, depending on the WGMODE setting. OVF is not automatically cleared and needs to be cleared by software. This is done by writing a one to its bit location.

### 20.5.12 Debug Control Register

**Name:** DBGCTRL
**Offset:** 0x0E
**Reset:** 0x00
**Property:** -

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

**Bit 0 – DBGRUN: Run in Debug**

| Value | Description |
|---|---|
| 0 | The peripheral is halted in break debug mode and ignores events. |
| 1 | The peripheral will continue to run in break debug mode when the CPU is halted. |

### 20.5.13 Temporary bits for 16-bit Access

The Temporary register is used by the CPU for single-cycle, 16-bit access to the 16-bit registers of this peripheral. It can also be read and written by software. See also Accessing 16-bit Registers. There is one common Temporary register for all the 16-bit registers of this peripheral.

**Name:** TEMP
**Offset:** 0x0F
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TEMP[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 – TEMP[7:0]: Temporary Bits for 16-bit Access**

### 20.5.14 Counter Register - Normal Mode

The TCA.CNTL and TCA.CNTH register pair represents the 16-bit value, TCA.CNT. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

CPU and UPDI write access has priority over internal updates of the register.

**Name:** CNT
**Offset:** 0x20
**Reset:** 0x00
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | CNT[15:8] | | | | | | | |
| 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 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CNT[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 15:8 – CNT[15:8]: Counter high byte**
These bits hold the MSB of the 16-bit counter register.

**Bits 7:0 – CNT[7:0]: Counter low byte**
These bits hold the LSB of the 16-bit counter register.

### 20.5.15 Period Register - Normal Mode

TCA.PER contains the 16-bit TOP value in the timer/counter.

The TCA.PERL and TCA.PERH register pair represents the 16-bit value, TCA.PER. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

**Name:** PER
**Offset:** 0x26
**Reset:** 0xFFFF
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | PER[15:8] | | | | | | | |
| 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 |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | PER[7:0] | | | | | | | |
| 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 15:8 – PER[15:8]: Periodic high byte**
These bits hold the MSB of the 16-bit period register.

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

### 20.5.16 Compare n Register - Normal Mode

This register is continuously compared to the counter value. Normally, the outputs from the comparators are then used for generating waveforms.

TCA.CMPn registers are updated with the buffer value from their corresponding CMPnBUF register when an UPDATE condition occurs.

The TCA.CMPnL and TCA.CMPnH register pair represents the 16-bit value, TCA.CMPn. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

**Name:** CMP0, CMP1, CMP2
**Offset:** 0x28 + n*0x02 [n=0..2]
**Reset:** 0x00
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | CMP[15:8] | | | | | | | |
| 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 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CMP[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 15:8 – CMP[15:8]: Compare high byte**
These bits hold the MSB of the 16-bit compare register.

**Bits 7:0 – CMP[7:0]: Compare low byte**
These bits hold the LSB of the 16-bit compare register.

### 20.5.17 Period Buffer Register

This register serves as the buffer for the period register (TCA.PER). Accessing this register using the CPU or UPDI will affect the PERBV flag.

The TCA.PERBUFL and TCA.PERBUFH register pair represents the 16-bit value, TCA.PERBUF. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

**Name:** PERBUF
**Offset:** 0x36
**Reset:** 0xFFFF
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | PERBUF[15:8] | | | | | | | |
| 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 |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | PERBUF[7:0] | | | | | | | |
| 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 15:8 – PERBUF[15:8]: Period Buffer high byte**
These bits hold the MSB of the 16-bit period buffer register.

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

### 20.5.18 Compare n Buffer Register

This register serves as the buffer for the associated compare registers (TCA.CMPn). Accessing any of these registers using the CPU or UPDI will affect the corresponding CMPnBV status bit.

The TCA.CMPnBUFL and TCA.CMPnBUFH register pair represents the 16-bit value, TCA.CMPnBUF. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

**Name:** CMP0BUF, CMP1BUF, CMP2BUF
**Offset:** 0x38 + n*0x02 [n=0..2]
**Reset:** 0x00
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | CMPBUF[15:8] | | | | | | | |
| 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 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CMPBUF[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 15:8 – CMPBUF[15:8]: Compare high byte**
These bits hold the MSB of the 16-bit compare buffer register.

**Bits 7:0 – CMPBUF[7:0]: Compare low byte**
These bits hold the LSB of the 16-bit compare buffer register.

## 20.6 Register Summary - TCA in Split Mode (CTRLD.SPLITM=1)

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | | | | | CLKSEL[2:0] | | | ENABLE |
| 0x01 | CTRLB | 7:0 | | HCMP2EN | HCMP1EN | HCMP0EN | | LCMP2EN | LCMP1EN | LCMP0EN |
| 0x02 | CTRLC | 7:0 | | HCMP2OV | HCMP1OV | HCMP0OV | | LCMP2OV | LCMP1OV | LCMP0OV |
| 0x03 | CTRLD | 7:0 | | | | | | | | SPLITM |
| 0x04 | CTRLECLR | 7:0 | | | | | CMD[1:0] | | | |
| 0x05 | CTRLESET | 7:0 | | | | | CMD[1:0] | | | |
| 0x06 ... 0x09 | Reserved | | | | | | | | | |
| 0x0A | INTCTRL | 7:0 | | LCMP2 | LCMP1 | LCMP0 | | | HUNF | LUNF |
| 0x0B | INTFLAGS | 7:0 | | LCMP2 | LCMP1 | LCMP0 | | | HUNF | LUNF |
| 0x0C ... 0x0D | Reserved | | | | | | | | | |
| 0x0E | DBGCTRL | 7:0 | | | | | | | | DBGRUN |
| 0x0F ... 0x1F | Reserved | | | | | | | | | |
| 0x20 | LCNT | 7:0 | LCNT[7:0] | | | | | | | |
| 0x21 | HCNT | 7:0 | HCNT[7:0] | | | | | | | |
| 0x22 ... 0x25 | Reserved | | | | | | | | | |
| 0x26 | LPER | 7:0 | LPER[7:0] | | | | | | | |
| 0x27 | HPER | 7:0 | HPER[7:0] | | | | | | | |
| 0x28 | LCMP0 | 7:0 | LCMP[7:0] | | | | | | | |
| 0x29 | HCMP0 | 7:0 | HCMP[7:0] | | | | | | | |
| 0x2A | LCMP1 | 7:0 | LCMP[7:0] | | | | | | | |
| 0x2B | HCMP1 | 7:0 | HCMP[7:0] | | | | | | | |
| 0x2C | LCMP2 | 7:0 | LCMP[7:0] | | | | | | | |
| 0x2D | HCMP2 | 7:0 | HCMP[7:0] | | | | | | | |

## 20.7 Register Description - Split Mode

### 20.7.1 Control A

**Name:** CTRLA
**Offset:** 0x00
**Reset:** 0x00
**Property:** -

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

**Bits 3:1 – CLKSEL[2:0]: Clock Select**
These bits select the clock frequency for the timer/counter.

| Value | Name | Description |
|---|---|---|
| 0x0 | DIV1 | $f_{TCA} = f_{CLK_PER}/1$ |
| 0x1 | DIV2 | $f_{TCA} = f_{CLK_PER}/2$ |
| 0x2 | DIV4 | $f_{TCA} = f_{CLK_PER}/4$ |
| 0x3 | DIV8 | $f_{TCA} = f_{CLK_PER}/8$ |
| 0x4 | DIV16 | $f_{TCA} = f_{CLK_PER}/16$ |
| 0x5 | DIV64 | $f_{TCA} = f_{CLK_PER}/64$ |
| 0x6 | DIV256 | $f_{TCA} = f_{CLK_PER}/256$ |
| 0x7 | DIV1024 | $f_{TCA} = f_{CLK_PER}/1024$ |

**Bit 0 – ENABLE: Enable**

| Value | Description |
|---|---|
| 0 | The peripheral is disabled |
| 1 | The peripheral is enabled |

### 20.7.2 Control B - Split Mode

**Name:** CTRLB
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | HCMP2EN | HCMP1EN | HCMP0EN | | LCMP2EN | LCMP1EN | LCMP0EN |
| Access | | R/W | R/W | R/W | | R/W | R/W | R/W |
| Reset | | 0 | 0 | 0 | | 0 | 0 | 0 |

**Bit 6 – HCMP2EN: High-byte Compare 2 Enable**
See LCMP0EN.

**Bit 5 – HCMP1EN: High-byte Compare 1 Enable**
See LCMP0EN.

**Bit 4 – HCMP0EN: High-byte Compare 0 Enable**
See LCMP0EN.

**Bit 2 – LCMP2EN: Low-byte Compare 2 Enable**
See LCMP0EN.

**Bit 1 – LCMP1EN: Low-byte Compare 1 Enable**
See LCMP0EN.

**Bit 0 – LCMP0EN: Low-byte Compare 0 Enable**
Setting the LCMPnEN/HCMPnEN bits in the FRQ or PWM waveform generation mode of operation will override the port output register for the corresponding WOn pin.

### 20.7.3 Control C - Split Mode

**Name:** CTRLC
**Offset:** 0x02
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | HCMP2OV | HCMP1OV | HCMP0OV | | LCMP2OV | LCMP1OV | LCMP0OV |
| Access | | R/W | R/W | R/W | | R/W | R/W | R/W |
| Reset | | 0 | 0 | 0 | | 0 | 0 | 0 |

**Bit 6 – HCMP2OV: High-byte Compare 2 Output Value**
See LCMP0OV.

**Bit 5 – HCMP1OV: High-byte Compare 1 Output Value**
See LCMP0OV.

**Bit 4 – HCMP0OV: High-byte Compare 0 Output Value**
See LCMP0OV.

**Bit 2 – LCMP2OV: Low-byte Compare 2 Output Value**
See LCMP0OV.

**Bit 1 – LCMP1OV: Low-byte Compare 1 Output Value**
See LCMP0OV.

**Bit 0 – LCMP0OV: Low-byte Compare 0 Output Value**
The LCMPnOV/HCMPn bits allow direct access to the waveform generator's output compare value when the timer/counter is not enabled. This is used to set or clear the WOn output value when the timer/counter is not running.

### 20.7.4 Control D

**Name:** CTRLD
**Offset:** 0x03
**Reset:** 0x00
**Property:** -

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

**Bit 0 – SPLITM: Enable Split Mode**
This bit sets the timer/counter in split mode operation. It will then work as two 8-bit timer/counters. The register map will change compared to normal 16-bit mode.

### 20.7.5 Control Register E Clear - Split Mode

The individual status bit can be cleared by writing a '1' to its bit location. This allows each bit to be cleared without use of a read-modify-write operation on a single register.
Each Status bit can be read out either by reading TCA.CTRLESET or TCA.CTRLECLR.

**Name:** CTRLECLR
**Offset:** 0x04
**Reset:** 0x00
**Property:** -

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

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

| Value | Name | Description |
|---|---|---|
| 0x0 | NONE | No command |
| 0x1 | UPDATE | Force update |
| 0x2 | RESTART | Force restart |
| 0x3 | RESET | Force hard Reset (ignored if TC is enabled) |

### 20.7.6 Control Register E Set - Split Mode

The individual status bit can be set by writing a '1' to its bit location. This allows each bit to be set without use of a read-modify-write operation on a single register.
Each Status bit can be read out either by reading TCA.CTRLESET or TCA.CTRLECLR.

**Name:** CTRLESET
**Offset:** 0x05
**Reset:** 0x00
**Property:** -

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

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

| Value | Name | Description |
|---|---|---|
| 0x0 | NONE | No command |
| 0x1 | UPDATE | Force update |
| 0x2 | RESTART | Force restart |
| 0x3 | RESET | Force hard Reset (ignored if TC is enabled) |

### 20.7.7 Interrupt Control Register - Split Mode

**Name:** INTCTRL
**Offset:** 0x0A
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | LCMP2 | LCMP1 | LCMP0 | | | HUNF | LUNF |
| Access | | R/W | R/W | R/W | | | R/W | R/W |
| Reset | | 0 | 0 | 0 | | | 0 | 0 |

**Bit 6 – LCMP2: Low-byte Compare Channel 0 Interrupt Enable**
See LCMP0.

**Bit 5 – LCMP1: Low-byte Compare Channel 1 Interrupt Enable**
See LCMP0.

**Bit 4 – LCMP0: Low-byte Compare Channel 0 Interrupt Enable**
Writing LCMPn bit to '1' enables low-byte compare interrupt from channel n.

**Bit 1 – HUNF: High-byte Underflow Interrupt Enable**
Writing HUNF bit to '1' enables high-byte underflow interrupt.

**Bit 0 – LUNF: Low-byte Underflow Interrupt Enable**
Writing HUNF bit to '1' enables low-byte underflow interrupt.

### 20.7.8 Interrupt Flag Register - Split Mode

The individual status bit can be cleared by writing a ‘1’ to its bit location. This allows each bit to be set without use of a read-modify-write operation on a single register.

**Name:** INTFLAGS
**Offset:** 0x0B
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | LCMP2 | LCMP1 | LCMP0 | | | HUNF | LUNF |
| Access | | R/W | R/W | R/W | | | R/W | R/W |
| Reset | | 0 | 0 | 0 | | | 0 | 0 |

**Bit 6 – LCMP2: Low-byte Compare Channel 0 Interrupt Flag**
See LCMP0 flag description.

**Bit 5 – LCMP1: Low-byte Compare Channel 0 Interrupt Flag**
See LCMP0 flag description.

**Bit 4 – LCMP0: Low-byte Compare Channel 0 Interrupt Flag**
The compare interrupt flag (LCMPn) is set on a compare match on the corresponding compare channel.

For all modes of operation, the LCMPn flag will be set when a compare match occurs between the Low-byte count register (LCNT) and the corresponding compare register (LCMPn). The LCMPn flag will not be cleared automatically and has to be cleared by software. This is done by writing a '1' to its bit location.

**Bit 1 – HUNF: High-byte Underflow Interrupt Flag**
This flag is set on a high-byte timer BOTTOM (underflow) condition. HUNF is not automatically cleared and needs to be cleared by software. This is done by writing a '1' to its bit location.

**Bit 0 – LUNF: Low-byte Underflow Interrupt Flag**
This flag is set on a low-byte timer BOTTOM (underflow) condition. LUNF is not automatically cleared and needs to be cleared by software. This is done by writing a '1' to its bit location.

### 20.7.9 Debug Control Register

**Name:** DBGCTRL
**Offset:** 0x0E
**Reset:** 0x00
**Property:** -

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

**Bit 0 – DBGRUN: Run in Debug**

| Value | Description |
|---|---|
| 0 | The peripheral is halted in break debug mode and ignores events. |
| 1 | The peripheral will continue to run in break debug mode when the CPU is halted. |

### 20.7.10 Low-byte Timer Counter Register - Split Mode

TCA.LCNT contains the counter value in low-byte timer. CPU and UPDI write access has priority over count, clear, or reload of the counter.

**Name:** LCNT
**Offset:** 0x20
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | LCNT[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 – LCNT[7:0]: Counter value for low-byte timer**
These bits define the counter value of the low-byte timer.

### 20.7.11 High-byte Timer Counter Register - Split Mode

TCA.HCNT contains the counter value in high-byte timer. CPU and UPDI write access has priority over count, clear, or reload of the counter.

**Name:** HCNT
**Offset:** 0x21
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | HCNT[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 – HCNT[7:0]: Counter value for high-byte timer**
These bits define the counter value in high-byte timer.

### 20.7.12 Low-byte Timer Period Register - Split Mode

The TCA.LPER register contains the TOP value of low-byte timer.

**Name:** LPER
**Offset:** 0x26
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | LPER[7:0] | | | | | | | |
| 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 7:0 – LPER[7:0]: Period value low-byte timer**
These bits hold the TOP value of low-byte timer.

### 20.7.13 High-byte Period Register - Split Mode

The TCA.HPER register contains the TOP value of high-byte timer.

**Name:** HPER
**Offset:** 0x27
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | HPER[7:0] | | | | | | | |
| 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 7:0 – HPER[7:0]: Period value high-byte timer**
These bits hold the TOP value of high-byte timer.

### 20.7.14 Compare Register n for low-byte Timer - Split Mode

The TCA.LCMPn register represents the compare value of compare channel n for low-byte Timer. This register is continuously compared to the counter value of low-byte timer, LCNT. Normally, the outputs from the comparators are then used for generating waveforms.

**Name:** LCMP
**Offset:** 0x28 + n*0x02 [n=0..2]
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | LCMP[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 – LCMP[7:0]: Compare value of channel n**
These bits hold the compare value of channel n that is compared to LCNT.

### 20.7.15 High-byte Compare Register n - Split Mode

The TCA.HCMPn register represents the compare value of compare channel n for high-byte Timer. This register is continuously compared to the counter value of high-byte timer, HCNT. Normally, the outputs from the comparators are then used for generating waveforms.

**Name:** HCMP0, HCMP1, HCMP2
**Offset:** 0x29 + n*0x02 [n=0..2]
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | HCMP[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 – HCMP[7:0]: Compare value of channel n**
These bits hold the compare value of channel n that is compared to HCNT.

# 21. TCB - 16-bit Timer/Counter Type B

## 21.1 Features

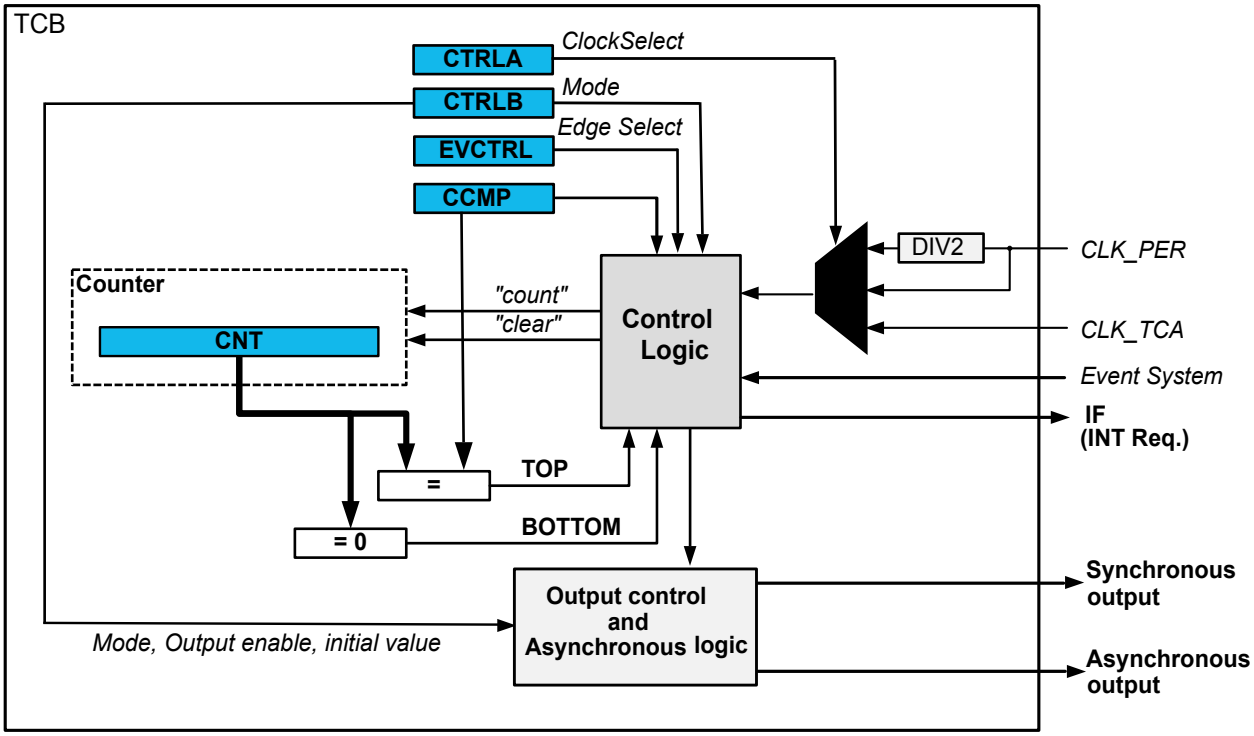
- 16-bit counter operation modes:
  - Periodic interrupt
  - Timeout check
  - Input capture
    - On event
    - Frequency measurement
    - Pulse width measurement
    - Frequency and pulse width measurement
  - Single shot
  - 8-bit Pulse Width Modulation (PWM)
- Noise canceler on event input
- Optional: Operation synchronous with TCA0

## 21.2 Overview

The capabilities of the 16-bit Timer/Counter type B (TCB) include frequency and waveform generation, and input capture on Event with time and frequency measurement of digital signals. The TCB consists of a base counter and control logic which can be set in one of eight different modes, each mode providing unique functionality. The base counter is clocked by the peripheral clock with optional prescaling.

### 21.2.1 Block Diagram

**Figure 21-1. Timer/Counter Type B Block Diagram**




#### 21.2.1.1 Noise Canceler

The noise canceler improves noise immunity by using a simple digital filter scheme. When the noise filter is enabled the peripheral monitors the event channel and keeps a record of the last four observed samples. If four consecutive samples are equal, the input is considered to be stable and the signal is fed to the edge detector.

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 input compare register.

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

### 21.2.2 Signal Description

| Signal | Description | Type |
|---|---|---|
| WO | Digital asynchronous output | Waveform Output |

**Related Links**

I/O Multiplexing and Considerations

### 21.2.3 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 21-1. TCB System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | Yes | WO |

| Dependency | Applicable | Peripheral |
|---|---|---|
| Interrupts | Yes | CPUINT |
| Events | Yes | EVSYS |
| Debug | Yes | UPDI |

**Related Links**

Clocks

Debug Operation

Interrupts

Events

#### 21.2.3.1 Clocks

This peripheral uses the system's peripheral clock CLK_PER. The peripheral has its own local prescaler, or can be configured to run off the prescaled clock signal of the Timer Counter type A (TCA).

**Related Links**

CLKCTRL - Clock Controller

#### 21.2.3.2 I/O Lines and Connections

Using the I/O lines of the peripheral requires configuration of the I/O pins.

**Related Links**

I/O Multiplexing and Considerations

PORT - I/O Pin Configuration

#### 21.2.3.3 Interrupts

Using the interrupts of this peripheral requires the Interrupt Controller to be configured first.

**Related Links**

CPUINT - CPU Interrupt Controller

SREG

Interrupts

#### 21.2.3.4 Events

The events of this peripheral are connected to the Event System.

**Related Links**

EVSYS - Event System

#### 21.2.3.5 Debug Operation

When the CPU is halted in debug mode, this peripheral will halt normal operation. This peripheral can be forced to continue operation during debugging.

This peripheral can be forced to operate with halted CPU by writing a '1' to the Debug Run bit (DBGRUN) in the Debug Control register of the peripheral (*peripheral*.DBGCTRL).

**Related Links**

UPDI - Unified Program and Debug Interface

## 21.3 Functional Description

### 21.3.1 Definitions

The following definitions are used throughout the documentation:

**Table 21-2. Timer/Counter Definitions**

| Name | Description |
|---|---|
| BOTTOM | The counter reaches BOTTOM when it becomes zero. |
| MAX | The counter reaches MAXimum when it becomes all ones. |
| TOP | The counter reaches TOP when it becomes equal to the highest value in the count sequence. |
| UPDATE | The update condition is met when the timer/counter reaches BOTTOM or TOP, depending on the waveform generator mode. |
| CNT | Counter register value |
| CCMP | Capture/Compare register value |

In general, the term “timer” is used when the timer/counter is counting periodic clock ticks. The term “counter” is used when the input signal has sporadic or irregular ticks.

### 21.3.2 Initialization

By default the TCB is in Periodic Interrupt mode. Follow these steps to start using it:

- Write a TOP value to the Compare/Capture register (TCB.CCMP).
- Enable the counter by writing a '1' to the ENABLE bit in the Control A register (TCB.CTRLA). The counter will start counting clock ticks according to the prescaler setting in the Clock Select bit field (CLKSEL in TCB.CTRLA).
- The counter value can be read from the Count register (TCB.CNT). The peripheral will generate an interrupt when the CNT value reaches TOP.

### 21.3.3 Operation

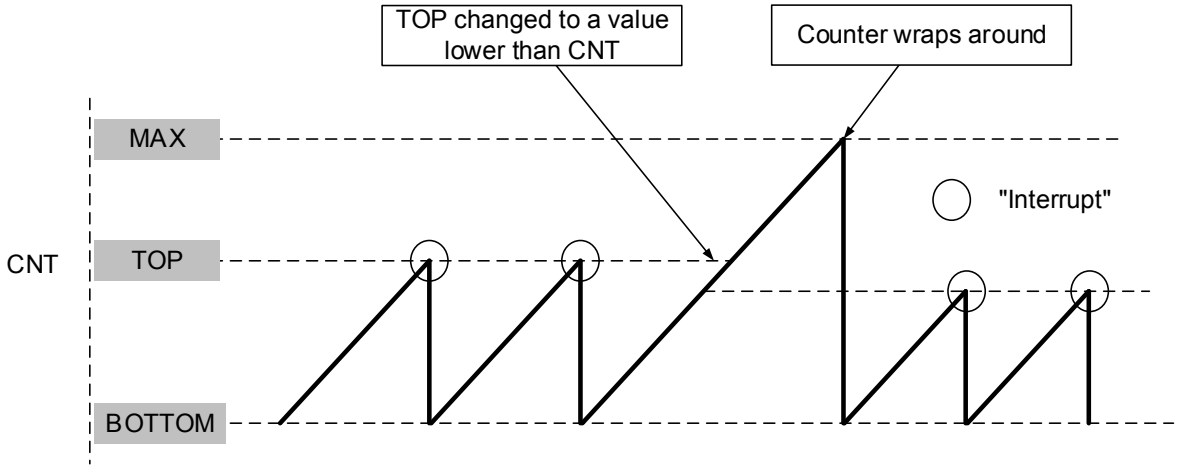
#### 21.3.3.1 Modes

The timer can be configured to run in one of the eight different modes listed below. The event pulse needs to be longer than one system clock cycle in order to guarantee edge detection.

**Periodic Interrupt Mode**

In the periodic interrupt mode the counter counts to the capture value and restarts from zero. Interrupt is generated when counter is equal to TOP. If TOP is updated to a value lower than count, the counter will continue until MAX and wrap around without generating an interrupt.

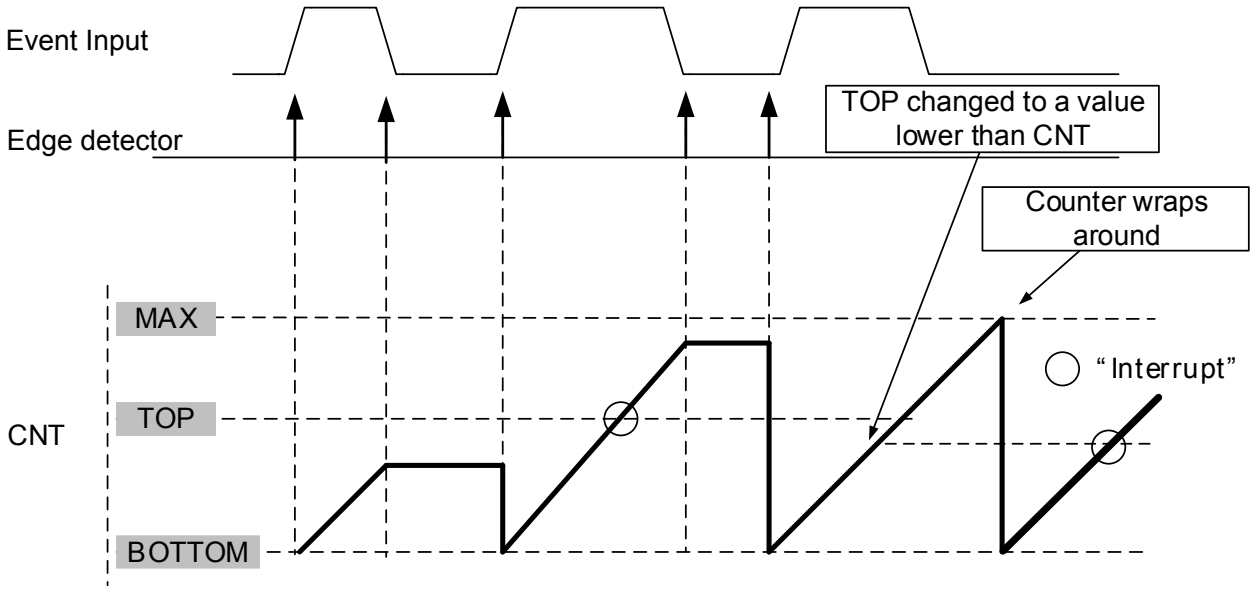
**Figure 21-2. Periodic Interrupt Mode**




**Timeout Check Mode**

In this mode, the counter counts to MAX and wraps around. On the first edge the counter is restarted and on the second edge the counter is stopped. If the count register (TCB.CNT) reaches TOP before the second edge, an interrupt will be generated. In freeze state, the counter will restart on a new edge. Reading count (TCB.CNT) or compare/capture (TCB.CCMP) register, or writing run bit (RUN in TCB.STATUS) in freeze state will have no effect.

**Figure 21-3. Timeout Check Mode**

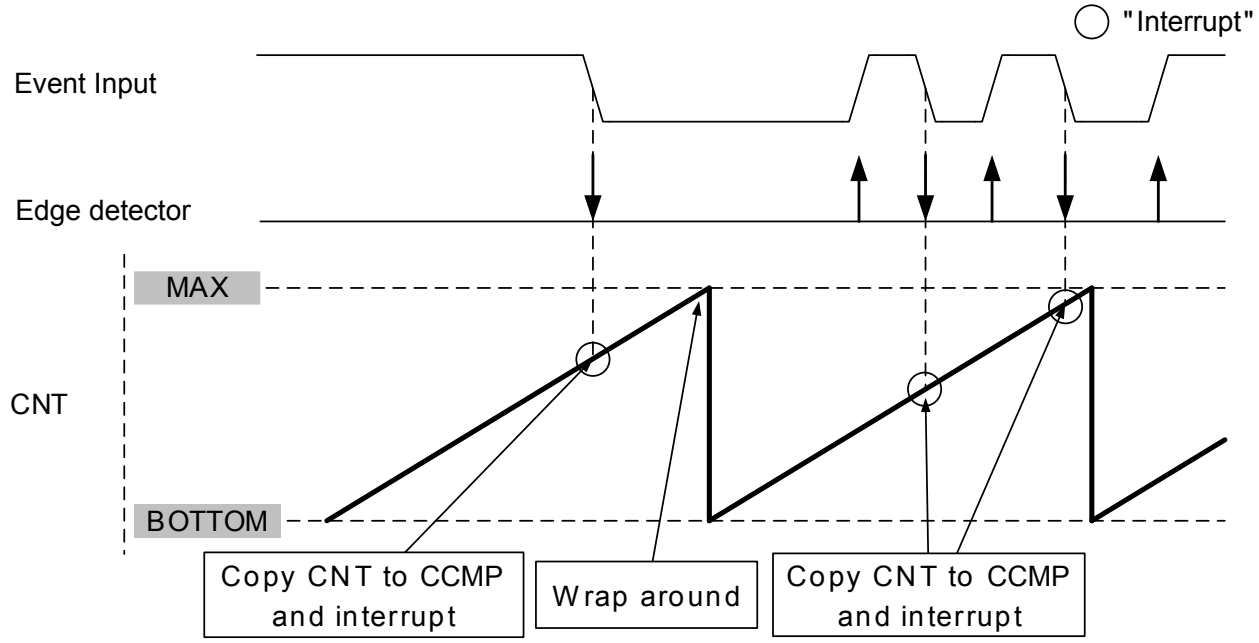



**Input Capture on Event Mode**

The counter will count from BOTTOM to MAX continuously. When an event is detected the counter value will be transferred to the Compare/Capture register (TCB.CCMP) and interrupt is generated. The module has an edge detector which can be configured to trigger count capture on either rising or falling edges.

The figure below shows the input capture unit configured to capture on falling edge on the event input signal. The interrupt flag is automatically cleared after the high byte of the capture register has been read.

**Figure 21-4. Input Capture on Event**




It is recommended to write zero to the TCB.CNT register when entering this mode from any other mode.

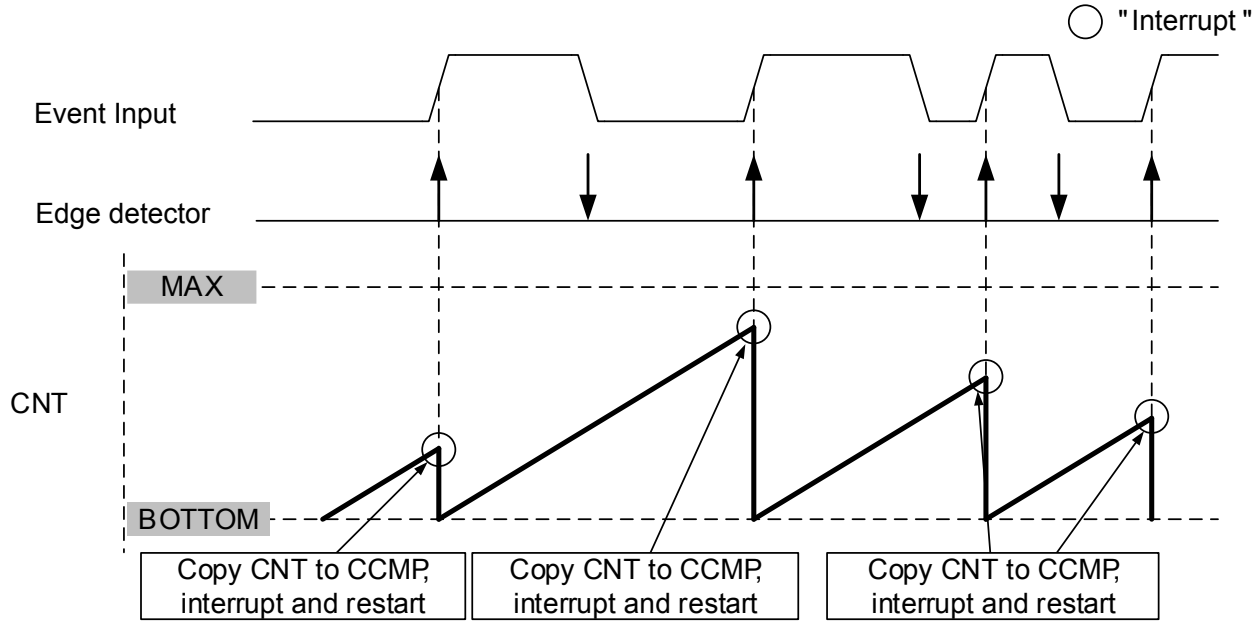
**Input Capture Frequency Measurement Mode**

In this mode, the TCB captures the counter value and restarts on either a positive or negative edge of the event input signal.

The interrupt flag is automatically cleared after the high byte of the Compare/Capture register (TCB.CCMP) has been read, and an interrupt request is generated.

The figure below illustrates this mode when configured to act on rising edge.

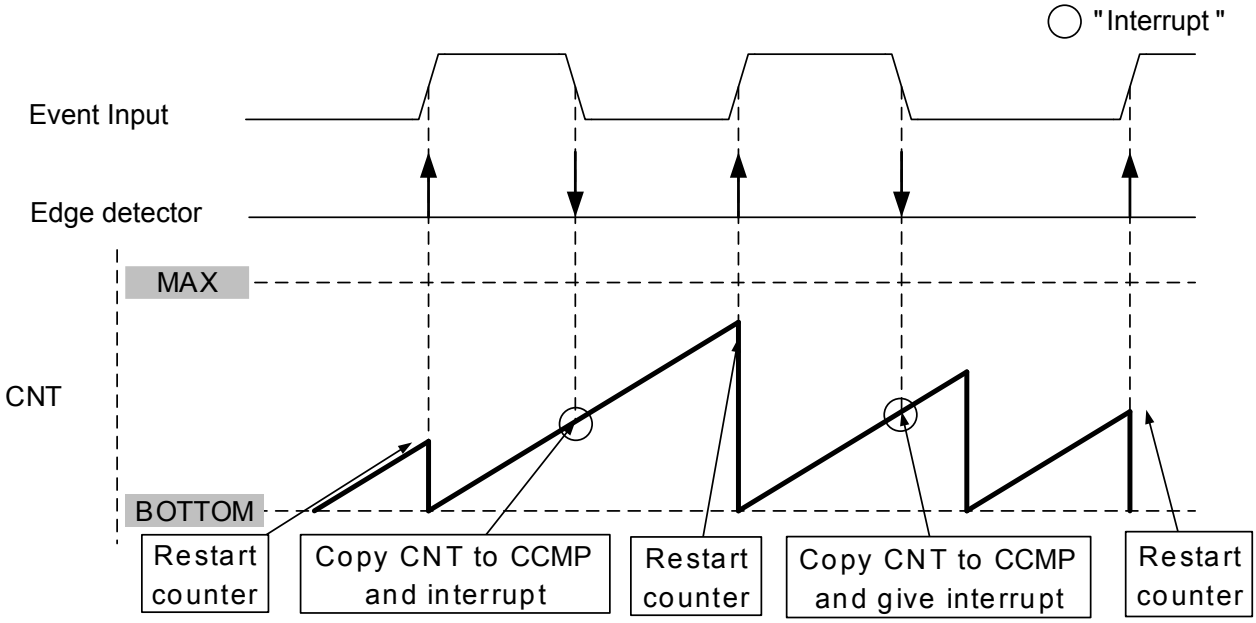
**Figure 21-5. Input Capture Frequency Measurement**



**Input Capture Pulse Width Measurement Mode**

The input capture pulse width measurement will restart the counter on a positive edge and capture on the next falling edge before an interrupt request is generated. The interrupt flag is automatically cleared when the high byte of the capture register is read. The timer will automatically switch between rising and falling edge detection, but a minimum edge separation of two clock cycles is required for correct behavior.

**Figure 21-6. Input Capture Pulse Width Measurement**

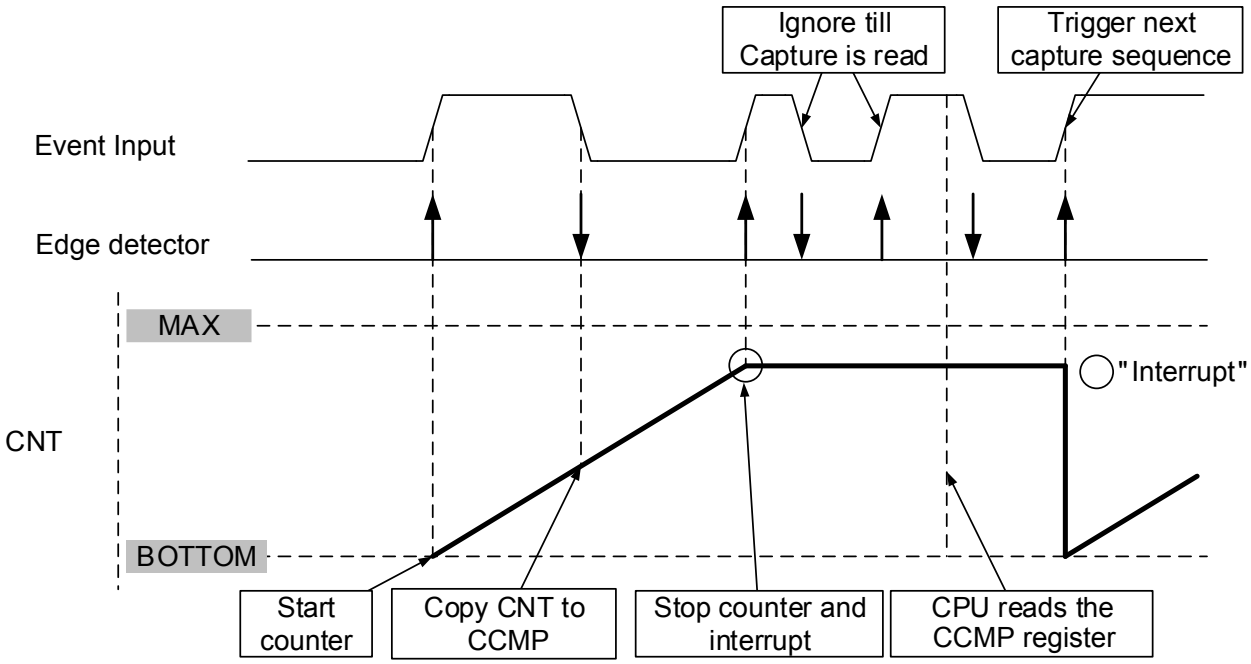



**Input Capture Frequency and Pulse Width Measurement Mode**

In this mode, the timer will start counting when a positive edge is detected on the even input signal. On the following falling edge, the count value is captured. The counter stops when the second rising edge of the event input signal is detected. This will also set the interrupt flag.

Reading the capture will clear the interrupt flag. When the capture register is read or the interrupt flag is cleared the TC is ready for a new capture sequence. The counter register should therefor be read before the capture register as this is reset to zero at the next positive edge.

**Figure 21-7. Input Capture Frequency and Pulse Width Measurement**




**Single Shot Mode**

This mode can be used to generate a pulse with a duration that is defined by the Compare register (TCB.CCMP), every time a rising or falling edge is observed on a connected event channel.

When the counter is stopped, the output pin is driven to low. If an event is detected on the connected event channel, the timer will reset and start counting from zero to TOP while driving its output high. The RUN bit in the Status register can be read to see if the counter is counting or not. When the counter register reaches the CCMP register value, counter will stop and the output pin will go low for at least one prescaler cycle. If a new event arrives during this time, that event will be ignored. The following figure shows an example waveform. There is a two clock cycle delay from when the event is received until the output is set high. If the ASYNC bit in TCB.CTRLB is written to '1', an asynchronous edge detector is used for input events to give immediate action. When the EDGE bit of the TCB.EVCTRL register is written to '1', any edge can trigger the start of counter. If the EDGE bit is '0', only positive edges will trigger the start.

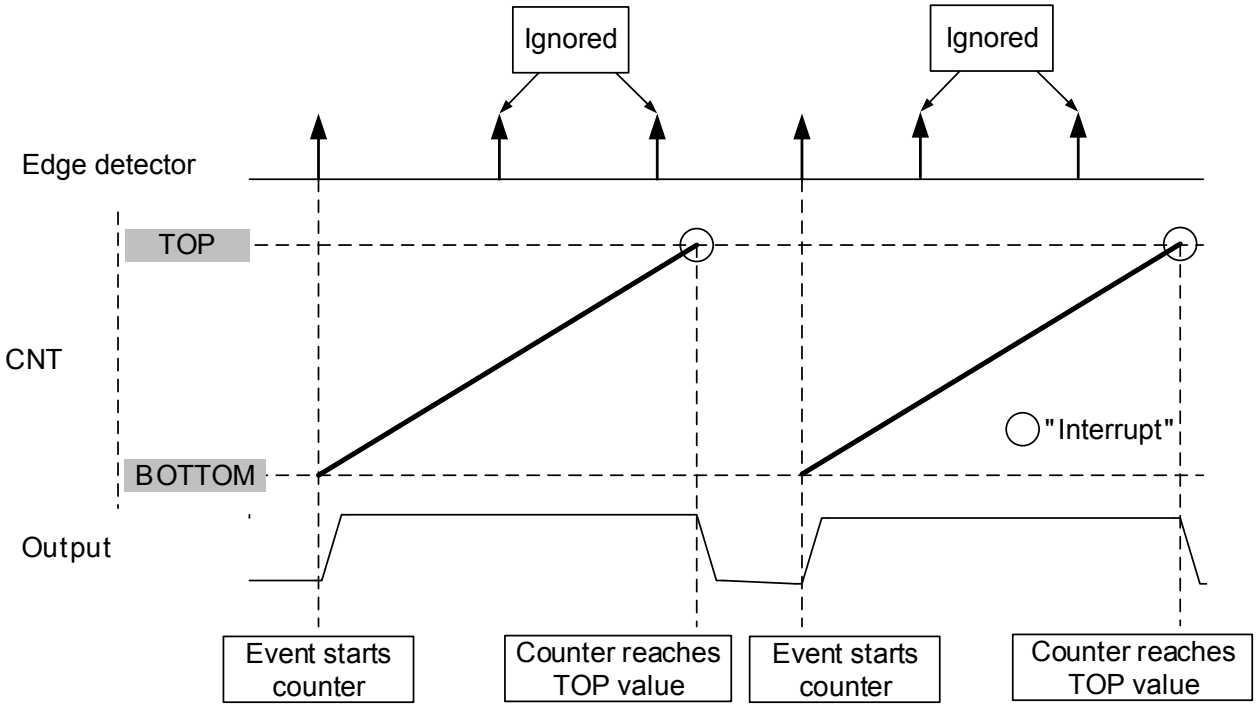
The counter will start as soon as the module is enabled, even without triggering event. This is prevented by writing TOP to the counter register.

Similar behavior is seen if the EDGE bit in the TCB.EVCTRL register is '1' while the module is enabled. Writing TOP to the Counter register prevents this as well.

It is not recommended to change configuration while the module is enabled.

© 2017 Microchip Technology Inc.

**Figure 21-8. Single-Shot Mode**




If the ASYNC bit in TCB.CTRLB is '0', the event pulse needs to be longer than one system clock cycle in order to guarantee edge detection.

#### 8-bit PWM Mode

This timer can also be configured to run in 8-bit PWM mode where each of the register pairs in the 16-bit compare/capture register (TCB.CCMPH and TCB.CCMPL) are used as individual compare registers. The counter will continuously count from zero to CCMPL and the output will be set at BOTTOM and cleared when the counter reaches CCMPH.

When this peripheral is enabled and in PWM mode, changing the value of the compare/capture register will change the output, but the transition may output invalid values. It is hence recommended to

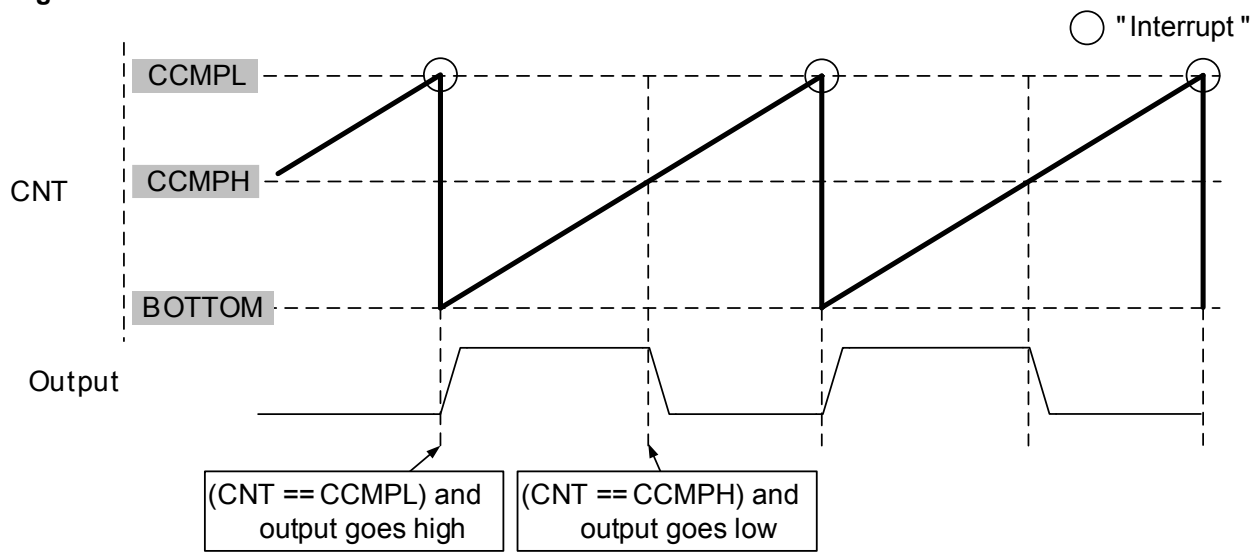
1. Disable the peripheral
2. Write compare/capture register to {CCMPH, CCMPL}
3. Write 0x0000 to count register
4. Re-enable the module.

CCMPH is the number of cycles for which the output will be driven high, CCMPL+1 is the period of the output pulse.

Output of the module for different capture register values are explained below.

- CCMPL = 0 ⇒ Output = 0
- CCMPL = 0xFF
  - CCMPH = 0 ⇒ Output = 0
  - 0 < CCMPH ≤ 0xFF ⇒ Output = 1 for CCMPH cycles, low for the rest of the period
- For 0 < CCMPL < 0xFF
  - CCMPH = 0 ⇒ Output = 0
  - If 0 < CCMPH ≤ CCL ⇒ Output = 1 for CCMPH cycles, low for the rest
  - CCMPH = CCMPL + 1 ⇒ Output = 1

**Figure 21-9. 8-bit PWM Mode**




#### 21.3.3.2 Output

If ASYNC in TCB.CTRLB is written to '0' ('1'), the output pin is driven synchronously (asynchronously) to the TCB clock. The bits CCMPINIT, CCMPEN, and CNTMODE in TCB.CTRLB control how the synchronous output is driven. The bits CCMPINIT, CCMPEN, and CNTMODE in TCB.CTRLB control how the synchronous output is driven. The different configurations and their impact on the output is listen in the table below.

**Table 21-3. Synchronous Output**

| CNTMODE | Output, CTRLB='0', CCMPEN=1 | Output, CTRLB='1', CCMPEN=1 |
|---|---|---|
| Single shot mode | Output high when counter starts and output low when counter stops | Output high when event arrives and output low when counter stops |
| 8-bit PWM mode | PWM mode output | PWM mode output |
| Modes except single shot and PWM | Bit CCMPINIT in TCB.CTRLB | Bit CCMPINIT in TCB.CTRLB |

#### 21.3.3.3 Noise Canceler

The noise canceler improves noise immunity by using a simple digital filter scheme. When the noise filter is enabled the peripheral monitors the event channel and keeps a record of the last four observed samples. If four consecutive samples are equal, the input is considered to be stable and the signal is fed to the edge detector.

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 input compare register.

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

#### 21.3.3.4 Synchronized with TCA0

The TCB can be configured to use the clock (CLK_TCA) of the Timer/Counter type A (TCA0) by writing 0x2 to the Clock Select bit field (CLKSEL) in the Control A register (TCB.CTRLA). In this setting the TCB will count either on the prescaled clock signal from TCA0 which is configured by CLKSEL in TCA.CTRLA,

or event clocking for TCA configured by EVACT in TCA.EVCTRL. Event clocking from TCA0 is only used if enable count on event (CNTEI) is set to '1' in TCA.EVCTRL.

When the Synchronize Update bit (SYNCUPD) in the Control A register (TCB.CTRLA) is written to '1', the TCB counter will restart when the TCA0 counter restarts.

**Related Links**

Block Diagram

### 21.3.4 Events

If TCB is connected to the Event System as an event generator, the TCB will generate a strobe on the connected event channel when the configured event condition is met.

The peripheral accepts one event input. If the Capture Event Input Enable bit (CAPTEI) in the Event Control register (TCB.EVCTRL) is written to '1', incoming events will result in an event action as defined by the Event Edge bit (EDGE) in TCB.EVCTRL. A change on event line needs to be held for at least on system clock cycle to guarantee action on event except single shot mode.

**Related Links**

EVCTRL

EVSYS - Event System

### 21.3.5 Interrupts

**Table 21-4. Available Interrupt Vectors and Sources**

| Offset | Name | Vector Description | Conditions |
|---|---|---|---|
| 0x00 | CAPT | TCB interrupt | Depending on operating mode. See description of CAPT in TCB.INTFLAG. |

When an interrupt condition occurs, the corresponding Interrupt Flag is set in the Interrupt Flags register of the peripheral (*peripheral*.INTFLAGS).

An interrupt source is enabled or disabled by writing to the corresponding enable bit in the peripheral's Interrupt Control register (*peripheral*.INTCTRL).

An interrupt request is generated when the corresponding interrupt source is enabled and the Interrupt Flag is set. The interrupt request remains active until the Interrupt Flag is cleared. See the peripheral's INTFLAGS register for details on how to clear Interrupt Flags.

**Related Links**

CPUINT - CPU Interrupt Controller

INTFLAGS

### 21.3.6 Sleep Mode Operation

TCB will halt operation in the Power Down Sleep mode. Standby sleep operation is dependent on the Run in Standby bit (RUNSTDBY) in the Control A register (TCB.CTRLA).

### 21.3.7 Synchronization

Not applicable.

### 21.3.8 Configuration Change Protection

Not applicable.

## 21.4 Register Summary - TCB

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | | RUNSTDBY | | SYNCUPD | | CLKSEL[1:0] | | ENABLE |
| 0x01 | CTRLB | 7:0 | | ASYNC | CCMPINIT | CCMPEN | | CNTMODE[2:0] | | |
| 0x02<br>...<br>0x03 | Reserved | | | | | | | | | |
| 0x04 | EVCTRL | 7:0 | | FILTER | | EDGE | | | | CAPTEI |
| 0x05 | INTCTRL | 7:0 | | | | | | | | CAPT |
| 0x06 | INTFLAGS | 7:0 | | | | | | | | CAPT |
| 0x07 | STATUS | 7:0 | | | | | | | | RUN |
| 0x08 | DBGCTRL | 7:0 | | | | | | | | DBGRUN |
| 0x09 | TEMP | 7:0 | TEMP[7:0] | | | | | | | |
| 0x0A | CNT | 7:0 | CNT[7:0] | | | | | | | |
| 0x0B | | 15:8 | CNT[15:8] | | | | | | | |
| 0x0C | CCMP | 7:0 | CCMP[7:0] | | | | | | | |
| 0x0D | | 15:8 | CCMP[15:8] | | | | | | | |

## 21.5 Register Description

### 21.5.1 Control A

**Name:** CTRLA
**Offset:** 0x00
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | RUNSTDBY | | SYNCUPD | | CLKSEL[1:0] | | ENABLE |
| Access | | R/W | | R/W | | R/W | R/W | R/W |
| Reset | | 0 | | 0 | | 0 | 0 | 0 |

**Bit 6 – RUNSTDBY: Run Standby**
Writing a '1' to this bit will enable the peripheral to run in Standby sleep mode. Not applicable when CLKSEL is set to 0x2 (CLK_TCA).

**Bit 4 – SYNCUPD: Synchronize Update**
When this bit is written to '1', the TCB will restart whenever the TCA0 is restarted.

**Bits 2:1 – CLKSEL[1:0]: Clock Select**
Writing these bits selects the clock source for this peripheral.

| Value | Description |
|---|---|
| 0x0 | CLK_PER |
| 0x1 | CLK_PER / 2 |

| Value | Description |
|---|---|
| 0x2 | Use CLK_TCA from TCA0 |
| 0x3 | Reserved |

**Bit 0 – ENABLE: Enable**
Writing this bit to '1' enables the Timer/Counter type B peripheral.

### 21.5.2 Control B

**Name:** CTRLB
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | ASYNC | CCMPINIT | CCMPEN | | CNTMODE[2:0] | | |
| Access | | R/W | R/W | R/W | | R/W | R/W | R/W |
| Reset | | 0 | 0 | 0 | | 0 | 0 | 0 |

**Bit 6 – ASYNC: Asynchronous Enable**
Writing this bit to '1' will allow asynchronous updates of the TCB output signal in single shot mode

| Value | Description |
|---|---|
| 0 | The output will go HIGH when the counter actually starts |
| 1 | The output will go HIGH when an Event arrives |

**Bit 5 – CCMPINIT: Compare/Capture Pin Initial Value**
This bit is used to set the initial output value of the pin when a pin output is used.

| Value | Description |
|---|---|
| 0 | Initial pin state is LOW |
| 1 | Initial pin state is HIGH |

**Bit 4 – CCMPEN: Compare/Capture Output Enable**
This bit is used to set the output value of the Compare/Capture Output.

| Value | Description |
|---|---|
| 0 | Compare/Capture Output is zero |
| 1 | Compare/Capture Output has a valid value |

**Bits 2:0 – CNTMODE[2:0]: Timer Mode**
Writing these bits selects the timer mode.

| Value | Description |
|---|---|
| 0x0 | Periodic interrupt mode |
| 0x1 | Timeout check mode |
| 0x2 | Input capture on event mode |
| 0x3 | Input capture frequency measurement mode |
| 0x4 | Input capture pulse width measurement mode |
| 0x5 | Input capture frequency and pulse width measurement mode |

| Value | Description |
|---|---|
| 0x6 | Single shot mode |
| 0x7 | 8-bit PWM mode |

### 21.5.3 Event Control

**Name:** EVCTRL
**Offset:** 0x04
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | FILTER | | EDGE | | | | CAPTEI |
| Access | | R/W | | R/W | | | | R/W |
| Reset | | 0 | | 0 | | | | 0 |

**Bit 6 – FILTER: Input Capture Noise Cancellation Filter**
Writing this bit to '1' enables the input capture noise cancellation unit.

**Bit 4 – EDGE: Event Edge**
This bit is used to select the Event edge. The effect of this bit is dependent on the selected Count Mode (CNTMODE) in TCB.CTRLB.

| Count Mode | EDGE | Positive Edge | Negative Edge |
|---|---|---|---|
| Periodic Interrupt Mode | 0 | Not Applicable | Not Applicable |
| | 1 | Not Applicable | Not Applicable |
| Timeout Check Mode | 0 | Start counter | Stop counter |
| | 1 | Stop counter | Start counter |
| Input Capture on Event Mode | 0 | Input capture frequency and pulse width measurement mode | Not Applicable |
| | 1 | Not Applicable | Capture = count |
| Input Capture Frequency Measurement Mode | 0 | Capture = count, initialize, interrupt | Not Applicable |
| | 1 | Not Applicable | Capture = count, initialize, interrupt |
| Input Capture Pulse Width Measurement Mode | 0 | Initialize | Capture = count, interrupt |
| | 1 | Capture = count, interrupt | Initialize |
| Input Capture Frequency and Pulse Width Measurement Mode | 0 | On 1st Positive: initialize<br>On following Negative: capture<br>2nd Positive: stop, interrupt | |
| | 1 | On 1st Negative: initialize<br>On following Positive: capture | |

| Count Mode | EDGE | Positive Edge | Negative Edge |
|---|---|---|---|
| | | 2nd Negative: stop, interrupt | |
| Single Shot Mode | 0 | Start counter | Not Applicable |
| | 1 | Start counter | Start counter |
| 8-bit PWM Mode | 0 | Not Applicable | Not Applicable |
| | 1 | Not Applicable | Not Applicable |

**Bit 0 – CAPTEI: Capture Event Input Enable**
Writing this bit to '1' enables the event input capture.

### 21.5.4 Interrupt Control

**Name:** INTCTRL
**Offset:** 0x05
**Reset:** 0x00
**Property:** -

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

**Bit 0 – CAPT: Capture Interrupt Enable**
Writing this bit to '1' enables the Capture interrupt.

### 21.5.5 Interrupt Flags

**Name:** INTFLAGS
**Offset:** 0x06
**Reset:** 0x00
**Property:** -

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

**Bit 0 – CAPT: Interrupt Flag**
This bit is set when an interrupt occurs. The interrupt conditions are dependent on the Counter mode (CNTMODE) in TCB.CTRLB.

This bit is cleared by writing a '1' to it or when the Capture register is read in capture mode.

| Counter Mode | Interrupt Set Condition |
|---|---|
| Periodic Interrupt Mode | Set when the counter reaches TOP |
| Timeout Check Mode | Set when the counter reaches TOP |
| Input Capture on Event Mode | Set when an event occurs and the capture register is loaded, Flag clears when capture is read |
| Input Capture Frequency Measurement Mode | Set on edge when the capture register is loaded and count initialized, Flag clears when capture is read |
| Input Capture Pulse Width Measurement Mode | Set on a edge when the capture register is loaded, previous edge initialized the count, Flag clears when capture is read |
| Input Capture Frequency and Pulse Width Measurement Mode | Set on second (positive or negative) edge when the counter is stopped, Flag clears when capture is read |
| Single Shot Mode | Set when counter reaches TOP |
| 8-bit PWM Mode | Set when the counter reaches CCH |

### 21.5.6 Status

**Name:** STATUS
**Offset:** 0x07
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | RUN |
| Access | | | | | | | | R |
| Reset | | | | | | | | 0 |

**Bit 0 – RUN: Run**
When the counter is running, this bit is set to '1'. When the counter is stopped, this bit is cleared '0'.

The bit it is read only and cannot be set by UPDI.

### 21.5.7 Debug Control

**Name:** DBGCTRL
**Offset:** 0x08
**Reset:** 0x00
**Property:** -

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

**Bit 0 – DBGRUN: Debug Run**

| Value | Description |
|---|---|
| 0 | The peripheral is halted in break debug mode and ignores events. |
| 1 | The peripheral will continue to run in break debug mode when the CPU is halted. |

### 21.5.8 Temporary Value

The Temporary register is used by the CPU for single-cycle, 16-bit access to the 16-bit registers of this peripheral. It can also be read and written by software. See also Accessing 16-bit Registers. There is one common Temporary register for all the 16-bit registers of this peripheral.

**Name:** TEMP
**Offset:** 0x09
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TEMP[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 – TEMP[7:0]: Temporary Value**

### 21.5.9 Count

The TCB.CNTL and TCB.CNTH register pair represents the 16-bit value TCB.CNT. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

CPU and UPDI write access has priority over internal updates of the register.

**Name:** CNT
**Offset:** 0x0A
**Reset:** 0x00
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | CNT[15:8] | | | | | | | |
| 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 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CNT[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 15:8 – CNT[15:8]: Count Value high**
These bits hold the MSB of the 16-bit counter register.

**Bits 7:0 – CNT[7:0]: Count Value low**
These bits hold the LSB of the 16-bit counter register.

### 21.5.10 Capture/Compare

The TCB.CCMPL and TCB.CCMPH register pair represents the 16-bit value TCB.CCMP. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

This register has different functions depending on the mode of operation:

- For capture operation, these registers contain the captured value of the counter at the time the capture occurs
- In periodic interrupt/timeout and single shot mode this register acts as the TOP value.
- In 8-bit PWM mode, TCB.CCMPL and TCB.CCMPH act as two independent registers.

**Name:** CCMP
**Offset:** 0x0C
**Reset:** 0x00
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | CCMP[15:8] | | | | | | | |
| 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 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CCMP[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 15:8 – CCMP[15:8]: Capture/Compare Value high byte**
These bits hold the MSB of the 16-bit compare, capture and top value

**Bits 7:0 – CCMP[7:0]: Capture/Compare Value low byte**
These bits hold the LSB of the 16-bit compare, capture and top value

# 22. TCD - 12-bit Timer/Counter Type D

## 22.1 Features

- 12-bit timer/counter
- Programmable prescaler
- Double buffered compare registers
- Waveform generation
  - One ramp mode
  - Two ramp mode
  - Four ramp mode
  - Dual-slope mode
- Two separate input capture, double buffered
- Connection to event system
  - Programmable filter
- Conditional waveform on external events
  - Fault handling
  - Input blanking
  - Overload protection function
  - Fast emergency stop by hardware
- Supports both half bridge and full bridge output

## 22.2 Overview

The Timer/Counter type D (TCD) is a high performance waveform controller that consists of an asynchronous counter, a prescaler, compare logic, capture logic, and control logic. The purpose of the TCD is to control power applications like LED, motor control, H-bridge and power converters.

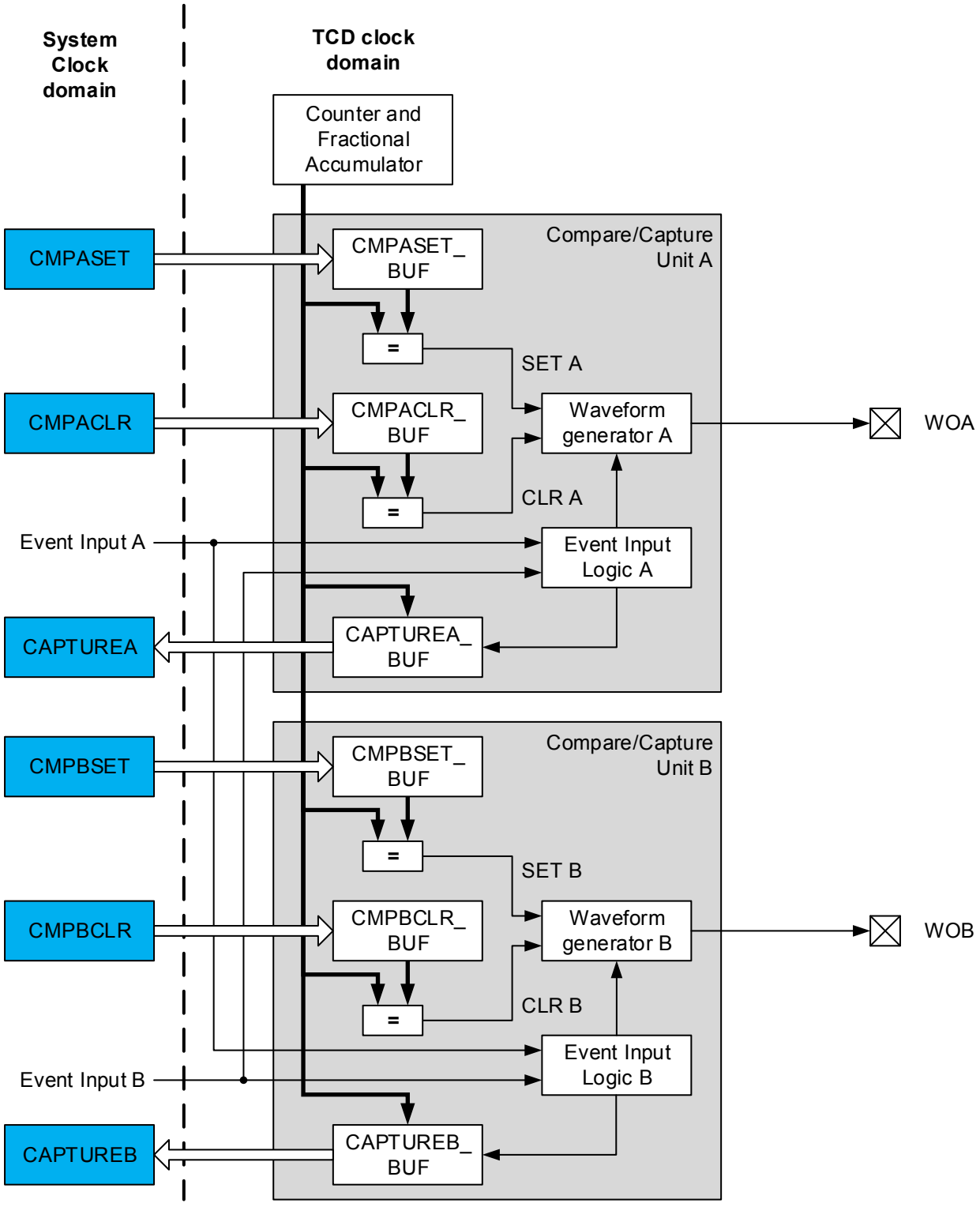
The TCD contains an counter that can run on a clock which is asynchronous from the system clock. It contains compare logic that can generate two independent outputs with optional dead time. It is connected to the event system for capture and deterministic fault control. The timer/counter can generate interrupts and events on compare match and overflow.

This device provides one instance of the TCD peripheral, TCD0.

© 2017 Microchip Technology Inc.

### 22.2.1 Block Diagram

**Figure 22-1. Timer/Counter Block Diagram**




The TCD core is asynchronous to the system clock. The timer/counter consist of two compare/capture units, each with a separate waveform output. In addition there are two extra waveform outputs which can be equal to the output from one of the units. The compare registers CMPxSET, CMPxCLR are stored in

the respective registers (TCD.CMPxSET, TCD.CMPxCLR), which consist of both a low and a high byte. The registers are synchronized to the TCD domain after writing to the registers.

During normal operation, the counter value is continuously compared to the compare registers. This is used to generate both interrupts and events.

The TCD can use the input events in ten different input modes, selected separately for the two input events. The input mode defines how the input event will affect the outputs, and where in the TCD cycle the counter should go when an event occurs.

### 22.2.2 Signal Description

| Signal | Description | Type |
|---|---|---|
| WOA | TCD waveform output A | Digital output |
| WOB | TCD waveform output B | Digital output |

### 22.2.3 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 22-1. TCD System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | Yes | WOA/B |
| Interrupts | Yes | CPUINT |
| Events | Yes | EVSYS |
| Debug | Yes | UPDI |

**Related Links**

Clocks
Debug Operation
Interrupts
Events

#### 22.2.3.1 Clocks

The TCD can be connected directly to the internal 16/20MHz RC Oscillator (OSC20M), to an external clock, or to the System clock. This is configured by the Clock Select bit field (CLKSEL) in the Control A register (TCD.CTRLA).

**Related Links**

CLKCTRL - Clock Controller

#### 22.2.3.2 I/O Lines and Connections

Using the I/O lines of the peripheral requires configuration of the I/O pins.

**Related Links**

I/O Multiplexing and Considerations
PORT - I/O Pin Configuration

#### 22.2.3.3 Interrupts

Using the interrupts of this peripheral requires the Interrupt Controller to be configured first.

**Related Links**

CPUINT - CPU Interrupt Controller

SREG

Interrupts

#### 22.2.3.4 Events

The events of this peripheral are connected to the Event System.

**Related Links**

EVSYS - Event System

#### 22.2.3.5 Debug Operation

When run-time debugging, this peripheral will continue normal operation. Halting the CPU in debugging mode will halt normal operation of the peripheral.

This peripheral can be forced to operate with halted CPU by writing a '1' to the Debug Run bit (DBGRUN) in the Debug Control register of the peripheral (*peripheral*.DBGCTRL).

When the Fault Detection bit (FAULTDET in TCD.DBGCTRL) is written to '1' and the CPU is halted in debug mode, an event/fault is created on both input event channels. These events/faults will last as long as the break, and can serve as safeguard in Debug mode, e.g. by forcing external components off.

If the peripheral is configured to require periodical service by the CPU through interrupts or similar, improper operation or data loss may result during halted debugging.

**Related Links**

UPDI - Unified Program and Debug Interface

### 22.2.4 Definitions

The following definitions are used throughout the documentation:

**Table 22-2. Timer/Counter definitions**

| Name | Description |
|---|---|
| TCD cycle | The sequence of four states that the counter need to go through before it has returned to the same position. |
| Input Blanking | Functionality to ignore event input for a programmable time in a selectable part of the TCD cycle. |
| Asynchronous output control | Allows the Event to override the output instantly when the Event occurs. It is used for handling non-recoverable faults. |
| One ramp | Counter is reset to zero once during a TCD cycle. |
| Two ramp | Counter is reset to zero two times during a TCD cycle. |

| Name | Description |
|---|---|
| Four ramp | Counter is reset to zero four times during a TCD cycle. |
| Dual ramp | Counter count both up and down between zero and selected top value. |

## 22.3 Functional Description

### 22.3.1 Initialization and Disabling

To initialize the TCD:

1. Configure the static registers to the desired functionality.
2. Write desired initial values to the double-buffered registers.
3. Ensure that the Enable Ready bit (ENRDY) in the Status register (TCD.STATUS) is set to '1'.
4. Enable the TCD by writing a '1' to the ENABLE bit in the Control A register (TCD.CTRLA).

It is possible to disable the TCD in two different ways:

1. By writing a '0' to ENABLE in TCD.CTRLA. This disables the TCD instantly when synchronized to the TCD core domain.
2. By writing a '1' to the Disable at End of Cycle Strobe bit (DISEOC) in the Control E register (TCD.CTRLE). This disables the TCD at the end of the TCD cycle.

The bit fields in the TCD.CTRLA register are enable-protected, with exception of the ENABLE bit. They can only be written when ENABLE is written to '0' first.

**Related Links**

Register Synchronization Categories

### 22.3.2 Operation

#### 22.3.2.1 Register Synchronization Categories

Most of the IO registers need to be synchronized to the asynchronous TCD core clock domain. This is done in different ways for different register categories:

- Command and Enable Control registers
- Doubled-buffered registers
- Static registers
- Normal IO and STATUS registers

See Table 22-3 for categorized registers.

**Command and Enable Registers**

Because of synchronization between the clock domains it is only possible to change the Enable bits while the Enable Ready bit (ENRDY) in the Status register (TCD.STATUS) is '1'.

The Control E register commands (TCD.CTRLE) are automatically synchronized to the TCD core domain when the TCD is enabled and as long as there not a synchronization ongoing already. Check in the Status register if the Command Ready bit (CCMDRDY) is '1' (TCD.STATUS) to ensure that it is possible to write a new command. TCD.CTRLE is a strobe register that will clear itself when the command is done.

The Control E register commands are:

- Synchronize at end of TCD cycle: synchronizes all doubled buffered registers to TCD clock domain at the end of the TCD cycle
- Synchronize: synchronized all doubled buffered registers to TCD clock domain when the command is synchronized to TCD clock domain
- Restart: Restarts the TCD counter
- Software Capture A: Capture TCD counter value to TCD.CAPTUREA
- Software Capture B: Capture TCD counter value to TCD.CAPTUREB

**Double-Buffered Registers**

The doubled-buffered registers can be updated in normal IO writes while the TCD is enabled and no synchronization between the two clock domains is ongoing. Check that the CMDRDY bit in TCD.STATUS is '1' to ensure that it is possible to update the doubled buffered IO registers. The values will be synchronized to the TCD core domain when a synchronization command is sent or when the TCD is enabled.

**Static Registers**

The static registers are kept static whenever the TCD is enabled. That means that these registers must be configured before enabling the TCD. It is not possible to write to these registers as long as the TCD is enabled. To see if the TCD is enabled, check if ENABLE in TCD.CTRLA is reading '1'.

**Normal IO and Status Registers**

The read-only registers inform about synchronization status and values synchronized from the core domain. The reset of these registers and normal IO registers are not constrained by any synchronization between the domains.

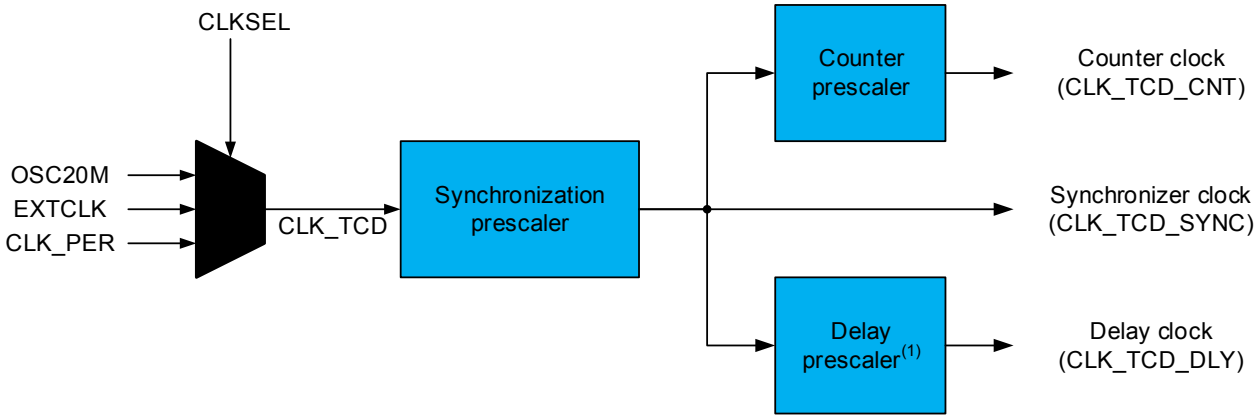
**Table 22-3. Categorization of Registers**

| Enable and Command registers | Doubled-buffered registers | Static registers | Read-only registers | Normal IO registers |
|---|---|---|---|---|
| TCD.CTRLA (ENABLE bit) | TCD.DLYCTRL | TCD.CTRLA (All bits Except ENABLE bit) | TCD.STATUS | TCD.INTCTRL |
| TCD.CTRLE | TCD.DLYVAL | TCD.CTRLB | TCD.CAPTUREA | TCD.INTFLAGS |
| | TCD.DITCTRL | TCD.CTRLC | TCD.CAPTUREB | |
| | TCD.DITVAL | TCD.CTRLD | | |
| | TCD.DBGCTRL | TCD.EVCTRLA | | |
| | TCD.CMPASET | TCD.EVCTRLB | | |
| | TCD.CMPACLR | TCD.INPUTCTRLA | | |
| | TCD.CMPBSET | TCD.INPUTCTRLB | | |
| | TCD.CMPBCLR | TCD.FAULTCTRL | | |

#### 22.3.2.2 Clock Selection and Prescalers

The TCD can select between 3 different clock sources that can be prescaled. There are three different prescalers with separate controls as shown below.

**Figure 22-2. Clock Selection and Prescalers Overview**




1. Used by input blanking/delay event out

The TCD synchronizer clock is separate from the other module clocks enabling faster synchronization between the TCD domain and the IO domain. The total prescalingfor the counter is:

$$\text{SYNCPRESC_division_factor} \times \text{CNTPRESC_division_factor}$$

The Delay prescaler is used to prescale the clock used for the input blanking/ delayed event output functionality. The prescaler can be configured independently allowing separate range and accuracy settings from the counter functionality.

#### 22.3.2.3 Waveform Generation Modes

The TCD provides four different waveform generation modes. The waveform generation modes determine how the counter is counting during a TCD cycle, and when the compare values are matching. A TCD cycle is split into these states:

- Dead time WOA (DTA)
- On time WOA (OTA)
- Dead time WOB (DTB)
- On time WOB (OTB)

In a standard configuration all states are present in the order they are listed and they are non-overlapping. The compare values Compare A Set (TCD.CMPASET), Compare A Clear (TCD.CMPACLR), Compare B Set (TCD.CMPBSET) and Compare B Clear (TCD.CMPBCLR) defines when each of the states are ending and the next is beginning. There are four different ways to go through a TCD cycle. The different ways are called Waveform Generation Modes. They are controlled by the Waveform Generation Mode bits (WGMODE) in the Control A register (TCD.CTRLA). The waveform generation modes are:

- One Ramp mode
- Two Ramp mode
- Four Ramp mode
- Dual Slope mode

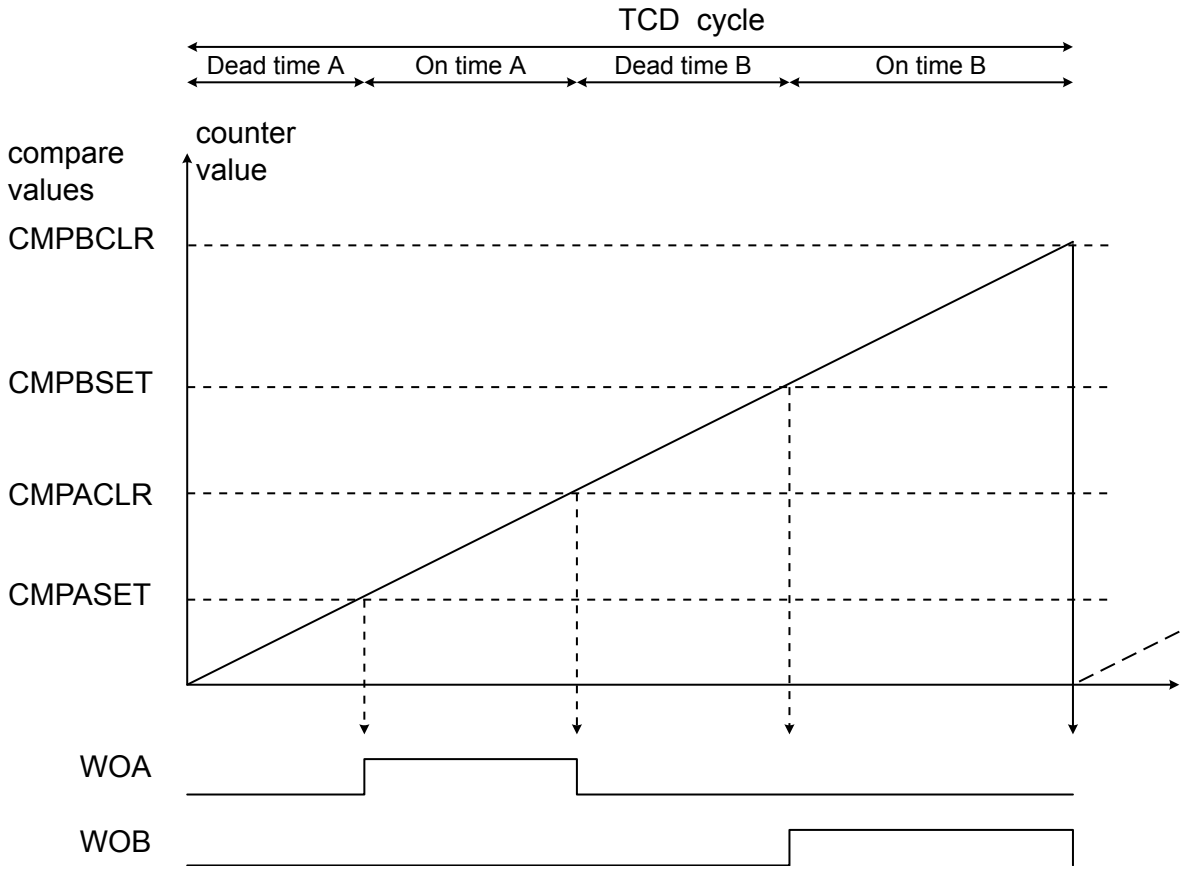
The name indicates how the counter is operating during one TCDcycle.

**One Ramp Mode**

In One Ramp Mode, TCD counter counts up until it reaches the CMPBCLR value. Then the TCD cycle is done and the counter restarts from 0x000, beginning a new TCD cycle. The TCD cycle period is

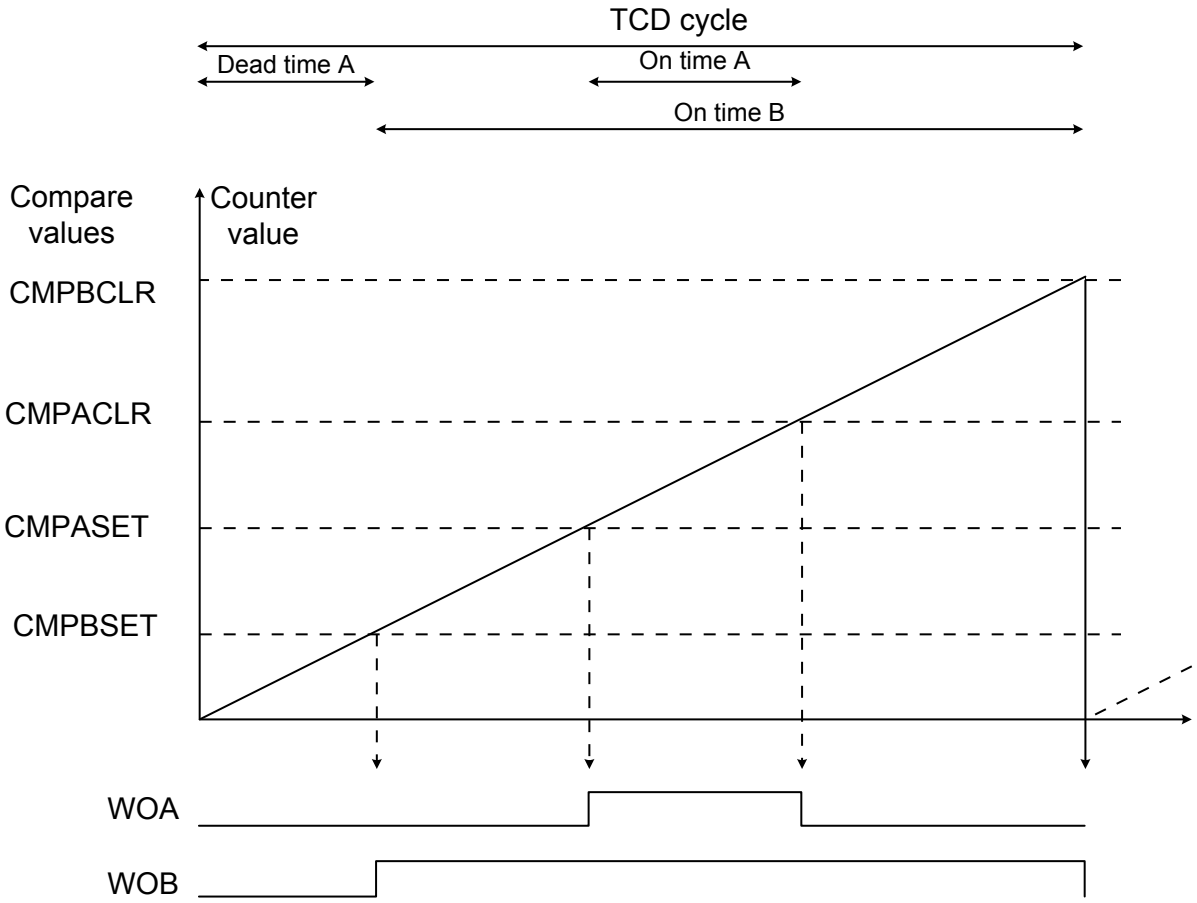
$$T_{\text{TCD_cycle}} = \frac{(\text{CMPBCLR} + 1)}{f_{\text{CLK_TCD_CNT}}}$$

**Figure 22-3. One Ramp Mode**




In the figure above, *CMPASET* < *CMPACLR* < *CMPBSET* < *CMPBCLR*. This is required in One Ramp Mode to avoid overlapping outputs. The figure below is an example where *CMPBSET* < *CMPASET* < *CMPACLR* < *CMPBCLR*, resulting in an overlap of the outputs.

**Figure 22-4. One Ramp Mode with *CMPBSET < CMPASET***



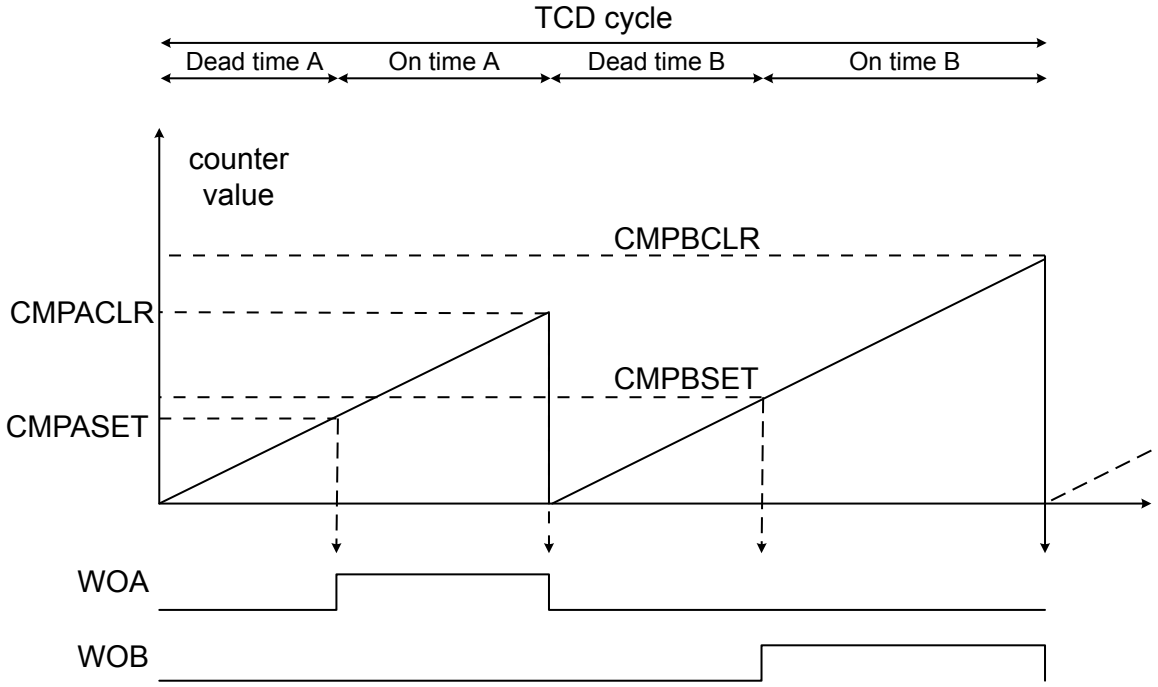

If any of the other compare values are bigger than CMPBCLR it will never be triggered when running in One ramp mode. And if The CMPACLR is smaller than the CMPASET value, the clear value will not have any effect.

#### Two Ramp Mode

In Two Ramp Mode the TCD counter counts up until it reaches the CMPACLR value, then it resets and counts up until it reaches the CMPBCLR value. Then, the TCD cycle is done and the counter restarts from 0x000, beginning a new TCD cycle. The TCD cycle period is give by:

$$T_{\mathrm{TCD_cycle}} = \frac{(\mathrm{CMPACLR} + 1 + \mathrm{CMPBCLR} + 1)}{f_{\mathrm{CLK_TCD_CNT}}}$$

**Figure 22-5. Two Ramp Mode**




In the figure above, *CMPASET* < *CMPACLR and CMPBSET* < *CMPBCLR*. This causes the outputs to go high. There are no restrictions on the CMPASET/CLR compared to the CMPBSET/CLR values.

In two ramp mode it is not possible to get overlapping outputs.

**Four Ramp Mode**

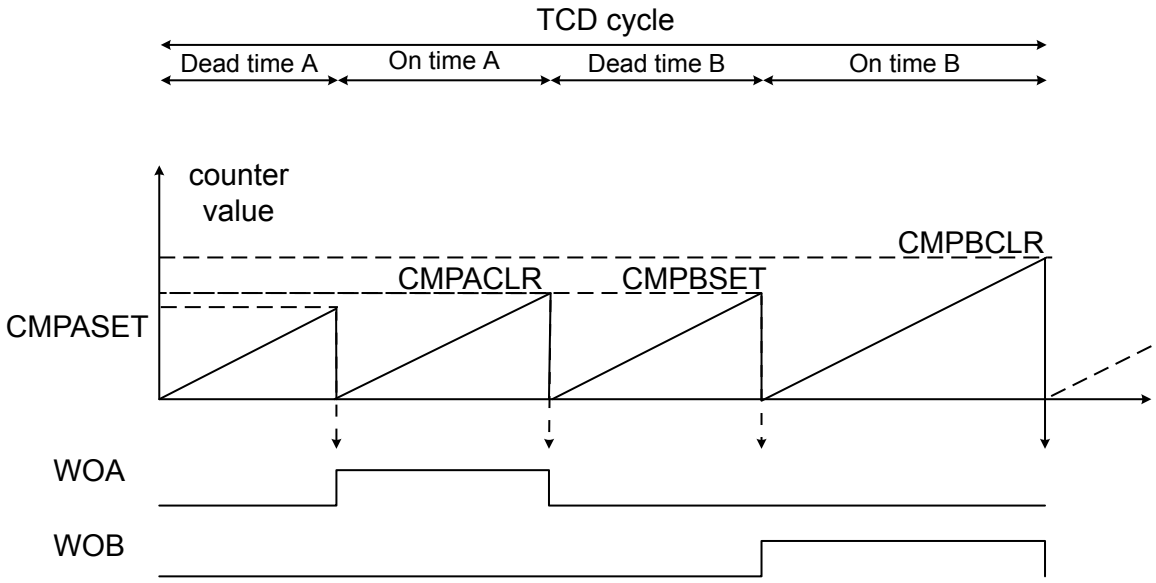
In Four Ramp Mode the TCD cycle is following this pattern:

1. A TCD cycle begins with the TCD counter counting up from zero until it reaches the CMPASET value, and resets to zero.
2. The Counter counts up from zero until it reaches the CMPACLR value, and resets to zero.
3. The Counter counts up from zero until it reaches the CMPBSET value, and resets to zero.
4. The Counter counts up from zero until it reaches the CMPBCLR value, and ends the TCD cycle by resetting to zero.

The TCD cycle period is given by

$$T_{\mathrm{TCD_cycle}} = \frac{(\mathrm{CMPASET} + 1 + \mathrm{CMPACLR} + 1 + \mathrm{CMPBSET} + 1 + \mathrm{CMPBCLR} + 1)}{f_{\mathrm{CLK_TCD_CNT}}}$$

**Figure 22-6. Four Ramp Mode**




There are no restrictions on the compare values compared to each others.

In Four Ramp Mode it is not possible to get overlapping outputs.

**Dual Slope Mode**

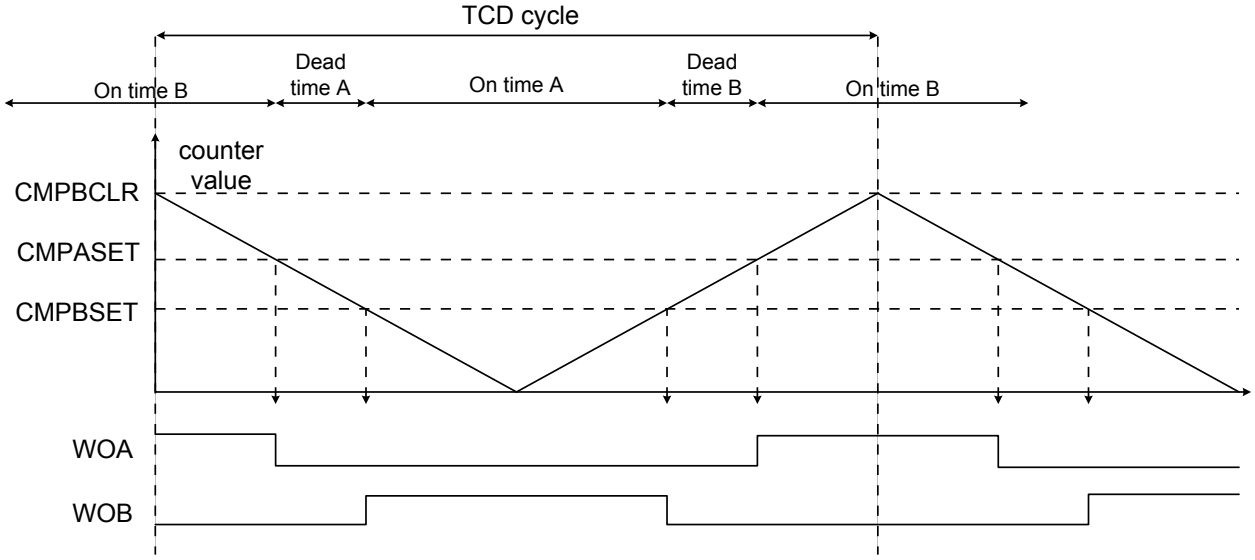
In Dual Slope mode, a TCD cycle consists of the TCD counter counting down from CMPBCLR value to zero, and up again to the CMPBCLR value. This gives a TCD cycle period:

$$T_{\mathrm{TCD_cycle}} = \frac{2 \times (\mathrm{CMPBCLR} + 1)}{f_{\mathrm{CLK_TCD_CNT}}}$$

The WOA output is set when the TCD counter counts up and matches the CMPASET value. WOA is cleared when the TCD counter counts down and matches the CMPASET value.

The WOB output is set when the TCD counter counts down and matches the CMPBSET value. WOB is cleared when the TCD counter counts up and matches the CMPBSET value.

**Figure 22-7. Dual Slope Mode**

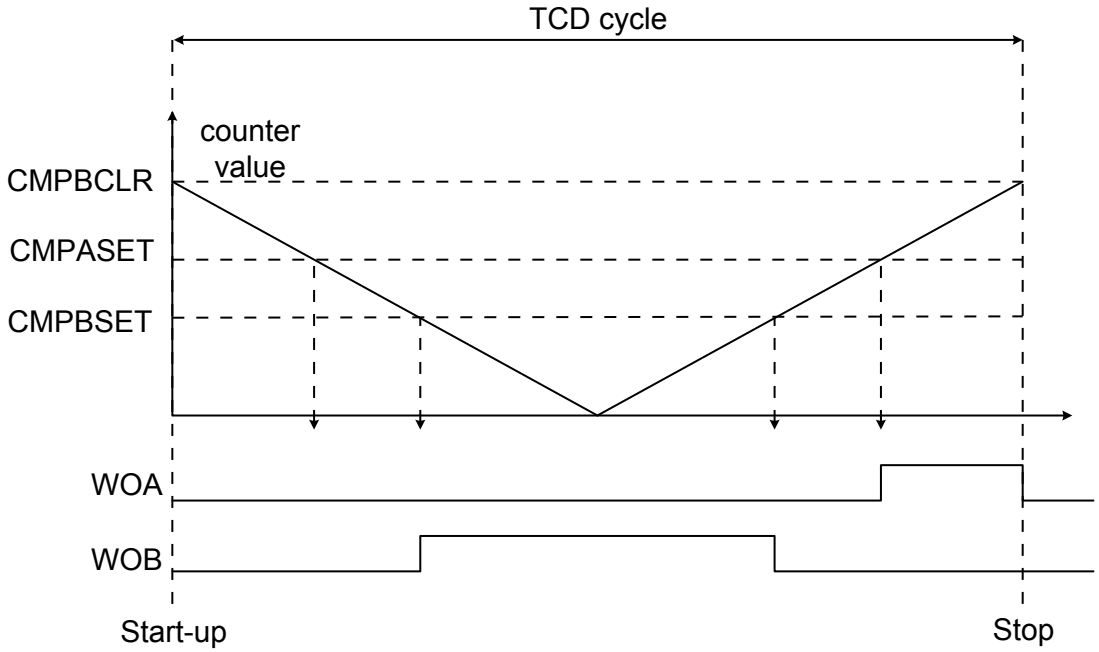



The outputs will be overlapping if CMPBSET > CMPASET.

CMPACLR is not used in Dual Slope Mode. Writing a value to CMPACLR has no effect.

When starting the TCD in Dual Slope Mode, the TCD counter starts at the CMPBCLR value and counts down. The WOA will not be set before the end of the first TCD cycle.

**Figure 22-8. Dual Slope Mode Starting and Stopping**




#### 22.3.2.4 TCD Inputs

The TCD has two inputs that are connected to the Event System, Input A and Input B. Each input has functionality that are connected to corresponding output (WOA and WOB). That functionality is controlled by the Event Control x registers (TCD.EVCTRLA and TCD.EVCTRLB) and the Input Control x registers (TCD.INPUTACTRL and TCD.INPUTBCTRL).

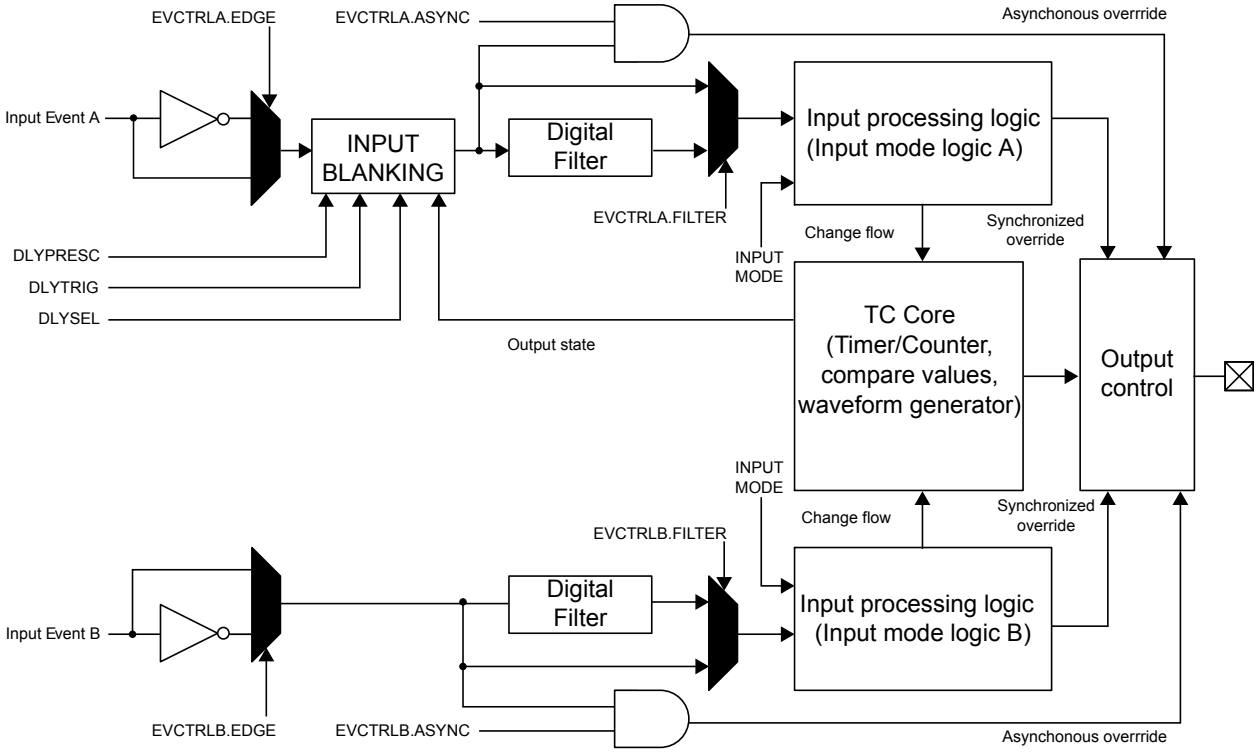
To enable the input Events, write a '1' to the Trigger Event Input Enable bit (TRIGEI) in the Event Control register (TCD.EVCTRLx). The inputs will be used as a fault detect and/or capture trigger. To enable capture trigger, write a '1' to the ACTION bit in Event Control register (TCD.EVCTRLx).

There are ten different input modes for the fault detection. The two inputs have the same functionality, except for input blanking which is only supported by input A. Input blanking is configured by the Delay Control and Delay Value registers (TCD.DLYCTRL and TCD.DLYVAL).

The inputs are connected to the Event System. The connections between the Event source and the TCD input must be configured in the Event System.

An overview of the input system is shown below.

**Figure 22-9. TCD Input Overview**




There is a delay of 2-3 clock cycles on the TCD synchronizer clock between receiving the input Event and processing it and overriding the outputs. If using the asynchronous Event detection, the outputs will override instantly outside the input processing.

**Input Blanking**

Input blanking functionality are masking out the input Events for a programmable time in a selectable part of the TCD cycle. Input blanking can be used to mask out "false" input Events that are triggered right after changes on the outputs.

To enable input blanking, write 0x1 to the Delay Select bit field in the Delay Control register (DLYSEL in TCD.DLYCTRL). The trigger source is selected by the Delay Trigger bit field (DLYTRIG in TCD.DLYCTRL).

Input blanking uses the Delay clock: after a trigger, a counter is counting up until the Delay Value (DLYVAL in TCD.DLYVAL) is reached before input blanking is turned off. The TCD delay clock is a prescaled version of the Synchronization clock. The division factor is set by the Delay Prescaler bit field in the Delay Control register (DLYPRESC in TCD.DLYCTRL). The duration of the input blanking is given by

$$t_{\mathrm{BLANK}} = \frac{\mathrm{DLYPRESC_division_factor} \times \mathrm{DLYVAL}}{f_{\mathrm{CLK_TCD_SYNC}}}$$

Input blanking is using the same logic as the programmable output Event. For this reason it is not possible to use both at the same time.

**Digital Filter**

The digital filter for Event input x is enabled by writing a '1' to the FILTER bit in the Event Control x register (TCD.EVCTRLx). When the digital filter is enabled, any pulse lasting less than 4 Counter clock cycles will be filtered out. Any change on incoming event will therefore take 4 Counter clock cycles before it will affect the input processing logic.

**Asynchronous Event Detection**

To enable asynchronous Event detection on an input Event, write a '1' to the Asynchronous Event Control bit (ASYNC) in the Event Control register (TCD.EVCTRLx).

The Asynchronous Event Detection makes it possible to asynchronously override the output when the input Event occurs. What the Input event will do is depending on Input Mode for the event input. The outputs have direct override while the counter flow will be changed when the event is synchronized to the Synchronization clock.

It is not possible to use both asynchronous Event detection and digital filtering at the same time.

**Input Modes**

The user can select between 10 input modes. The selection is done by writing the Input Mode bit field (INPUTMODE) in the Input control x register (TCD.INPUTCTRLx).

**Table 22-4. Input mode description**

| INPUTMODE | Description |
|---|---|
| 0x0 | Input has no action |
| 0x1 | Stop output, jump to opposite compare cycle and wait |
| 0x2 | Stop output, execute opposite compare cycle and wait |
| 0x3 | Stop output, execute opposite compare cycle while fault active |
| 0x4 | Stop all outputs, maintain frequency |
| 0x5 | Stop all outputs, execute dead time while fault active |
| 0x6 | Stop all outputs, jump to next compare cycle and wait |
| 0x7 | Stop all outputs, wait for software action |
| 0x8 | Stop output on edge, jump to next compare cycle |
| 0x9 | Stop output on edge, maintain frequency |
| 0xA | Stop output at level, maintain frequency |
| other | Reserved |

Not all input modes works in all waveform generation modes. Below is a table that shows what waveform generation modes the different input modes are valid in.

**Table 22-5. Ramp mode the different input modes are vaild in**

| INPUTMODE | One Ramp mode | Two ramp mode | Four ramp mode | Dual slope mode |
|---|---|---|---|---|
| 0x1 | Valid | Valid | Valid | Do not use |
| 0x2 | Do not use | Valid | Valid | Do not use |
| 0x3 | Do not use | Valid | Valid | Do not use |
| 0x4 | Valid | Valid | Valid | Valid |
| 0x5 | Do not use | Valid | Valid | Do not use |
| 0x6 | Do not use | Valid | Valid | Do not use |
| 0x7 | Valid | Valid | Valid | Valid |

© 2017 Microchip Technology Inc.

| INPUTMODE | One Ramp mode | Two ramp mode | Four ramp mode | Dual slope mode |
|---|---|---|---|---|
| 0x8 | Valid | Valid | Valid | Do not use |
| 0x9 | Valid | Valid | Valid | Do not use |
| 0xA | Valid | Valid | Valid | Do not use |

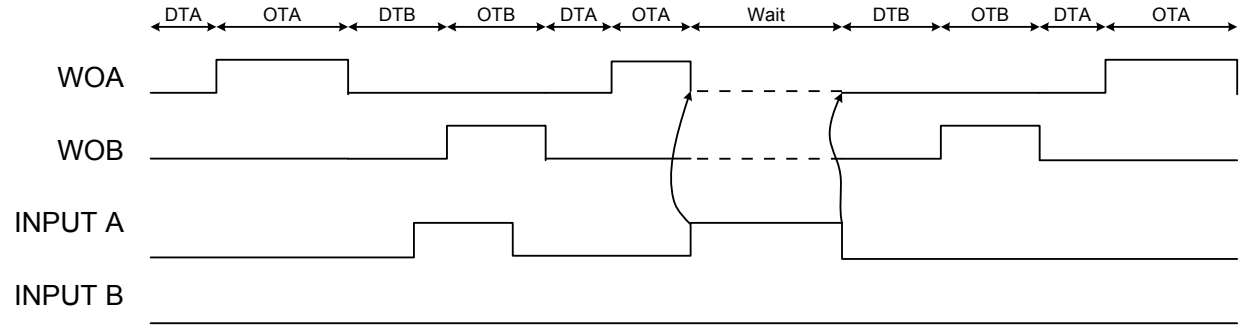
In the following sections the different input modes are presented in detail.

***Input Mode 1: Stop Output, Jump to Opposite Compare Cycle and Wait***

An input Event in Input mode 1 will stop the output signal, jump to the opposite dead-time, and wait until the input event goes low before the TCD counter continues.

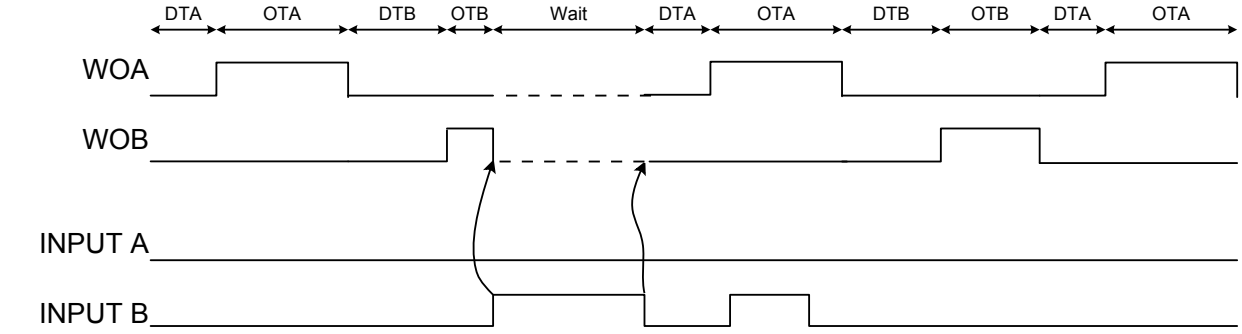
If Input mode 1 is used on input A, an Event will only have an effect if the TCD is in Dead time A or On-time A, and it will only effect the output WOA. When the Event is done, the TCD counter starts at Dead time B.

**Figure 22-10. Input Mode 1 on Input A**




If Input mode 1 is used on input B, an Event will only have an effect if the TCD is in Dead time B or On-time B, and it will only effect the output WOB. When the event is done, the TCD counter starts at Dead time A.

**Figure 22-11. Input Mode 1 on Input B**

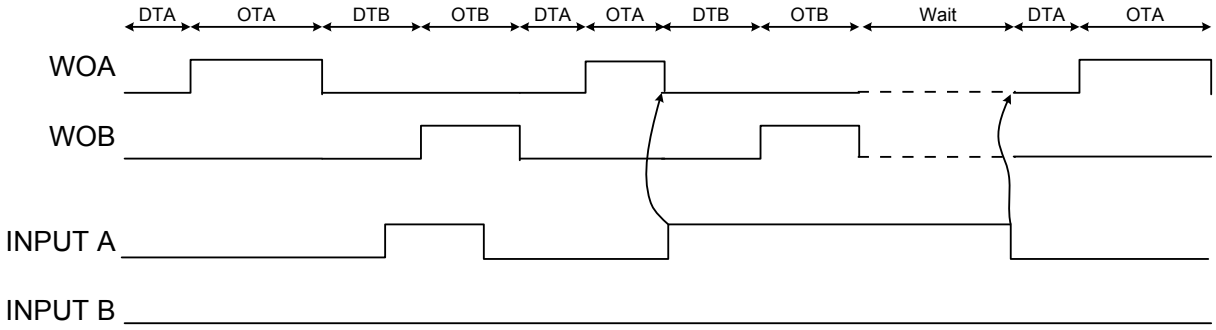



***Input Mode 2: Stop Output, Execute Opposite Compare Cycle and Wait***

An input Event in Input mode 2 will stop the output signal, execute to the opposite dead-time and on-time, then wait until the input Event goes low before the TCD counter continues. If the input is done before the opposite dead-time and on-time have finished, there will be no waiting, but the opposite dead-time and on-time will continue.

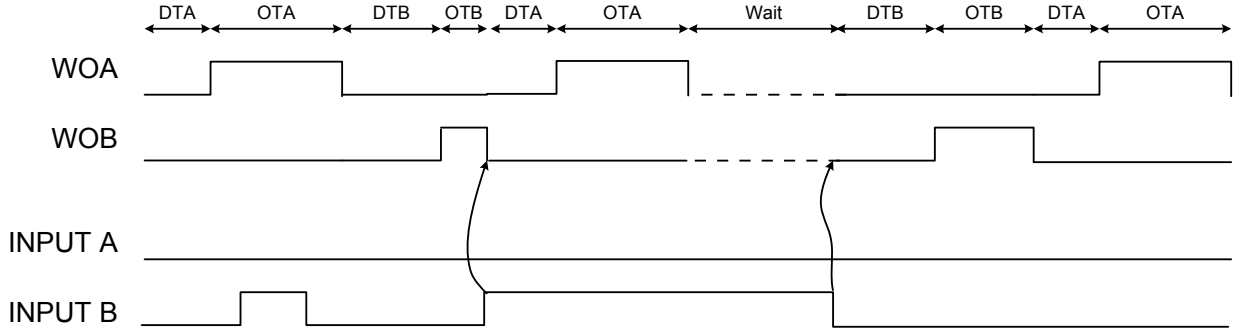
If Input mode 2 is used on input A, an Event will only have an effect if the TCD is in Dead time A or On-time A, and it will only effect the output WOA.

**Figure 22-12. Input Mode 2 on Input A**




If Input mode 2 is used on input B, an Event will only have an effect if the TCD is in Dead time B or On-time B, and it will only effect the output WOB.

**Figure 22-13. Input Mode 2 on Input B**

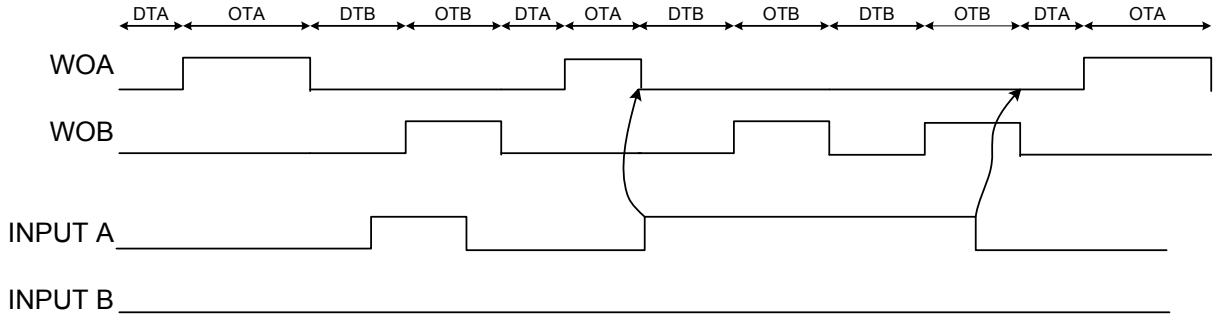



***Input Mode 3: Stop Output, Execute Opposite Compare Cycle while Fault Active***

An input Event in Input mode 3 will stop the output signal and start executing the opposite dead-time and on-time, as long as the fault/input is active. When the input is released, the ongoing dead time and/or on-time will finish and then, normal flow will start.

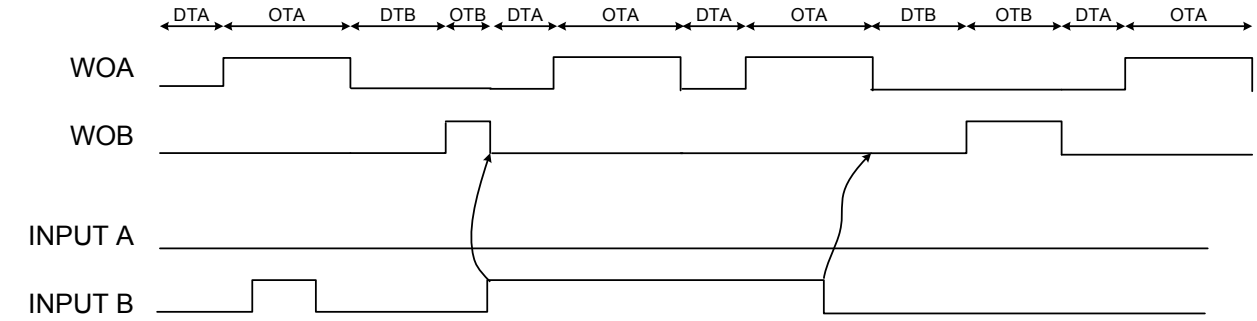
If Input mode 3 is used on input A, an Event will only have an effect if the TCD is in Dead time A or On-time A.

**Figure 22-14. Input Mode 3 on Input A**




If Input mode 3 is used on input B, an Event will only have an effect if the TCD is in Dead time B or On-time B.

**Figure 22-15. Input Mode 3 on Input B**

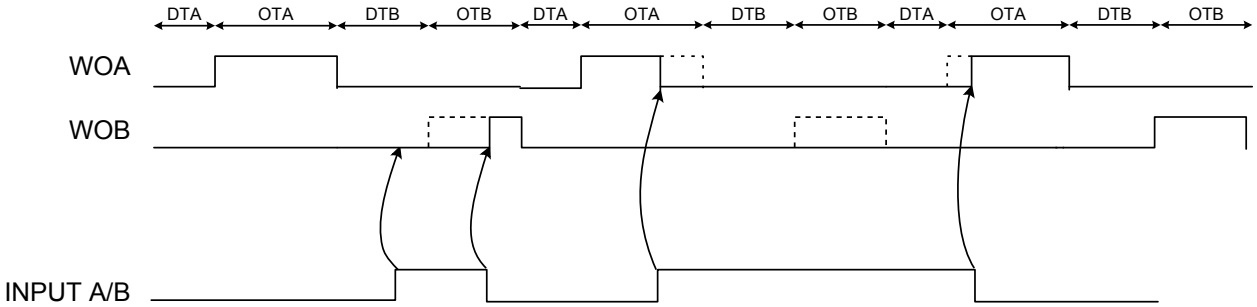



***Input Mode 4: Stop all Outputs, Maintain Frequency***

When input mode 4 is used, both input A and input B will give the same functionality.

An input Event will deactivate the outputs as long as the Event is active. TheTCD counter will not be affected by Events in this input mode.

**Figure 22-16. Input Mode 4**

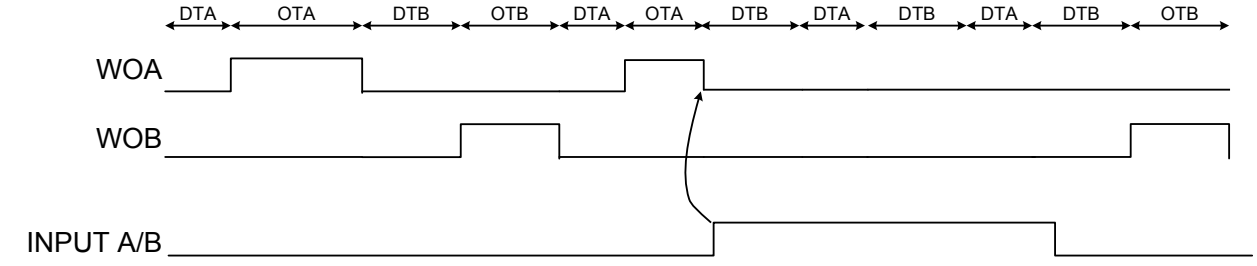



***Input Mode 5: Stop all Outputs, Execute Dead Time while Fault Active***

When input mode 5 is used, both input A and input B give the same functionality:

The input Event stops the outputs and starts on the opposite dead time if it occurs during an on-time. If the Event occurs during a dead time, it will continue until the next on-time should start, but instead it will jump to the next dead time. As long as the input Event is active, alternating dead times will occur. When the input Event stops, the ongoing dead time will finish and the next on-time will continue in the normal flow.

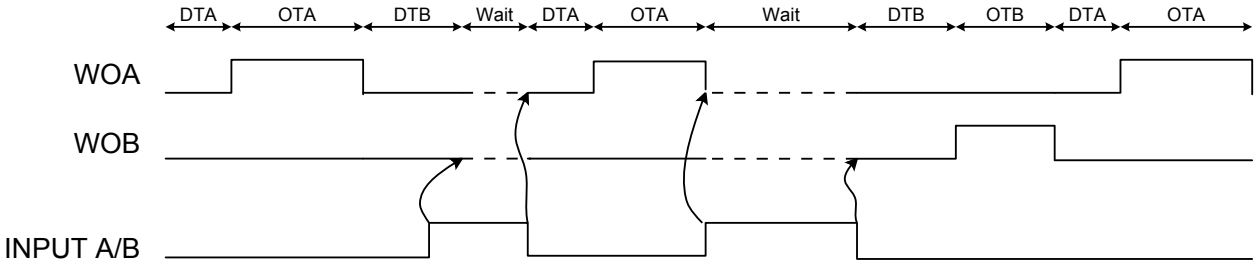
**Figure 22-17. Input Mode 5**




***Input Mode 6: Stop all Outputs, Jump to next Compare Cycle and Wait***

When input mode 6 is used both input A and input B will give the same functionality. The input Event stops the outputs and jumps to the opposite dead time if it occurs during a on time. If the Event occurs during a dead time, it will continue until the next on-time should start, but instead, it will jump to the next dead time. As long as the input Event is active, the TCD counter will wait. When the input Events stops, the next dead time will start and normal flow will continue.

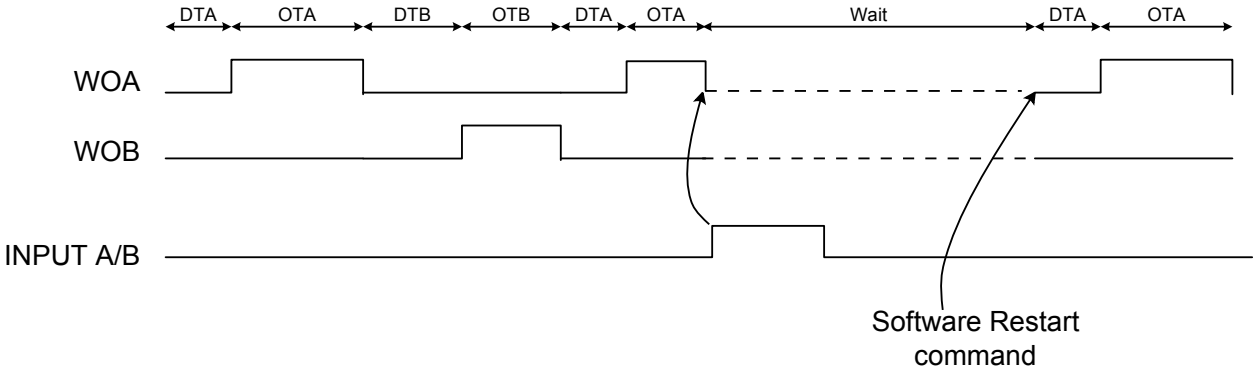
**Figure 22-18. Input Mode 6**




***Input Mode 7: Stop all Outputs, Wait for Software Action***

When input mode 7 is used, both input A and input B will give the same functionality. The input Events stops the outputs and TCD counter. It will be stopped until a Restart command is given. If the input Event still is high when the Restart command is given, it will just stop again. When the TCD counter restarts, it will always start on Dead time A.

**Figure 22-19. Input Mode 7**

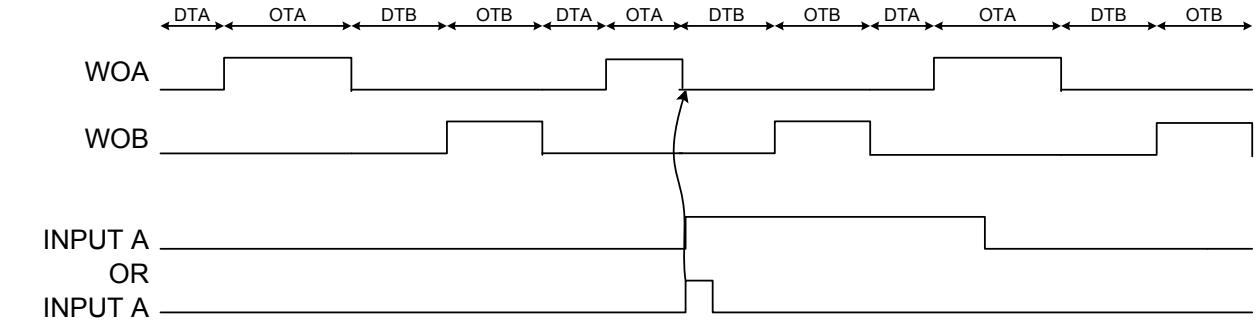



***Input Mode 8: Stop Output on Edge, Jump to Next Compare Cycle***

In Input mode 8, a positive edge on the input Event while the corresponding output is on will cause the output to stop and the TCD counter jump to the opposite Dead time.

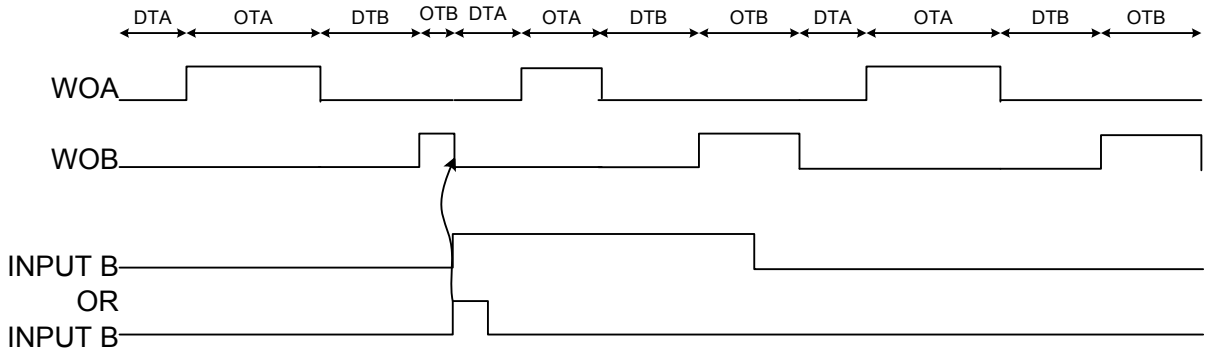
If Input mode 8 is used on input A and an positive input event occurs while in On time A, the TCD counter jumps to Dead time B.

**Figure 22-20. Input Mode 8 on Input A**




If Input mode 8 is used on input B and an positive input event occurs while in On time B, the TCD counter jumps to Dead time A.

**Figure 22-21. Input Mode 8 on Input B**

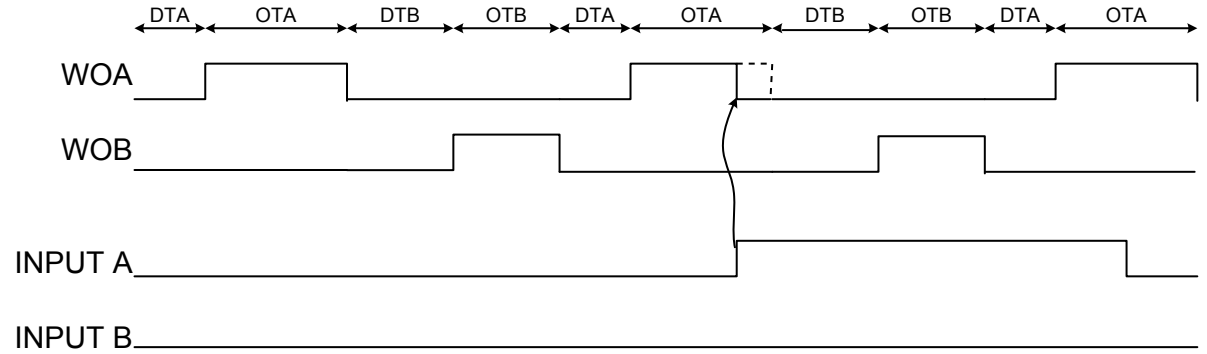



***Input Mode 9: Stop Output at Level, Maintain Frequency***

In Input mode 9 a positive edge on the input Event while the corresponding output is on will cause the output to stop during the rest of the on-time. The TCD counter will not be affected by the Event, only the output.

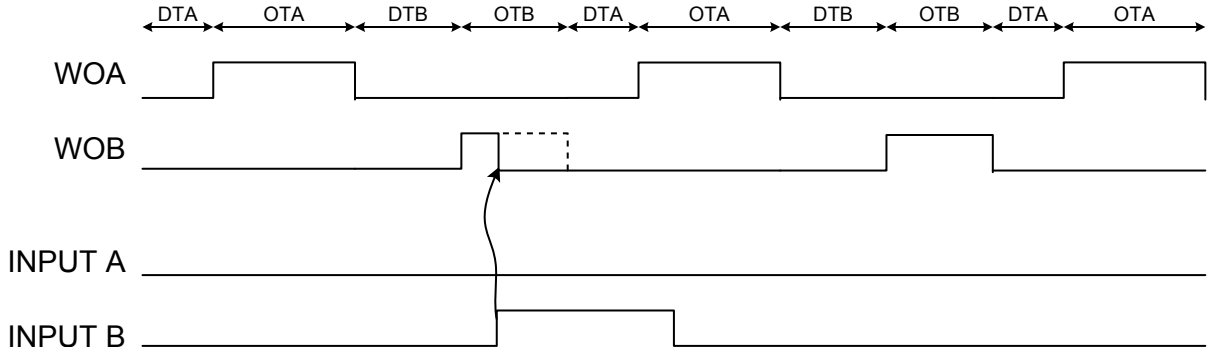
If Input mode 9 is used on input A and an positive input Event occurs while in On time A, the output will be off for the rest of the on-time.

**Figure 22-22. Input Mode 9 on Input A**




If Input mode 9 is used on input B and an positive input event occurs while in On time B, the output will be off for the rest of the on-time.

**Figure 22-23. Input Mode 9 on Input B**

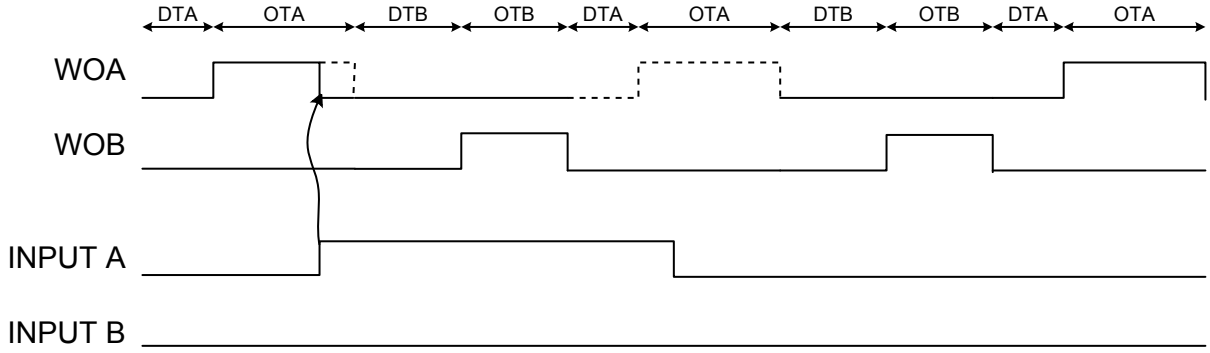



***Input Mode 10: Stop Output on Edge, Maintain Frequency***

In Input mode 10 the input Event will cause the corresponding output to stop as long as the input is active. If the input goes low while there should have been an on-time on the corresponding output, the output will be deactivated for the rest of the on-time, too. The TCD counter is not affected by the Event, only the output.

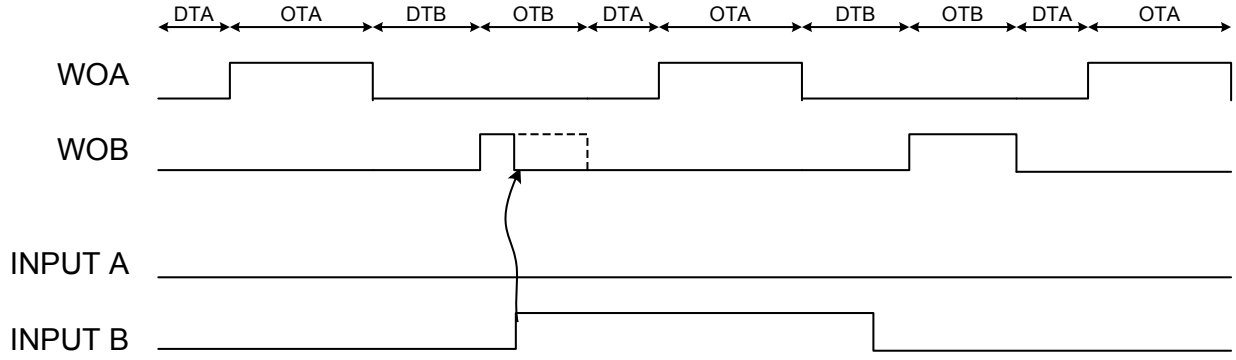
If Input mode 10 is used on input A and an input event occurs, the WOA will be off as long as the event lasts. If released during an on-time, it will be off for the rest of the on-time, too.

**Figure 22-24. Input Mode 10 on Input A**




If Input mode 10 is used on input B and an input Event occurs, the WOB will be off as long as the Event lasts. If released during an on-time, it will be off for the rest of the on-time, too.

**Figure 22-25. Input Mode 10 on Input B**




***Input Mode Summary***

The table below summarizes the conditions as illustrated in the timing diagrams of the preceding sections.

**Table 22-6. Input mode summary**

| INPUTMODE | Trigger → Output affected | Fault On/Active | Fault Release/Inactive |
|---|---|---|---|
| 0x00 | - | No action | No action |
| 0x01 | Input A→WOA | End current on-time and wait | Start with dead-time for other compare |
| | Input B→WOB | | |
| 0x02 | Input A→WOA | End current on-time. | Start with dead-time for current compare |
| | Input B→WOB | Execute other compare cycle and wait | |
| 0x03 | Input A→WOA | End current on-time. | Re-enable current compare cycle |
| | Input B→WOB | Execute other compare cycle | |
| 0x04 | Input A→{WOA, WOB} | Deactivate outputs | |
| | Input B→{WOA, WOB} | | |

| INPUTMODE | Trigger → Output affected | Fault On/Active | Fault Release/Inactive |
|---|---|---|---|
| 0x05 | Input A→{WOA, WOB} | Execute dead-time only | |
| | Input B→{WOA, WOB} | | |
| 0x06 | Input A→{WOA, WOB} | End on-time and wait | Start with dead-time for other compare |
| | Input B→{WOA, WOB} | | |
| 0x07 | Input A→{WOA, WOB} | End on-time and wait for software action | Start with dead-time for current compare |
| | Input B→{WOA, WOB} | | |
| 0x08 | Input A→WOA | End current on-time and continue with other off-time | |
| | Input B→WOB | | |
| 0x09 | Input A→WOA | Block current on-time and continue sequence | |
| | Input B→WOB | | |
| 0x0A | Input A→WOA | Deactivate on-time until end of sequence while trigger is active | |
| | Input B→WOB | | |
| other | - | - | - |

#### 22.3.2.5 Dithering

If it is not possible to achieve the desired frequency because of pre-scaler/period selection limitations, Dithering can be used to approximate the desired frequency and reduce waveform drift.

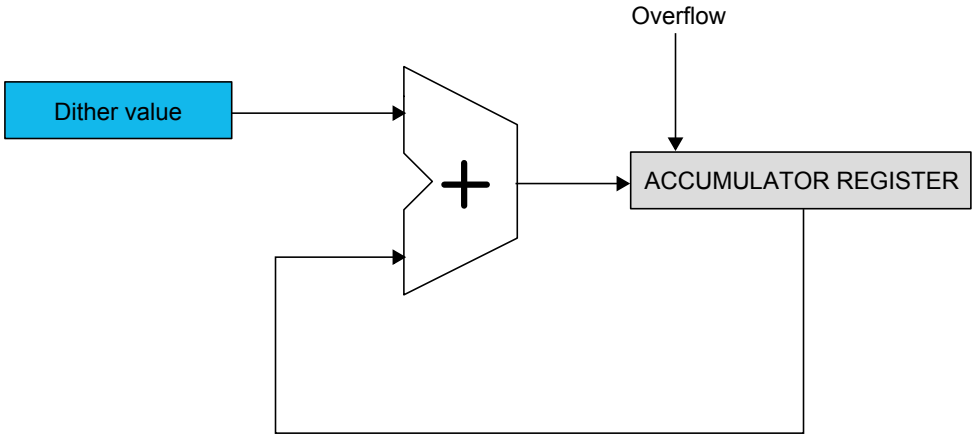
Dither accumulates the fractional error of the counter clock for each cycle. When the fractional error overflows, an additional cycle is added to the selected part of the cycle.

**Example**

If the timer clock frequency is 10MHz, it will give the timer a resolution of 100ns. Then, the output frequency should be 75 kHz. Here 75 kHz means a period of 13333ns, and that is not possible to achieve with a constant period with a 100ns resolution because it equals 133.33 cycles. The output period can be set to either 133 cycles (75.188kHz) or 134 cycles (74.626 kHz).

It is possible to change the period between the two frequencies manually in the firmware to get an average output frequency of 75 kHz. (change the every 3rd period to 134 cycles) The dither can do this automatically by accumulating the error (0.33 cycles). The accumulator calculate when the accumulated error is larger than one clock cycle and when that happens it adds an additional cycle to the timer period.

**Figure 22-26. Dither Logic**




The user can select where in the TCD cycle the dither will be added by writing to the Dither Selection bits in the Dither Control register (DITHERSEL in TCD.DITCTRL):

- On-time B
- On-time A and B
- Dead-time B
- Dead-time A and B

How much the dithering will affect the TCD cycle time depends on what Waveform Generation Mode is used, see table below.

Dithering is not supported in Dual Slope Mode.

**Table 22-7. Mode-Dependent Dithering Additions to TCD Cycle**

| WAVEGEN | DITHERSEL in TCD.DITCTRL | Additional TCD clock cycles to TCD cycle |
|---|---|---|
| One Ramp Mode | On-time B | 1 |
| | On-time A and B | 1 |
| | Dead-time B | 0 |
| | Dead-time A and B | 0 |
| Two Ramp Mode | On-time B | 1 |
| | On-time A and B | 2 |
| | Dead-time B | 0 |
| | Dead-time A and B | 0 |
| Four Ramp Mode | On-time B | 1 |
| | On-time A and B | 2 |
| | Dead-time B | 1 |
| | Dead-time A and B | 2 |
| Dual Slope Mode | On-time B | 0 (not supported) |
| | On-time A and B | 0 (not supported) |

© 2017 Microchip Technology Inc.

| WAVEGEN | DITHERSEL in TCD.DITCTRL | Additional TCD clock cycles to TCD cycle |
|---|---|---|
| | Dead-time B | 0 (not supported) |
| | Dead-time A and B | 0 (not supported) |

The differences in the number of TCD clock cycles added to the TCD cycle is caused by the different number of compare values used by the TCD cycle. For example in One Ramp Mode, only CMPBCLR affects the TCD cycle time.

For DITHERSEL configurations where no extra cycles are added to the TCD cycles, compensation is reached by shortening the following output state.

> **Example:**
>
> In One Ramp Mode with DITHERSEL selecting Dead-time B, the Dead-time B will be increased by one cycle when dither overflow occurs. This reduces On-time B by one cycle.

#### 22.3.2.6 TCD Counter Capture

Because the TCD counter is asynchronous to the system clock it is not possible to read out the counter value directly. It is possible to capture the TCD counter value, synchronized to the IO clock domain in two different ways.

- Capture value on input Events
- Software capture

The capture logic contains two separate capture blocks, CAPTUREA and CAPTUREB, that can capture and synchronize the TCD counter value to the IO clock domain. CAPTUREA/B can be triggered by input Event A/B or by software.

The capture values can be read by reading first TCD.CAPTUREXL and then TCD.CAPTUREXH registers.

**Captures Triggered by Input Events**

To enable capture on input Event, write a '1' to the ACTION bit in the respective Event Control x register (TCD.EVCTRL) when configuring an Event input.

When a capture has occurred, the TRIGA/B flag is raised in the Interrupt Flags register (TCD.INTFLAGS). The according TRIGA/B interrupt is executed if enabled by writing a '1' to the respective Trigger Interrupt x Enable bit (TRIGx) in the Interrupt Control register (TCD.INTCTRL). By polling TRIGx in TCD.INTFLAGS, the user knows that a CAPTUREx value is available, and can read out the value by reading first the TCD.CAPTUREXL and then TCD.CAPTURXH registers.

> **Example**
>
> In order to do PWM capture both event A and B should be connected to the same asynchronous event channel that contains the PWM signal. To get information on the PWM signal one event input should be configured to capture the rising edge of the signal. The other event input should be configured to capture the falling edge of the signal.

**Capture Triggered by Software**

Software can capture the TCD value by writing a '1' to respective Software Capture A/B Strobe bit (SCAPTUREx) in the Control E register (TCD.CTRLE). When this command is executed and the Command Ready bit (CMDRDY) in the Status register (TCD.STATUS) reads '1' again, the CAPTUREA/B

value is available. It can now be read by reading first the TCD.CAPTUREXL and then the TCD.CAPTURxH registers.

**Using Capture together with input modes**

The capture functionality can be used together with input modes. The same event will then both capture the counter value and trigger a change in the counter flow depending on the input mode selected.

> **Example**
>
> If the user what to reset the counter when running in One Ramp mode whenever the input event capture the counter value this can be done by using input event B and setting INPUTCTRLB register to 0x08.

**Related Links**

Initialization and Disabling

#### 22.3.2.7 Output Control

The outputs are configured by writing to the Fault Control register (TCD.FAULTCTRL). TCD.FAULTCTRL is only reset to zero after a POR reset. During the reset sequence after any Reset, TCD.FAULTCTRL will get its values from the TCD Fuse (FUSE.TCDCFG).

The Compare x Enable bits (CMPxEN in TCD.FAULTCTRL) enable the different outputs. The CMPx bits in TCD.FAULTCTRL set the value the registers should have after Reset or when a fault is triggered.

The TCD itself generates two different outputs, WOA and WOB.

The user can also override the outputs based on the TCD counter state by writing a '1' to the Compare Output Value Override bit in the Control C register (CMPOVR in TCD.CTRLC). The user can then select the output values in the different dead- and on times by writing to the Compare x Value bit fields in the Control D register (CMPAVAL and CMPBVAL in TCD.CTRLD).

When used in One Ramp mode, WOA will only use the setup for Dead Time A (DTA) and On Time A (OTA) to set the output. WOB will only use Dead-Time B (DTB) and On Time B (OTB) values to set the output.

When using the override feature together with faults detection (input modes) the CMPA bit in TCD.FAULTCTRL should be equal to CMPAVAL[0] and [2] in TCD.CTRLD. if not the first cycle after a fault is detected can have the wrong polarity on the outputs. The same applies to CMPB in TCD.FAULTCTRL bit, which should be equal to CMPBVAL[0] and [2] in TCD_CTRLD.

Due to the asynchronous nature of the TCD, that input events immediately can affect the output signal, there is a risk of nano-second spikes occurring on the output when there is no load on the pin. The case occurs in any input mode different from 0 and when an input event is triggering. The spike value will always be in direction of the CMPx value given by the FAULTCTRL register.

**Related Links**

TCD0CFG

### 22.3.3 Events

The TCD can generate the following output events:

- TCD counter matches CMPBCLR
- TCD counter matches CMPASET
- TCD counter matches CMPBSET

- Programmable TCD output event. The user can select trigger, and all the different compare matches. In addition it is possible to delay the output Event from 0 to 256 TCD delay cycles.

The three events based on the counter match directly generate event strobes that last one clock cycle on the TCD counter clock. The programmable output event generates a event strobe that last one clock cycle on the TCD synchronizer clock.

The TCD has the possibility to receive these input Events:

- Input A
- Input B

**Related Links**

TCD Inputs

EVSYS - Event System

#### 22.3.3.1 Programmable output events

Programmable output event uses the same logic as the Input blanking for trigger selection and delay. It is therefore not possible to configure the functionalities independently. If the input blanking functionality is used, the output event cannot be delayed and the trigger used for input blanking will also be used for the output event.

The programmable output events are controlled by the DLYCTRL and DLYVAL registers. It is possible to delay the output event by 0 to 256 TCD delay clock cycles if the DLYTRIG bits in DLYCTRL is set to 0x2. The delayed output event functionality uses the TCD delay clock and counts until the DLYVAL value is reached before the trigger is sent out as a event. The TCD delay clock is a prescaled version of the TCD synchronization clock and the division factor is set by the DLYPRESC bits in the DLYCTRL register. The output event will be delayed by TCD clock period x DLYPRESC division factor x DLYVAL.

### 22.3.4 Interrupts

**Table 22-8. Available Interrupt Vectors and Sources**

| Offset | Name | Vector Description | Conditions |
|---|---|---|---|
| 0x00 | OVF | Overflow interrupt | The TCD is done with one TCD cycle. |
| 0x02 | TRIG | Trigger interrupt | • TRIGA: Counter is entering On-Time A<br>• TRIGB: Counter is entering On-Time B |

When an interrupt condition occurs, the corresponding Interrupt Flag is set in the Interrupt Flags register of the peripheral (*peripheral*.INTFLAGS).

An interrupt source is enabled or disabled by writing to the corresponding enable bit in the peripheral's Interrupt Control register (*peripheral*.INTCTRL).

An interrupt request is generated when the corresponding interrupt source is enabled and the Interrupt Flag is set. The interrupt request remains active until the Interrupt Flag is cleared. See the peripheral's INTFLAGS register for details on how to clear Interrupt Flags.

When several interrupt request conditions are supported by an interrupt vector, the interrupt requests are ORed together into one combined interrupt request to the Interrupt Controller. The user must read the peripheral's INTFLAGS register to determine which of the interrupt conditions are present.

**Related Links**

CPUINT - CPU Interrupt Controller

SREG

### 22.3.5 Sleep Mode Operation

The TCD operates in Idle sleep, and is stopped when entering Standby and Power Down sleep modes.

### 22.3.6 Synchronization

The TCD has two different clock domains and needs to synchronize the communication between the domains. See Initialization section for detail on how the synchronization of values from the IO clock domain to the TCD clock domain is done. See the Capture section for details on how the synchronization of values from the TCD clock domain to the IO clock domain is done.

**Related Links**

Initialization and Disabling

TCD Counter Capture

### 22.3.7 Configuration Change Protection

This peripheral has registers that are under Configuration Change Protection (CCP). In order to write to these, a certain key must be written to the CPU.CCP register first, followed by a write access to the protected bits within four CPU instructions.

It is possible to try writing to these registers any time, but the values are not altered.

The following registers are under CCP:

**Table 22-9. TCD - Registers under Configuration Change Protection**

| Register | Key |
|---|---|
| TCD.FAULTCTRL | IOREG |

## 22.4 Register Summary - TCD

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | | CLKSEL[1:0] | | CNTPRES[1:0] | | SYNCPRES[1:0] | | ENABLE |
| 0x01 | CTRLB | 7:0 | | | | | | | WGMODE[1:0] | |
| 0x02 | CTRLC | 7:0 | CMPDSEL | CMPCSEL | | | FIFTY | | AUPDATE | CMPOVR |
| 0x03 | CTRLD | 7:0 | CMPBVAL[3:0] | | | | CMPAVAL[3:0] | | | |
| 0x04 | CTRLE | 7:0 | DISEOC | | | SCAPTUREB | SCAPTUREA | RESTART | SYNC | SYNCEOC |
| 0x05 ... 0x07 | Reserved | | | | | | | | | |
| 0x08 | EVCTRLA | 7:0 | CFG[1:0] | | | EDGE | | ACTION | | TRIGEI |
| 0x09 | EVCTRLB | 7:0 | CFG[1:0] | | | EDGE | | ACTION | | TRIGEI |
| 0x0A ... 0x0B | Reserved | | | | | | | | | |
| 0x0C | INTCTRL | 7:0 | | | | | TRIGB | TRIGA | | OVF |
| 0x0D | INTFLAGS | 7:0 | | | | | TRIGB | TRIGA | | OVF |
| 0x0E | STATUS | 7:0 | PWMACTB | PWMACTA | | | | | CMDRDY | ENRDY |
| 0x0F | Reserved | | | | | | | | | |
| 0x10 | INPUTCTRLA | 7:0 | | | | | INPUTMODE[3:0] | | | |
| 0x11 | INPUTCTRLB | 7:0 | | | | | INPUTMODE[3:0] | | | |
| 0x12 | FAULTCTRL | 7:0 | CMPDEN | CMPCEN | CMPBEN | CMPAEN | CMPD | CMPC | CMPB | CMPA |
| 0x13 | Reserved | | | | | | | | | |
| 0x14 | DLYCTRL | 7:0 | | | DLYPRESC[1:0] | | DLYTRIG[1:0] | | DLYSEL[1:0] | |
| 0x15 | DLYVAL | 7:0 | DLYVAL[7:0] | | | | | | | |
| 0x16 ... 0x17 | Reserved | | | | | | | | | |
| 0x18 | DITCTRL | 7:0 | | | | | | | DITHERSEL[1:0] | |
| 0x19 | DITVAL | 7:0 | | | | | DITHER[3:0] | | | |
| 0x1A ... 0x1D | Reserved | | | | | | | | | |
| 0x1E | DBGCTRL | 7:0 | | | | | | FAULTDET | | DBGRUN |
| 0x1F ... 0x21 | Reserved | | | | | | | | | |
| 0x22 | CAPTUREA | 7:0 | CAPTURE[7:0] | | | | | | | |
| 0x23 | | 15:8 | | | | | CAPTURE[11:8] | | | |
| 0x24 | CAPTUREB | 7:0 | CAPTURE[7:0] | | | | | | | |
| 0x25 | | 15:8 | | | | | CAPTURE[11:8] | | | |
| 0x26 ... 0x27 | Reserved | | | | | | | | | |
| 0x28 | CMPASET | 7:0 | CMPSET[7:0] | | | | | | | |
| 0x29 | | 15:8 | | | | | CMPSET[11:8] | | | |
| 0x2A | CMPACLR | 7:0 | CMPCLR[7:0] | | | | | | | |

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x2B | | 15:8 | | | | | CMPCLR[11:8] | | | |
| 0x2C | CMPBSET | 7:0 | CMPSET[7:0] | | | | | | | |
| 0x2D | | 15:8 | | | | | CMPSET[11:8] | | | |
| 0x2E | CMPBCLR | 7:0 | CMPCLR[7:0] | | | | | | | |
| 0x2F | | 15:8 | | | | | CMPCLR[11:8] | | | |

## 22.5 Register Description

### 22.5.1 Control A

**Name:** CTRLA
**Offset:** 0x00
**Reset:** 0x00
**Property:** Enable-protected

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | CLKSEL[1:0] | | CNTPRES[1:0] | | SYNCPRES[1:0] | | ENABLE |
| Access | | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bits 6:5 – CLKSEL[1:0]: Clock Select**
The clock select bits select the clock source of the TCD clock.

| Value | Description |
|---|---|
| 0x0 | OSC20M |
| 0x1 | Reserved |
| 0x2 | External clock |
| 0x3 | System clock |

**Bits 4:3 – CNTPRES[1:0]: Counter Prescaler**
The Counter Prescaler bits select the division factor of the TCD counter clock.

| Value | Description |
|---|---|
| 0x0 | Division factor 1 |
| 0x1 | Division factor 4 |
| 0x2 | Division factor 32 |
| 0x3 | Reserved |

**Bits 2:1 – SYNCPRES[1:0]: Synchronization Prescaler**
The synchronization prescaler bits select the division factor of the TCD clock.

| Value | Description |
|---|---|
| 0x0 | Division factor 1 |
| 0x1 | Division factor 2 |
| 0x2 | Division factor 4 |
| 0x3 | Division factor 8 |

**Bit 0 – ENABLE: Enable**
When this bit is written to, it will automatically be synchronized to the TCD clock domain.

This bit can be changed as long as synchronization of this bit is not ongoing, see Enable Ready bit (ENRDY) in Status register (STATUS).

This bit is not enable-protected.

| Value | Description |
|---|---|
| 0 | The TCD is disabled. |
| 1 | The TCD is enabled and running. |

### 22.5.2 Control B

**Name:** CTRLB
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

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

**Bits 1:0 – WGMODE[1:0]: Waveform Generation Mode**
These bits select the waveform generation

| Value | Name | Description |
|---|---|---|
| 0x0 | ONERAMP | One ramp mode |
| 0x1 | TWORAMP | Two ramp mode |
| 0x2 | FOURRAMP | Four ramp mode |
| 0x3 | DS | Dual-slope mode |

### 22.5.3 Control C

**Name:** CTRLC
**Offset:** 0x02
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CMPDSEL | CMPCSEL | | | FIFTY | | AUPDATE | CMPOVR |
| Access | R/W | R/W | | | R/W | | R/W | R/W |
| Reset | 0 | 0 | | | 0 | | 0 | 0 |

**Bit 7 – CMPDSEL: Compare D Output Select**

| Value | Name | Description |
|---|---|---|
| 0 | PWMA | Waveform A |
| 1 | PWMB | Waveform B |

**Bit 6 – CMPCSEL: Compare C Output Select**

| Value | Name | Description |
|---|---|---|
| 0 | PWMA | Waveform A |
| 1 | PWMB | Waveform B |

**Bit 3 – FIFTY: Fifty Percent Waveform**
If the two waveforms have identical characteristics, this bit can be written to '1'. This will cause any values written to register CMPBSET/CLR also to be written to the register CMPASET/CLR.

**Bit 1 – AUPDATE: Automatically Update**
If this bit is set a synchronization at the end of the TCD cycle is automatically requested after the Compare B Clear High register (CMPBCLRH) is written.

If the Fifty Percent Waveform is enabled by setting the FIFTY bit in this register, writing the Compare A Clear High register will also request a synchronization at the end of the TCD cycle if the AUPDATE bit is set.

**Bit 0 – CMPOVR: Compare Output Value Override**
When this bit is written, default values of the Waveform Outputs A and B are overridden by the values written in the Compare x Value in active state bit fields in the Control D register (CTRLD.CMPnxVAL). See the Control D register description for more details.

### 22.5.4 Control D

**Name:** CTRLD
**Offset:** 0x03
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CMPBVAL[3:0] | | | | CMPAVAL[3:0] | | | |
| 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 0:3, 4:7 – CMPAVAL, CMPBVAL: Compare x Value (in active state)**
These bits sets the active state for the different ramps for compare x.

These settings are only valid if the Compare Output Value Override bit in the Control C register (CTRLC.CMPOVR) is written to '1'.

| CMPxVAL | A_off | A_on | B_off | B_on |
|---|---|---|---|---|
| PWMA | CMPAVAL[0] | CMPAVAL[1] | CMPAVAL[2] | CMPAVAL[3] |
| PWMB | CMPBVAL[0] | CMPBVAL[1] | CMPBVAL[2] | CMPBVAL[3] |

In One Ramp mode, PWMA will only use A_off and A_on values and PWMB will only use B_off and B_on values. This is due to possible overlap between the values A_off, A_on, B_off and B_on.

### 22.5.5 Control E

**Name:** CTRLE
**Offset:** 0x04
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DISEOC | | | SCAPTUREB | SCAPTUREA | RESTART | SYNC | SYNCEOC |
| Access | R/W | | | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | | | 0 | 0 | 0 | 0 | 0 |

**Bit 7 – DISEOC: Disable at End of TCD Cycle Strobe**
When this bit is written, the TCD will automatically disable at the end of the TCD cycle.

When this bit is written to '1', the ENRDY in TCD.STATUS will keep low until the TCD is disabled.

Writing to this bit only has effect if there is no ongoing synchronization of Enable. See also ENRDY bit in TCD.STATUS.

**Bit 4 – SCAPTUREB: Software Capture B Strobe**
When this bit is written to '1', a software capture to Capture register B (TCD.CAPTUREBL/H) is done as soon as the strobe is synchronized to the TCD domain.

Writing to this bit only has effect if there is no ongoing synchronization of a command. See also CMDRDY bit in TCD.STATUS.

**Bit 3 – SCAPTUREA: Software Capture A Strobe**
When this bit is written to '1', a software capture to Capture register A (TCD.CAPTUREAL/H) is done as soon as the strobe is synchronized to the TCD domain.

Writing to this bit only has effect if there is no ongoing synchronization of a command. See also CMDRDY bit in TCD.STATUS.

**Bit 2 – RESTART: Restart Strobe**
When this bit is written a Restart of the TCD counter is executed as soon as this bit is synchronized to the TCD domain.

Writing to this bit only has effect if there is no ongoing synchronization of a command. See also CMDRDY bit in TCD.STATUS.

**Bit 1 – SYNC: Synchronize Strobe**
When this bit is written to '1' the doubled buffered registers will be loaded to the TCD domain as soon as this bit is synchronized to the TCD domain.

Writing to this bit only has effect if there is no ongoing synchronization of a command. See also CMDRDY bit in TCD.STATUS.

**Bit 0 – SYNCEOC: Synchronize End of TCD Cycle Strobe**
When this bit is written to '1' the doubled buffered registers will be loaded to the TCD domain at the end of the next TCD cycle.

Writing to this bit only has effect if there is no ongoing synchronization of a command. See also CMDRDY bit in TCD.STATUS.

### 22.5.6 Event Control x

**Name:** EVCTRLA, EVCTRLB
**Offset:** 0x08 + n*0x01 [n=0..1]
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CFG[1:0] | | | EDGE | | ACTION | | TRIGEI |
| Access | R/W | R/W | | R/W | | R/W | | R/W |
| Reset | 0 | 0 | | 0 | | 0 | | 0 |

**Bits 7:6 – CFG[1:0]: Event Configuration**
When the Input Capture Noise canceler is activated (FILTERON), the Event input is filtered. The filter function requires four successive equal valued samples of the Retrigger pin for changing its output. The Input Capture is therefore delayed by four clock cycles when the noise canceler is enabled.

When the Asynchronous Event is enabled (ASYNCON), the Event input will qualify the output directly.

| Value | Name | Description |
|---|---|---|
| 0x0 | NEITHER | Neither Filter nor Asynchronous Event is enabled. |
| 0x1 | FILTERON | Input Capture Noise Cancellation Filter enabled. |
| 0x2 | ASYNCON | Asynchronous Event output qualification enabled. |
| other | - | Reserved. |

**Bit 4 – EDGE: Edge Selection**
This bit is used to select the active edge or level for the event input.

| Value | Name | Description |
|---|---|---|
| 0 | FALL_LOW | The falling edge or low level of the Event input generates Retrigger or Fault action. |
| 1 | RISE_HIGH | The rising edge or high level of the Event input generates Retrigger or Fault action. |

**Bit 2 – ACTION: Event Action**
This bit enables Capture on Event input. By default, the input will trigger a Fault, depending on the Input x register input mode (TCD.INPUTx). It is also possible to trigger a Capture on the Event input.

| Value | Name | Description |
|---|---|---|
| 0 | FAULT | Event triggers a Fault. |
| 1 | CAPTURE | Event triggers a Fault and Capture. |

**Bit 0 – TRIGEI: Trigger Event Input Enable**
Writing this bit to '1' enables Event as trigger for input A.

### 22.5.7 Interrupt Control

**Name:** INTCTRL
**Offset:** 0x0C
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | TRIGB | TRIGA | | OVF |
| Access | | | | | R/W | R/W | | R/W |
| Reset | | | | | 0 | 0 | | 0 |

**Bits 2, 3 – TRIGA, TRIGB: Trigger x Interrupt Enable**
Writing this bit to '1' enables executing an interrupt when trigger input x is received.

**Bit 0 – OVF: Counter Overflow**
Writing this bit to '1' enables executing an interrupt at Restart of the sequence or Overflow of the counter.

### 22.5.8 Interrupt Flags

**Name:** INTFLAGS
**Offset:** 0x0D
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | TRIGB | TRIGA | | OVF |
| Access | | | | | R/W | R/W | | R/W |
| Reset | | | | | 0 | 0 | | 0 |

**Bits 2, 3 – TRIGA, TRIGB: Trigger x Interrupt Flag**
When a capture has occurred, the TRIGA/B is raised. This bit is cleared by writing a '1' to it.

**Bit 0 – OVF: Overflow Interrupt Flag**
When a capture is overflow, this flag is raised. This bit is cleared by writing a '1' to it.

### 22.5.9 Status

**Name:** STATUS
**Offset:** 0x0E
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | PWMACTB | PWMACTA | | | | | CMDRDY | ENRDY |
| Access | R/W | R/W | | | | | R | R |
| Reset | 0 | 0 | | | | | 0 | 0 |

**Bits 6, 7 – PWMACTA, PWMACTB: PWM Activity on x**
This bit is set by hardware each time the output WO1 toggles from 0 to 1 or from 1 to 0.

This status bit must be cleared by software by writing a '1' to it before new PWM activity can be detected.

**Bit 1 – CMDRDY: Command Ready**
This status bit tells when a command is synced to the TCD domain and the system is ready to receive new commands.

The following clears the CMDRDY bit:

1. TCD.CTRLE SYNCEOC strobe
2. TCD.CTRLE SYNC strobe
3. TCD.CTRLE RESTART strobe
4. TCD.CTRLE SCAPTUREA Capture A strobe
5. TCD.CTRLE SCAPTUREB Capture B strobe
6. TCD.CTRLC AUPDATE written to '1' and writing to TCD.CMPBCLRH register

**Bit 0 – ENRDY: Enable Ready**
This status bit tells when the ENABLE value in TCD.CTRLA is synced to the TCD domain, and is ready to be written to again.

The following clears the ENRDY bit:

1. Writing to the ENABLE bit in (TCD.CTRLA
2. TCD.CTRLE DISEOC strobe
3. Going to break in an On-Chip Debugging (OCD) session with and the Debug Run bit (DBGCTRL) in TCD.DBGCTRL is not '1'

### 22.5.10 Input Control x

**Name:** INPUTCTRLA, INPUTCTRLB
**Offset:** 0x10 + n*0x01 [n=0..1]
**Reset:** 0x00
**Property:** -

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

**Bits 3:0 – INPUTMODE[3:0]: Input Mode**

| Value | Name | Description |
|---|---|---|
| 0x0 | NONE | Input has no action |
| 0x1 | JMPWAIT | Stop output, jump to opposite compare cycle and wait |
| 0x2 | EXECWAIT | Stop output, execute opposite compare cycle and wait |
| 0x3 | EXECFAULT | Stop output, execute opposite compare cycle while fault active |
| 0x4 | FREQ | Stop all outputs, maintain frequency |
| 0x5 | EXECDT | Stop all outputs, execute dead time while fault active |
| 0x6 | WAIT | Stop all outputs, jump to next compare cycle and wait |
| 0x7 | WAITSW | Stop all outputs, wait for software action |
| 0x8 | EDGETRIG | Stop output on edge, jump to next compare cycle |
| 0x9 | EDGETRIGFREQ | Stop output on edge, maintain frequency |
| 0xA | LVLTRIGFREQ | Stop output at level, maintain frequency |

### 22.5.11 Fault Control

**Name:** FAULTCTRL
**Offset:** 0x12
**Reset:** Loaded from fuse
**Property:** Configuration Change Protection

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CMPDEN | CMPCEN | CMPBEN | CMPAEN | CMPD | CMPC | CMPB | CMPA |
| 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 4, 5, 6, 7 – CMPAEN, CMPBEN, CMPCEN, CMPDEN: Compare x Enable**
These bits enable Compare as output on pin. At reset the content is kept and during the reset sequence loaded from the TCD Configuration Fuse (FUSE.TCDFG).

**Bits 0, 1, 2, 3 – CMPA, CMPB, CMPC, CMPD: Compare Value x**
These bits set the default state from Reset, or when a input Event triggers a Fault causing changes to the output. At reset the content is kept and during the reset sequence loaded from the TCD Configuration Fuse (FUSE.TCDFG).

### 22.5.12 Delay Control

**Name:** DLYCTRL
**Offset:** 0x14
**Reset:** 0x00
**Property:** -

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

**Bits 5:4 – DLYPRESC[1:0]: Delay Prescaler**
These bits control the prescaler settings for the blanking or output event delay.

| Value | Description |
|---|---|
| 0x0 | Prescaler division factor 1 |
| 0x1 | Prescaler division factor 2 |
| 0x2 | Prescaler division factor 4 |
| 0x3 | Prescaler division factor 8 |

**Bits 3:2 – DLYTRIG[1:0]: Delay Trigger**
These bits control what should trigger the blanking or output event delay.

| Value | Name | Description |
|---|---|---|
| 0x0 | CMPASET | CMPASET triggers delay |
| 0x1 | CMPACLR | CMPACLR triggers delay |
| 0x2 | CMPBSET | CMPBSET triggers delay |
| 0x3 | CMPBCLR | CMPASET triggers delay (end of cycle) |

**Bits 1:0 – DLYSEL[1:0]: Delay Select**
These bits control what function should be used by the delay trigger the blanking or output event delay.

| Value | Description |
|---|---|
| 0x0 | Delay functionality not used |
| 0x1 | Input blanking enabled |
| 0x2 | Event delay enabled |
| 0x3 | Reserved |

### 22.5.13 Delay Value

**Name:** DLYVAL
**Offset:** 0x15
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DLYVAL[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 – DLYVAL[7:0]: Delay Value**
These bits configure the blanking/ output event delay time or event output synchronization delay in number of prescaled TCD cycles.

### 22.5.14 Dither Control

**Name:** DITCTRL
**Offset:** 0x18
**Reset:** 0x00
**Property:** -

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

**Bits 1:0 – DITHERSEL[1:0]: Dither Select**
These bits select which Compare register is using the dither function. See also Dithering.

| Value | Name | Description |
|---|---|---|
| 0x0 | ONTIMEB | On-time ramp B |
| 0x1 | ONTIMEAB | On-time ramp A and B |
| 0x2 | DEADTIMEB | Dead-time ramp B |
| 0x3 | DEADTIMEAB | Dead-time ramp A and B |

### 22.5.15 Dither Value

**Name:** DITVAL
**Offset:** 0x19
**Reset:** 0x00
**Property:** -

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

**Bits 3:0 – DITHER[3:0]: Dither Value**
These bits configure the fractional adjustment of the on-time or off-time according to Dither Selection bits (DITHERSEL) in the Dither Control register (TCD.DITCTRL). The DITHER value is added to a 4-bit accumulator at the end of each TCD cycle. When the accumulator overflows the frequency adjustment will occur.

The DITHER bits are doubled buffered so the new value is copied in at an update condition.

### 22.5.16 Debug Control

**Name:** DBGCTRL
**Offset:** 0x1E
**Reset:** 0x00
**Property:** -

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

**Bit 2 – FAULTDET: Fault Detection**
This bit defines how the peripheral behaves when stopped in Debug Mode.

| Value | Name | Description |
|---|---|---|
| 0 | NONE | No fault is generated if TCD is stopped in debug mode. |
| 1 | FAULT | A fault is generated and both trigger flags are set if TCD is halted in debug mode. |

**Bit 0 – DBGRUN: Debug Run**

| Value | Description |
|---|---|
| 0 | The peripheral is halted in break debug mode and ignores events. |
| 1 | The peripheral will continue to run in break debug mode when the CPU is halted. |

### 22.5.17 Capture x

For capture operation, these registers constitute the second buffer level and access point for the CPU. The TCD.CAPTUREx registers are updated with the buffer value when an UPDATE condition occurs. CAPTURE A register contains the value from the TCD counter when a Trigger A or a software capture A occurs. CAPTURE B register contain the value from the TCD counter when Trigger B or software capture B occurs.

The TCD counter value is synchronized to CAPTUREx by either software or an event.

The capture register is blocked for update of new capture data until TCDn.CAPTURExH is read.

**Name:** CAPTUREA, CAPTUREB
**Offset:** 0x22 + n*0x02 [n=0..1]
**Reset:** 0x00
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | | | | | CAPTURE[11:8] | | | |
| Access | | | | | R | R | R | R |
| Reset | | | | | 0 | 0 | 0 | 0 |

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

**Bits 11:0 – CAPTURE[11:0]: Capture Byte**

### 22.5.18 Compare Set x

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

**Name:** CMPASET, CMPBSET
**Offset:** 0x28 + n*0x04 [n=0..1]
**Reset:** 0x00
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | | | | | CMPSET[11:8] | | | |
| Access | | | | | R/W | R/W | R/W | R/W |
| Reset | | | | | 0 | 0 | 0 | 0 |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CMPSET[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 11:0 – CMPSET[11:0]: Compare Set**
These bits hold value of the compare register.

### 22.5.19 Compare Clear x

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

**Name:** CMPACLR, CMPBCLR
**Offset:** 0x2A + n*0x04 [n=0..1]
**Reset:** 0x00
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | | | | | CMPCLR[11:8] | | | |
| Access | | | | | R/W | R/W | R/W | R/W |
| Reset | | | | | 0 | 0 | 0 | 0 |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CMPCLR[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 11:0 – CMPCLR[11:0]: Compare x Clear**
These bits hold the value of the compare register.

# 23. RTC - Real Time Counter

## 23.1 Features

- 16-bit resolution
- Selectable clock source
  - 32.768kHz external crystal (XOSC32K)
  - External clock
  - 32KHz internal ULP oscillator (OSCULP32K)
  - OSCULP32K divided by 32
- Programmable 15-bit clock prescaling
- One compare register
- One period register
- Clear timer on period overflow
- Optional interrupt/Event on overflow and compare match
- Periodic interrupt and Event

## 23.2 Overview

The RTC peripheral offers two timing functions: the Real-Time Counter (RTC) and a Periodic Interrupt Timer (PIT).

The PIT functionality can be enabled independent of the RTC functionality.

**RTC - Real-Time Counter**

The RTC counts (prescaled) clock cycles in a Counter register, and compares the content of the Counter register to a Period register and a Compare register.

The RTC can generate both interrupts and Events on compare match or overflow. It will generate a compare interrupt and/or Event at the first count after the counter equals the Compare register value, and an overflow interrupt and/or Event at the first count after the counter value equals the Period register value. The overflow will also reset the Counter value to zero.

The RTC peripheral typically runs continuously, including in low-power sleep modes, to keep track of time. It can wake up the device from sleep modes and/or interrupt the device at regular intervals.

The reference clock is typically the 32.768kHz output from an external crystal. The RTC can also be clocked from an external clock signal, the 32KHz internal Ultra-Low Power oscillator (OSCULP32K), or the OSCULP32K divided by 32.

The RTC peripheral includes a 15-bit programmable prescaler that can scale down the reference clock before it reaches the counter. A wide range of resolutions and time-out periods can be configured for the RTC. With a 32.768kHz clock source, the maximum resolution is 30.5µs, and time-out periods can be up to 2 seconds. With a resolution of 1s, the maximum time-out period is more than 18 hours (65536 seconds). The RTC can give a compare interrupt and/or Event when the counter equals the compare register value, and an overflow interrupt and/or Event when it equals the period register value.

© 2017 Microchip Technology Inc.

**PIT - Periodic Interrupt Timer**

Using the same clock source as the RTC function, the PIT can request an interrupt or trigger an output Event on every *n*-th clock period. *n* can be selected from {4, 8, 16,.. 32768} for interrupts, and from {64, 128, 256,... 8192} for Events.

The PIT uses the same clock source (CLK_RTC) as the RTC function .

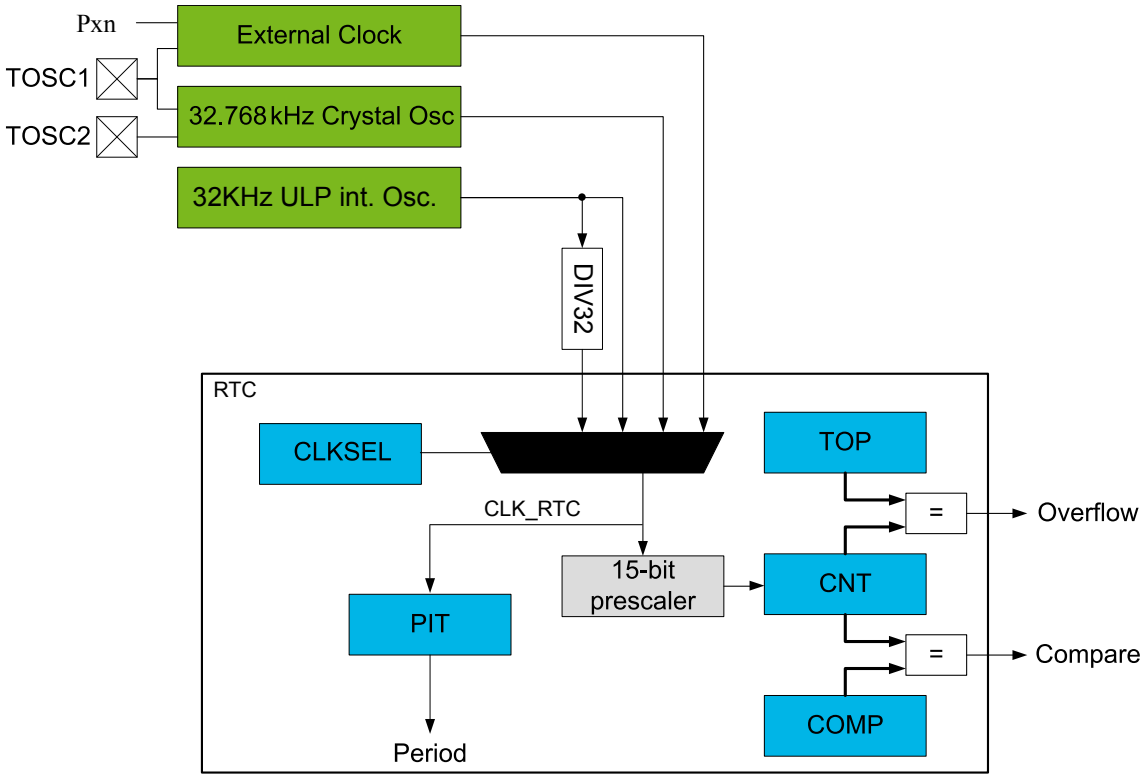
**Related Links**

RTC Functional Description

PIT Functional Description

### 23.2.1 Block Diagram

**Figure 23-1. Block Diagram**




### 23.2.2 Signal Description

Not applicable.

### 23.2.3 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 23-1. RTC System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | Yes | PORT |
| Interrupts | Yes | CPUINT |

| Dependency | Applicable | Peripheral |
|---|---|---|
| Events | Yes | EVSYS |
| Debug | Yes | UPDI |

**Related Links**

Clocks

I/O Lines and Connections

Debug Operation

Interrupts

Events

#### 23.2.3.1 Clocks

System clock (CLK_PER) is required to be at least four times faster than RTC clock (CLK_RTC) for reading counter value, and this is regardless of the RTC_PRESC setting.

**Related Links**

CLKCTRL - Clock Controller

#### 23.2.3.2 I/O Lines and Connections

A 32.768kHz crystal can be connected to the TOSC1 or TOSC2 pins, along with any required load capacitors.

An external clock can be used on the TOSC1 pin.

**Related Links**

CLKCTRL - Clock Controller

#### 23.2.3.3 Interrupts

Using the interrupts of this peripheral requires the Interrupt Controller to be configured first.

**Related Links**

CPUINT - CPU Interrupt Controller

SREG

Interrupts

#### 23.2.3.4 Events

The events of this peripheral are connected to the Event System.

**Related Links**

EVSYS - Event System

#### 23.2.3.5 Debug Operation

When run-time debugging, this peripheral will continue normal operation. Halting the CPU in debugging mode will halt normal operation of the peripheral.

This peripheral can be forced to operate with halted CPU by writing a '1' to the Debug Run bit (DBGRUN) in the Debug Control register of the peripheral (*peripheral*.DBGCTRL).

**Related Links**

UPDI - Unified Program and Debug Interface

## 23.3 RTC Functional Description

The RTC peripheral offers two timing functions: the Real-Time Counter (RTC) and a Periodic Interrupt Timer (PIT). This subsection describes the RTC.

**Related Links**

PIT Functional Description

### 23.3.1 Initialization

To operate the RTC, the source clock for the RTC counter must be configured before enabling the RTC peripheral, and the desired actions (interrupt requests, output Events).

**Related Links**

CLKCTRL - Clock Controller

PIT Functional Description

#### 23.3.1.1 Configure the clock CLK_RTC

To configure CLK_RTC, follow these steps:

1. Configure the desired oscillator to operate as required, in the Clock Controller peripheral (CLKCTRL).
2. Write the Clock Select bits (CLKSEL) in the Clock Selection register (RTC.CLKSEL) accordingly.

The CLK_RTC clock configuration is used by both RTC and PIT functionality.

#### 23.3.1.2 Configure RTC

To operate the RTC, follow these steps:

1. Configure the RTC-internal prescaler by writing the PRESCALER bit field in the Control A register (RTC.CTRLA).
2. Set the Compare value in the Compare register (RTC.CMP), and/or the Overflow value in the Top register (RTC.PER).
3. Enable the desired Interrupts by writing to the respective Interrupt Enable bits (CMP, OVF) in the Interrupt Control register (RTC.INTCTRL).
4. Enable the RTC by writing a '1' to the RTC Enable bit (RTCEN) in the Control A register (RTC.CTRLA).

### 23.3.2 Operation - RTC

#### 23.3.2.1 Enabling, Disabling, and Resetting

The RTC is enabled by setting the Enable bit in the Control A register (CTRLA.ENABLE=1). The RTC is disabled by writing CTRLA.ENABLE=0.

## 23.4 PIT Functional Description

The RTC peripheral offers two timing functions: the Real-Time Counter (RTC) and a Periodic Interrupt Timer (PIT). This subsection describes the PIT.

**Related Links**

RTC Functional Description

### 23.4.1 Initialization

To operate the PIT, follow these steps:

1. Configure the RTC clock CLK_RTC as described in Configure the clock CLK_RTC.
2. Select the period for the interrupt by writing the PERIOD bit field in the PIT Control A register (RTC.PITCTRLA).
3. Enable the interrupt by writing a '1' to the Periodic Interrupt bit (PI) in the PIT Interrupt Control register (RTC.PITINTCTRL).
4. Enable the PIT by writing a '1' to the PIT Enable bit (PITENABLE) in the PIT Control A register (RTC.PITCTRLA).

### 23.4.2 Operation - PIT

#### 23.4.2.1 Enabling, Disabling, and Resetting

The PIT is enabled by setting the Enable bit in the PIT Control A register (RTC_PITCTRLA.PITEN=1).
The PIT is disabled by writing RTC_PITCTRLA.PITEN=0.

## 23.5 Events

The RTC, when enabled, will generate the following output Events:

- Overflow (OVF): Generated when the counter has reached its top value and wrapped to zero.
- Compare (CMP): Indicates a match between the counter value and the compare register.

The PIT, when enabled, will generate the following output Events:

- Event 0: 8192 RTC clock periods interval.
- Event 1: 4096 RTC clock periods interval.
- Event 2: 2048 RTC clock periods interval.
- Event 3: 1024 RTC clock periods interval.
- Event 4: 512 RTC clock periods interval.
- Event 5: 256 RTC clock periods interval.
- Event 6: 128 RTC clock periods interval.
- Event 7: 64 RTC clock periods interval.

The Event users are configured by the Event System (EVSYS).

**Related Links**

EVSYS - Event System

## 23.6 Interrupts

**Table 23-2. Available Interrupt Vectors and Sources**

| Offset | Name | Vector Description | Conditions |
|---|---|---|---|
| 0x00 | RTC | Real-time counter overflow and compare match interrupt | • Overflow (OVF): The counter has reached its top value and wrapped to zero.<br>• Compare (CMP): Match between the counter value and the compare register. |
| 0x02 | PIT | Periodic Interrupt Timer interrupt | A time period has passed, as configured in RTC_PITCTRLA.PERIOD. |

When an interrupt condition occurs, the corresponding Interrupt Flag is set in the Interrupt Flags register of the peripheral (*peripheral*.INTFLAGS).

An interrupt source is enabled or disabled by writing to the corresponding enable bit in the peripheral's Interrupt Control register (*peripheral*.INTCTRL).

An interrupt request is generated when the corresponding interrupt source is enabled and the Interrupt Flag is set. The interrupt request remains active until the Interrupt Flag is cleared. See the peripheral's INTFLAGS register for details on how to clear Interrupt Flags.

**Related Links**

CPUINT - CPU Interrupt Controller

INTCTRL

PITINTCTRL

## 23.7 Sleep Mode Operation

The RTC will continue to operate in Idle sleep mode. It will run in Standby sleep mode if RTC.CTRLA.RUNSTDBY is set.

The PIT will continue to operate in any sleep mode.

**Related Links**

CTRLA

## 23.8 Synchronization

Both the RTC and the PIT are asynchronous, operating from a different clock source (CLK_RTC) independently of the main clock (CLK_PER). For control and count register updates, it will take a number of RTC clock and/or peripheral clock cycles before an updated register value is available in a register or until a configuration change has effect on the RTC or PIT, respectively. This synchronization time is described for each register in the Register Description.

For some RTC registers, a Synchronization Busy flag is available (CMPBUSY, PERBUSY, CNTBUSY, CTRLABUSY) in the Status register (RTC.STATUS).

For the RTC.PITCTRLA register, a Synchronization Busy flag (SYNCBUSY) is available in the PIT Status register (RTC.PITSTATUS).

Check for busy should be performed before writing to the mentioned registers.

**Related Links**

CLKCTRL - Clock Controller

## 23.9 Configuration Change Protection

Not applicable.

## 23.10 Register Summary - RTC

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | RUNSTDBY | PRESCALER[3:0] | | | | | | RTCEN |
| 0x01 | STATUS | 7:0 | | | | | CMPBUSY | PERBUSY | CNTBUSY | CTRLABUSY |
| 0x02 | INTCTRL | 7:0 | | | | | | | CMP | OVF |
| 0x03 | INTFLAGS | 7:0 | | | | | | | CMP | OVF |
| 0x04 | TEMP | 7:0 | TEMP[7:0] | | | | | | | |
| 0x05 | DBGCTRL | 7:0 | | | | | | | | DBGRUN |
| 0x06 | Reserved | | | | | | | | | |
| 0x07 | CLKSEL | 7:0 | | | | | | | CLKSEL[1:0] | |
| 0x08 | CNT | 7:0 | CNT[7:0] | | | | | | | |
| 0x09 | | 15:8 | CNT[15:8] | | | | | | | |
| 0x0A | PER | 7:0 | PER[7:0] | | | | | | | |
| 0x0B | | 15:8 | PER[15:8] | | | | | | | |
| 0x0C | CMP | 7:0 | CMP[7:0] | | | | | | | |
| 0x0D | | 15:8 | CMP[15:8] | | | | | | | |
| 0x0E ... 0x0F | Reserved | | | | | | | | | |
| 0x10 | PITCTRLA | 7:0 | | PERIOD[3:0] | | | | | | PITEN |
| 0x11 | PITSTATUS | 7:0 | | | | | | | | CTRLBUSY |
| 0x12 | PITINTCTRL | 7:0 | | | | | | | | PI |
| 0x13 | PITINTFLAGS | 7:0 | | | | | | | | PI |
| 0x14 | Reserved | | | | | | | | | |
| 0x15 | PITDBGCTRL | 7:0 | | | | | | | | DBGRUN |

## 23.11 Register Description

### 23.11.1 Control A

**Name:** CTRLA
**Offset:** 0x00
**Reset:** 0x00
**Property:** -

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

**Bit 7 – RUNSTDBY: Run in Standby**

| Value | Description |
|---|---|
| 0 | RTC disabled in Standby sleep mode |
| 1 | RTC enabled in Standby sleep mode |

**Bits 6:3 – PRESCALER[3:0]: Prescaler**
These bits define the prescaling of the CLK_RTC clock signal. Due to synchronization between the RTC clock and system clock domains, there is a latency of two RTC clock cycles from updating the register until this has an effect. Application software needs to check that the STATUS.CTRLABUSY flag is cleared before writing to this register.

| Value | Name | Description |
|---|---|---|
| 0x0 | DIV1 | RTC clock / 1 (no prescaling) |
| 0x1 | DIV2 | RTC clock / 2 |
| 0x2 | DIV4 | RTC clock / 4 |
| 0x3 | DIV8 | RTC clock / 8 |
| 0x4 | DIV16 | RTC clock / 16 |
| 0x5 | DIV32 | RTC clock / 32 |
| 0x6 | DIV64 | RTC clock / 64 |
| 0x7 | DIV128 | RTC clock / 128 |
| 0x8 | DIV256 | RTC clock / 256 |
| 0x9 | DIV512 | RTC clock / 512 |
| 0xA | DIV1024 | RTC clock / 1024 |
| 0xB | DIV2048 | RTC clock / 2048 |
| 0xC | DIV4096 | RTC clock / 4096 |
| 0xD | DIV8192 | RTC clock / 8192 |
| 0xE | DIV16384 | RTC clock / 16384 |
| 0xF | DIV32768 | RTC clock / 32768 |

**Bit 0 – RTCEN: RTC Enable**

| Value | Description |
|---|---|
| 0 | RTC disabled |
| 1 | RTC enabled |

### 23.11.2 Status

**Name:** STATUS
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | CMPBUSY | PERBUSY | CNTBUSY | CTRLABUSY |
| Access | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bit 3 – CMPBUSY: Compare Synchronization Busy**
This bit is indicating whether the RTC is busy synchronizing the Compare register (RTC.CMP) in RTC clock domain.

**Bit 2 – PERBUSY: Period Synchronization Busy**
This bit is indicating whether the RTC is busy synchronizing the Period register (RTC.PER) in RTC clock domain.

**Bit 1 – CNTBUSY: Counter Synchronization Busy**
This bit is indicating whether the RTC is busy synchronizing the Count register (RTC.CNT) in RTC clock domain.

**Bit 0 – CTRLABUSY: Control A Synchronization Busy**
This bit is indicating whether the RTC is busy synchronizing the Control A register (RTC.CTRLA) in RTC clock domain.

### 23.11.3 Interrupt Control

**Name:** INTCTRL
**Offset:** 0x02
**Reset:** 0x00
**Property:** -

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

**Bit 1 – CMP: Compare Match Interrupt Enable**
Enable interrupt on compare match, i.e. when the Counter value (RTC.CNT) matches the Compare value (RTC.CMP).

**Bit 0 – OVF: Overflow Interrupt Enable**
Enable interrupt on counter overflow, i.e. when the Counter value (RTC.CNT) matched the Period value (RTC.PER) and wraps around to zero.

### 23.11.4 Interrupt Flag

**Name:** INTFLAGS
**Offset:** 0x03
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | CMP | OVF |
| Access | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bit 1 – CMP: Compare Match Interrupt Flag**
This flag is set when the Counter value (RTC.CNT) matches the Compare value (RTC.CMP).

Writing a '1' to this bit clears the flag.

**Bit 0 – OVF: Overflow Interrupt Flag**
This flag is set when the Counter value (RTC.CNT) has reached the Period value (RTC.PER) and wrapped to zero.

Writing a '1' to this bit clears the flag.

### 23.11.5 Temporary

The Temporary register is used by the CPU for single-cycle, 16-bit access to the 16-bit registers of this peripheral. It can also be read and written by software. See also Accessing 16-bit Registers. There is one common Temporary register for all the 16-bit registers of this peripheral.

**Name:** TEMP
**Offset:** 0x4
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TEMP[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 – TEMP[7:0]: Temporary**

### 23.11.6 Debug Control

**Name:** DBGCTRL
**Offset:** 0x05
**Reset:** 0x00
**Property:** -

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

**Bit 0 – DBGRUN: Debug Run**

| Value | Description |
|---|---|
| 0 | The peripheral is halted in break debug mode and ignores events. |
| 1 | The peripheral will continue to run in break debug mode when the CPU is halted. |

### 23.11.7 Clock Selection

**Name:** CLKSEL
**Offset:** 0x07
**Reset:** 0x00
**Property:** -

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

**Bits 1:0 – CLKSEL[1:0]: Clock Select**
Writing these bits selects the source for the RTC clock (CLK_RTC).

When configuring the RTC to use either XOSC32K or the external clock on TOSC1, XOSC32K needs to be enabled and the Source Select bit (SEL) and Run Standby bit (RUNSTDBY) in the XOSC32K Control A register of the Clock Controller (CLKCTRL.XOSC32KCTRLA) must be configured accordingly.

| Value | Description |
|---|---|
| 0x0 | 32KHz from OSCULP32K |
| 0x1 | 1KHz from OSCULP32K |
| 0x2 | 32.768kHz from XOSC32K |
| 0x3 | External clock from TOSC1 pin |

### 23.11.8 Count

The RTC.CNTL and RTC.CNTH register pair represents the 16-bit value, RTC.CNT. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

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

**Name:** CNT
**Offset:** 0x08
**Reset:** 0x00
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | CNT[15:8] | | | | | | | |
| 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 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CNT[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 15:8 – CNT[15:8]: Counter high byte**
These bits hold the MSB of the 16-bit counter register.

**Bits 7:0 – CNT[7:0]: Counter low byte**
These bits hold the LSB of the 16-bit counter register.

### 23.11.9 Period

The RTC.PERL and RTC.PERH register pair represents the 16-bit value, RTC.PER. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

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

**Name:** PER
**Offset:** 0x0A
**Reset:** 0xFF
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | PER[15:8] | | | | | | | |
| 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 |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | PER[7:0] | | | | | | | |
| 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 15:8 – PER[15:8]: Period high byte**
These bits hold the MSB of the 16-bit period register.

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

### 23.11.10 Compare

The RTC.CMPL and RTC.CMPH register pair represents the 16-bit value, RTC.CMP. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

**Name:** CMP
**Offset:** 0x0C
**Reset:** 0x00
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | CMP[15:8] | | | | | | | |
| 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 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CMP[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 15:8 – CMP[15:8]: Compare high byte**
These bits hold the MSB of the 16-bit compare register.

**Bits 7:0 – CMP[7:0]: Compare low byte**
These bits hold the LSB of the 16-bit compare register.

### 23.11.11 Periodic Interrupt Timer Control A

**Name:** PITCTRLA
**Offset:** 0x10
**Reset:** 0x00
**Property:** -

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

**Bits 6:3 – PERIOD[3:0]: Period**
Writing this bit field selects the number of RTC clock cycles between each interrupt.

| Value | Name | Description |
|---|---|---|
| 0x0 | OFF | No interrupt |
| 0x1 | CYC4 | 4 cycles |
| 0x2 | CYC8 | 8 cycles |
| 0x3 | CYC16 | 16 cycles |
| 0x4 | CYC32 | 32 cycles |
| 0x5 | CYC64 | 64 cycles |
| 0x6 | CYC128 | 128 cycles |
| 0x7 | CYC256 | 256 cycles |
| 0x8 | CYC512 | 512 cycles |
| 0x9 | CYC1024 | 1024 cycles |
| 0xA | CYC2048 | 2048 cycles |
| 0xB | CYC4096 | 4096 cycles |
| 0xC | CYC8192 | 8192 cycles |
| 0xD | CYC16384 | 16384 cycles |
| 0xE | CYC32768 | 32768 cycles |
| 0xF | - | Reserved |

**Bit 0 – PITEN: Periodic Interrupt Timer Enable**
Writing a '1' to this bit enables the Periodic Interrupt Timer.

### 23.11.12 Periodic Interrupt Timer Status

**Name:** PITSTATUS
**Offset:** 0x11
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | CTRLBUSY |
| Access | | R | R | R | R | R | R | R |
| Reset | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bit 0 – CTRLBUSY: PITCTRLA Synchronization Busy**
This bit indicates whether the RTC is busy synchronizing the Periodic Interrupt Timer Control A register (RTC.PITCTRLA) in the RTC clock domain.

### 23.11.13 PIT Interrupt Control

**Name:** PITINTCTRL
**Offset:** 0x12
**Reset:** 0x00
**Property:** -

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

**Bit 0 – PI: Periodic interrupt**

| Value | Description |
|---|---|
| 0 | The periodic interrupt is disabled |
| 1 | The periodic interrupt is enabled |

### 23.11.14 PIT Interrupt Flag

**Name:** PITINTFLAGS
**Offset:** 0x13
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | PI |
| Access | | R | R | R | R | R | R | R |
| Reset | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bit 0 – PI: Periodic interrupt Flag**
This flag is set when a periodic interrupt is issued.

Writing a '1' clears the flag.

### 23.11.15 Periodic Interrupt Timer Debug Control

**Name:** PITDBGCTRL
**Offset:** 0x15
**Reset:** 0x00
**Property:** -

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

**Bit 0 – DBGRUN: Debug Run**

| Value | Description |
|---|---|
| 0 | The peripheral is halted in break debug mode and ignores events. |
| 1 | The peripheral will continue to run in break debug mode when the CPU is halted. |

# 24. USART - Universal Synchronous and Asynchronous Receiver and Transmitter

## 24.1 Features

- Full-duplex or one-wire half-duplex operation
- Asynchronous or synchronous operation
  - Synchronous clock rates up to 1/2 of the device clock frequency
  - Asynchronous clock rates up to 1/8 of the device clock frequency
- Supports serial frames with:
  - 5, 6, 7, 8, or 9 data bits
  - Optionally even and odd parity bits
  - 1 or 2 stop bits
- Fractional baud rate generator
  - Can generate desired baud rate from any system clock frequency
  - No need for external oscillator with certain frequencies
- Built-in error detection and correction schemes
  - Odd or even parity generation and parity check
  - Data overrun and framing error detection
  - Noise filtering includes false start bit detection and digital low-pass filter
- Separate interrupts for
  - Transmit complete
  - Transmit Data Register empty
  - Receive complete
- Multiprocessor communication mode
  - Addressing scheme to address a specific devices on a multi-device bus
  - Enable unaddressed devices to automatically ignore all frames
- Start Frame detection in UART mode
- Master SPI mode
  - Double buffered operation
  - Configurable data order
  - Operation up to 1/2 of the peripheral clock frequency
- IRCOM module for IrDA compliant pulse modulation/demodulation
- LIN slave support
  - Auto-baud and Break character detection
- RS-485 support

## 24.2 Overview

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) peripheral is a fast and flexible serial communication module. The USART supports full-duplex communication, asynchronous and synchronous operation and one-wire configurations. The USART can be set in SPI master mode and used for SPI communication.

The USART uses three communication lines for data transfer:

- RxD for receiving
- TxD for transmitting
- XCK for the transmission clock in synchronous operation

Communication is frame based, and the frame format can be customized to support a wide range of standards. One frame can be directly followed by a new frame, or the communication line can return to the idle (high) state. A serial frame consists of:

- 1 start bit
- 5, 6, 7, 8, or 9 data bits (MSB or LSB first)
- Parity bit: Even, odd, or none
- 1 or 2 stop bits.

The USART is buffered in both directions, enabling continued data transmission without any delay between frames. Separate interrupts for receive and transmit completion allow fully interrupt driven communication. Frame error and buffer overflow are detected in hardware and indicated with separate status flags. Even or odd parity generation and parity check can also be enabled.

The main functional blocks are the clock generator, the transmitter, and the receiver:

- The clock generator includes a fractional baud rate generator that is able to generate a wide range of USART baud rates from any system clock frequencies. This removes the need to use an external crystal oscillator with a specific frequency to achieve a required baud rate. It also supports external clock input in synchronous slave operation.
- The transmitter consists of a single write buffer (DATA), a shift register and a parity generator. The write buffer allows continuous data transmission without any delay between frames.
- The receiver consists of a two-level receive buffer (DATA) and a Shift Register. Data and clock recovery units ensure robust synchronization and noise filtering during asynchronous data reception. It includes frame error, buffer overflow, and parity error detection.

When the USART is set in one-wire mode, the transmitter and the receiver share the same RxD I/O pin.

When the USART is set in master SPI mode, all USART-specific logic is disabled, leaving the transmit and receive buffers, Shift registers, and baud rate generator enabled. Pin control and interrupt generation are identical in both modes. The registers are used in both modes, but their functionality differs for some control settings.

An IRCOM module can be enabled for one USART to support IrDA 1.4 physical compliant pulse modulation and demodulation for baud rates up to 115.2kbps.

The USART can be linked to the Configurable Custom Logic unit (CCL). When used with the CCL, the TxD/RxD data can be encoded/decoded before the signal is fed into the USART receiver or after the signal is output from transmitter when the USART is connected to CCL LUT outputs.

This device provides one instance of the USART peripheral, USART0.

**Related Links**

CCL – Configurable Custom Logic

### 24.2.1 Signal Description

| Signal | Type | Description |
|---|---|---|
| RxD | Input/output | Receiving line |
| TxD | Output | Transmitting line |

| Signal | Type | Description |
|---|---|---|
| XCK | Input/output | Clock for synchronous operation |
| XDIR | Output | Transmit Enable for RS485 |

**Related Links**

I/O Multiplexing and Considerations

### 24.2.2 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 24-1. USART System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | Yes | PORT |
| Interrupts | Yes | CPUINT |
| Events | Yes | EVSYS |
| Debug | Yes | UPDI |

**Related Links**

Debug Operation

Clocks

I/O Lines and Connections

Interrupts

Events

#### 24.2.2.1 Clocks

This peripheral depends on the peripheral clock.

**Related Links**

CLKCTRL - Clock Controller

#### 24.2.2.2 I/O Lines and Connections

Using the I/O lines of the peripheral requires configuration of the I/O pins.

**Related Links**

PORT - I/O Pin Configuration

I/O Multiplexing and Considerations

#### 24.2.2.3 Interrupts

Using the interrupts of this peripheral requires the Interrupt Controller to be configured first.

**Related Links**

CPUINT - CPU Interrupt Controller

SREG

Interrupts

#### 24.2.2.4 Events

The events of this peripheral are connected to the Event System.

**Related Links**

EVSYS - Event System

### 24.2.2.5 Debug Operation

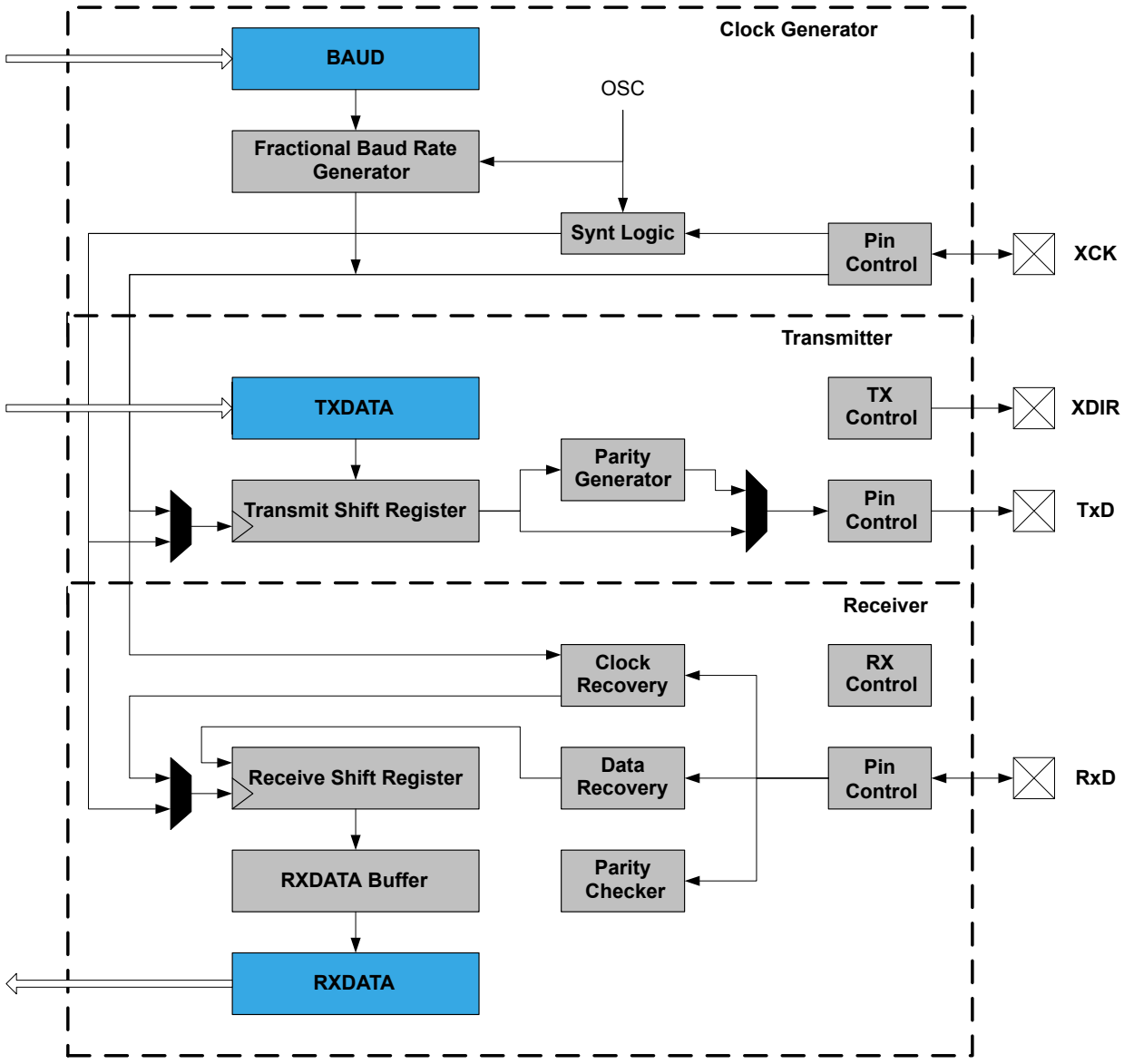
When the CPU is halted in debug mode, this peripheral will continue normal operation. If the peripheral is configured to require periodical service by the CPU through interrupts or similar, improper operation or data loss may result during debugging. This peripheral can be forced to halt operation during debugging.

**Related Links**

UPDI - Unified Program and Debug Interface

### 24.2.2.6 Block Diagram

**Figure 24-1. USART Block Diagram**



## 24.3 Functional Description

### 24.3.1 Initialization

For setting the USART in full-duplex mode, the following initialization sequence is recommended:

1. Set the TxD pin value high, and optionally set the XCK pin low (OUT[n] in PORT.OUT).
2. Set the TxD and optionally the XCK pin as output (DIR[n] in PORT.DIR).
3. Set the baud rate (USART.BAUD) and frame format.
4. Set the mode of operation (enables XCK pin output in synchronous mode).
5. Enable the transmitter or the receiver, depending on the usage.

For interrupt-driven USART operation, global interrupts should be disabled during the initialization.

Before doing a re-initialization with a changed baud rate or frame format, be sure that there are no ongoing transmissions while the registers are changed.

For setting the USART in one-wire mode, the following initialization sequence is recommended:

1. Set the TxD/RxD pin value high, and optionally set the XCK pin low.
2. Optionally, write the ODME bit in the USART.CTRLB register to '1' for Wired-AND functionality.
3. Set the TxD/RxD and optionally the XCK pin as output.
4. Select the baud rate and frame format.
5. Select the mode of operation (enables XCK pin output in synchronous mode).
6. Enable the transmitter or the receiver, depending on the usage.

For interrupt-driven USART operation, global interrupts should be disabled during the initialization.

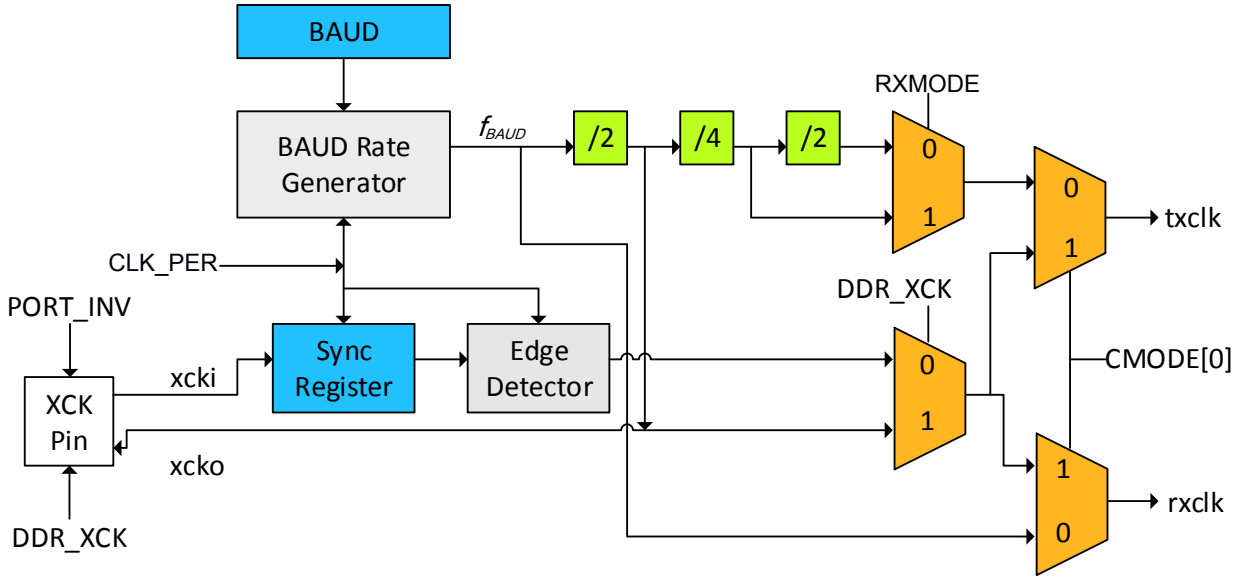
Before doing a re-initialization with a changed baud rate or frame format, be sure that there are no ongoing transmissions while the registers are changed.

### 24.3.2 Operation

#### 24.3.2.1 Clock Generation

The clock used for baud rate generation and for shifting and sampling data bits is generated internally by the fractional baud rate generator or externally from the transfer clock (XCK) pin. Five modes of clock generation are supported: normal and double-speed asynchronous mode, master and slave synchronous mode, and master SPI mode.

**Figure 24-2. Clock Generation Logic Block Diagram**




**Internal Clock Generation - The Fractional Baud Rate Generator**

The baud rate generator is used for internal clock generation for asynchronous modes, synchronous master mode, and master SPI mode operation. The output frequency generated ($f_{BAUD}$) is determined by the Baud register value (BAUD in USART.BAUD) and the peripheral clock frequency ($f_{CLK_PER}$). The following table contains equations for calculating the baud rate (in bits per second) and for calculating the BAUD value for each mode of operation. It also shows the maximum baud rate versus peripheral clock frequency. For asynchronous operation, the BAUD register value is 16 bits. The 10 MSBs (BAUD[15:6]) hold the integer part, while the 6 LSBs (BAUD[5:0]) hold the fractional part. In synchronous mode, only the integer part of the BAUD register determine the baud rate.

**Table 24-2. Equations for Calculating Baud Rate Register Setting**

| Operating Mode | Conditions | Baud Rate (Bits Per Seconds) | USART.BAUD Register Value Calculation |
|---|---|---|---|
| Asynchronous | $f_{BAUD} \leq \frac{f_{CLK_PER}}{S}$ | $f_{BAUD} = \frac{64 \times f_{CLK_PER}}{S \times BAUD}$ | $BAUD = \frac{64 \times f_{CLK_PER}}{S \times f_{BAUD}}$ |
| Synchronous | $f_{BAUD} \leq \frac{f_{CLK_PER}}{2}$ | $f_{BAUD} = \frac{f_{CLK_PER}}{2 \times BAUD[15:6]}$ | $BAUD[15:6] = \frac{f_{CLK_PER}}{2 \times f_{BAUD}}$ |

*S* is the number of samples per bit. In Asynchronous operating mode (CMODE[0]=0), it could be set as 16 (NORMAL mode) or 8 (CLK2X mode) by RXMODE in USART.CTRLB. For Synchronous operating mode (CMODE[0]=1), *S* equals 2.

**External Clock**

External clock (XCK) is used in synchronous slave mode operation. The XCK clock input is sampled on the peripheral clock frequency, and the maximum XCK clock frequency ($f_{XCK}$) is limited by the following:

$$f_{XCK} < \frac{f_{CLK_PER}}{4}$$

For each high and low period, XCK clock cycles must be sampled twice by the peripheral clock. If the XCK clock has jitter, or if the high/low period duty cycle is not 50/50, the maximum XCK clock speed must be reduced accordingly.

#### Double Speed Operation

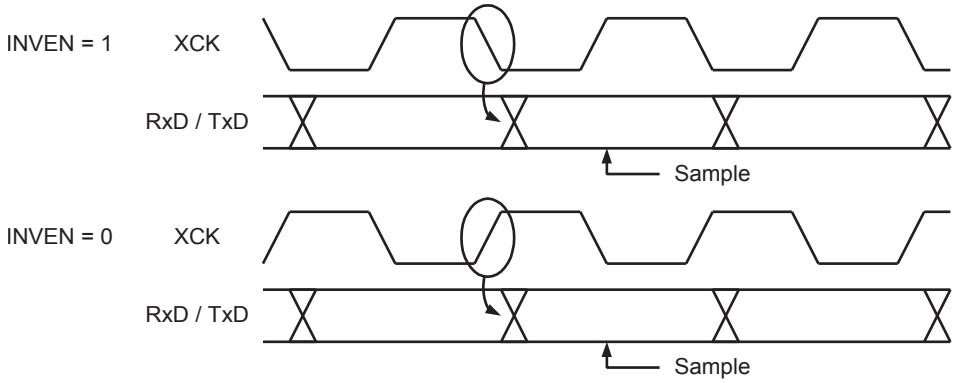
Double speed operation allows for higher baud rates under asynchronous operation with lower peripheral clock frequencies. This operation mode is enabled by writing the RXMODE bit in the Control B register (USART.CTRLB) to CLK2X.

When enabled, the baud rate for a given asynchronous baud rate setting shown in Table 24-2 will be doubled. In this mode, the receiver will use half the number of samples (reduced from 16 to 8) for data sampling and clock recovery. This requires a more accurate baud rate setting and peripheral clock. See Asynchronous Data Reception for more details.

#### Synchronous Clock Operation

When synchronous mode is used, the XCK pin controls whether the transmission clock is input (slave mode) or output (master mode). The corresponding port pin must be set to output for master mode or to input for slave mode (PORT.DIR[n]). The normal port operation of the XCK pin will be overridden. The dependency between the clock edges and data sampling or data change is the same. Data input (on RxD) is sampled at the XCK clock edge which is opposite the edge where data output (TxD) is changed.

**Figure 24-3. Synchronous Mode XCK Timing**




The I/O pin can be inverted by writing a '1' to the Inverted I/O Enable bit (INVEN) in the Pin n Control register of the Port peripheral (PORT.PINnCTRL). Using the inverted I/O setting for the corresponding XCK port pin, the XCK clock edges used for data sampling and data change can be selected. If inverted I/O is disabled (INVEN=0), data will be changed at the rising XCK clock edge and sampled at the falling XCK clock edge. If inverted I/O is enabled (INVEN=1), data will be changed at the falling XCK clock edge and sampled at the rising XCK clock edge.

#### Master SPI Mode Clock Generation

For master SPI mode operation, only internal clock generation is supported. This is identical to the USART synchronous master mode, and the baud rate or BAUD setting is calculated using the same equations (see Table 24-2).

There are four combinations of the SPI clock (SCK) phase and polarity with respect to the serial data, and these are determined by the Clock Phase bit (UCPHA) in the Control C register (USART.CTRLC) and the Inverted I/O Enable bit (INVEN) in the Pin n Control register of the Port peripheral (PORT.PINnCTRL). The data transfer timing diagrams are shown in Figure 24-4.

Data bits are shifted out and latched in on opposite edges of the XCK signal, ensuring sufficient time for data signals to stabilize. The settings are summarized in the following Table. Changing the setting of any of these bits during transmission will corrupt both the receiver and transmitter.

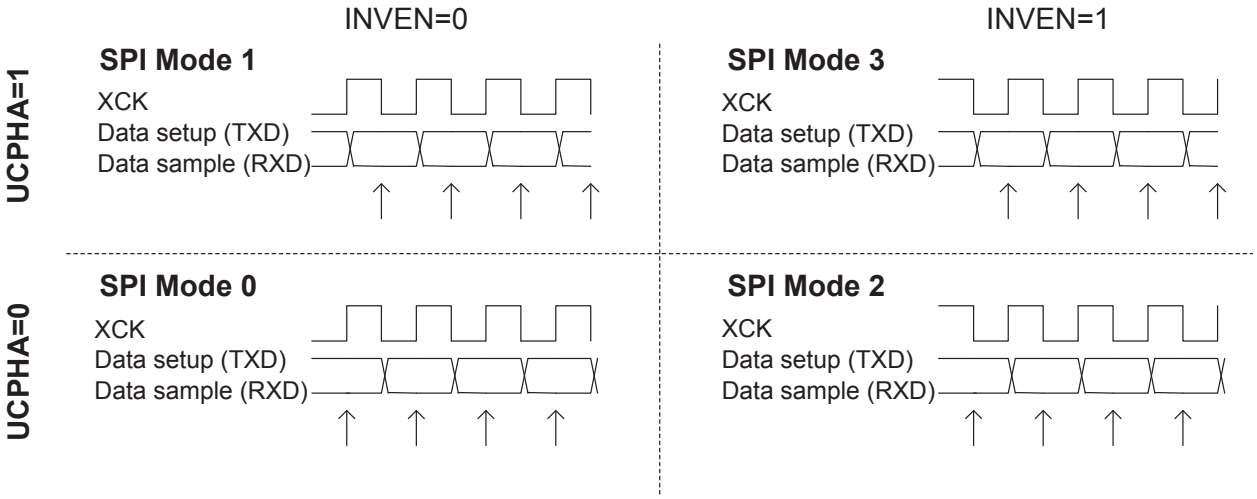
© 2017 Microchip Technology Inc.

**Table 24-3. Functionality of INVEN in PORT.PINnCTRL and UCPHA in USART.CTRLC**

| SPI mode | INVEN | UCPHA | Leading edge | Trailing edge |
|---|---|---|---|---|
| 0 | 0 | 0 | Rising, sample | Falling, setup |
| 1 | 0 | 1 | Rising, setup | Falling, sample |
| 2 | 1 | 0 | Falling, sample | Rising, setup |
| 3 | 1 | 1 | Falling, setup | Rising, sample |

The leading edge is the first clock edge of a clock cycle. The trailing edge is the last clock edge of a clock cycle.

**Figure 24-4. UCPHA and INVEN Data Transfer Timing Diagrams**




**Related Links**

CTRLC

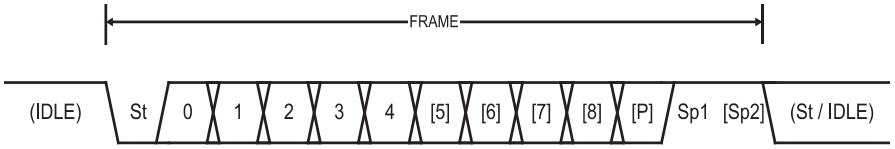
### 24.3.2.2 Frame Formats

Data transfer is frame based, where a serial frame consists of one character of data bits with synchronization bits (start and stop bits) and an optional parity bit for error checking. This does not apply to master SPI operation (See SPI Frame Formats). The USART accepts all combinations of the following as valid frame formats:

- 1 start bit
- 5, 6, 7, 8 or 9 data bits
- no, even, or odd parity bit
- 1 or 2 stop bits

Figure 24-5 illustrates the possible combinations of frame formats. Bits inside brackets are optional.

**Figure 24-5. Frame Formats**




St Start bit, always low.

(n) Data bits (0 to 8).

P Parity bit, may be odd or even.

Sp Stop bit, always high.

IDLE No transfer on the communication line (RxD or TxD). The IDLE state is always high.

**Parity**

Even or odd parity can be selected for error checking by writing the Parity Mode bits (PMODE) in the Control C register (USART.CTRLC). If even parity is selected, the parity bit is set to '1' if the number of logical one data bits is odd (making the total number of logical ones even). If odd parity is selected, the parity bit is set to '1' if the number of logical one data bits is even (making the total number of ones odd).

When enabled, the parity checker calculates the parity of the data bits in incoming frames and compares the result with the parity bit of the corresponding frame. If a parity error is detected, the parity error flag is set.

**SPI Frame Formats**

The serial frame in SPI mode is defined to be one character of eight data bits. The USART in master SPI mode has two valid frame formats:

- 8-bit data, msb first
- 8-bit data, lsb first

The data order is selected by writing to the Data Order bit (UDORD) in the Control C register (USART.CTRLC).

After a complete frame is transmitted, a new frame can directly follow it, or the communication line can return to the idle (high) state.

#### 24.3.2.3 Data Transmission - USART Transmitter

When the transmitter has been enabled, the normal Port operation of the TxD pin is overridden by the USART and given the function as the transmitter's serial output. The direction of the pin n must be configured as output by writing the Direction register for the corresponding port (PORT_DIR.DIR[n]). If the USART is configured for one-wire operation, the USART will automatically override the RxD/TxD pin to output, when the transmitter is enabled.

**Related Links**

PORTMUX - Port Multiplexer

PORT - I/O Pin Configuration

**Sending Frames**

A data transmission is initiated by loading the Transmit buffer (DATA in USART.TXDATA) with the data to be sent. The data in the transmit buffer are moved to the Shift Register when the Shift Register is empty and ready to send a new frame. The Shift Register is loaded if it is in idle state (no ongoing transmission) or immediately after the last stop bit of the previous frame is transmitted. When the Shift Register is loaded with data, it will transfer one complete frame.

When the entire frame in the Shift Register has been shifted out and there are no new data present in the transmit buffer, the Transmit Complete Interrupt Flag (TXCIF in USART.STATUS) is set and the optional interrupt is generated.

TXDATA can only be written when the Data Register Empty Flag (DREIF in USART.STATUS) is set, indicating that the register is empty and ready for new data.

When using frames with fewer than eight bits, the most-significant bits written to TXDATA are ignored. If 9-bit characters are used, DATA[8] in USART.TXDATAH has to be written before DATA[7:0] in USART.TXDATAL.

#### Disabling the Transmitter

A disabling of the transmitter will not become effective until ongoing and pending transmissions are completed; i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When the transmitter is disabled, it will no longer override the TxDn pin, and the pin direction is set as input automatically by hardware, even if it was configured as output by the user.

### 24.3.2.4 Data Reception - USART Receiver

When the receiver is enabled, the RxD pin functions as the receiver's serial input. The direction of the pin n must be set as input in the Direction register of the Port (PORT_DIR.DIR[n]=0), which is the default pin setting.

#### Receiving Frames

The receiver starts data reception when it detects a valid start bit. Each bit that follows the start bit will be sampled at the baud rate or XCK clock, and shifted into the Receive Shift Register until the first stop bit of a frame is received. A second stop bit will be ignored by the receiver. When the first stop bit is received and a complete serial frame is present in the Receive Shift Register, the contents of the Shift Register will be moved into the receive buffer. The receive complete interrupt flag (RXCIF in USART.STATUS) is set, and the optional interrupt is generated.

The receiver buffer can be read by reading RXDATA, comprising of DATA[7:0] in USART.RXDATAL, and DATA[8] in USART.RXDATAH. RXDATA should not be read unless the Receive Complete Interrupt Flag (RXCIF in USART.STATUS) is set. When using frames with fewer than eight bits, the unused most-significant bits are read as zero. If 9-bit characters are used, the ninth bit (DATA[8] in USART.RXDATAH) must be read before the low byte (DATA[7.0] in USART.RXDATAL).

#### Receiver Error Flags

The USART receiver has three error flags in the Receiver Data Register High Byte register (USART.RXDATAH):

- Frame Error (FERR)
- Buffer overflow (BUFOVF)
- Parity error (PERR)

The error flags are located in the receive FIFO buffer together with their corresponding frame. Due to the buffering of the error flags, the USART.RXDATAH must be read before the USART.RXDATAL, since reading the USART.RXDATAL changes the FIFO buffer.

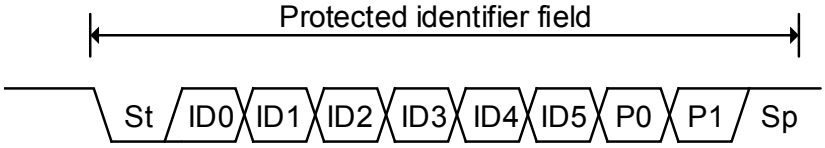
#### Parity Checker

When enabled, the parity checker calculates the parity of the data bits in incoming frames and compares the result with the parity bit of the corresponding frame. If a parity error is detected, the Parity Error flag (PERR in USART.RXDATAH) is set.

If USART LIN mode is enabled (by writing RXMODE to '1' in USART.CTRLB), a parity check is only performed on the protected identifier field. An parity error is detected if one of the equations below is not true which sets PERR in USART.RXDATAH.

$$P0 = ID0 \oplus ID1 \oplus ID2 \oplus ID4$$
$$P1 = \neg\left(ID1 \oplus ID3 \oplus ID4 \oplus ID5\right)$$

**Figure 24-6. Protected identifier field and mapping of identifier and parity bits**



#### Disabling the Receiver

A disabling of the receiver will be immediate. The receiver buffer will be flushed, and data from ongoing receptions will be lost.

#### Flushing the Receive Buffer

If the receive buffer has to be flushed during normal operation, read the DATA location (USART.RXDATAH and USART.RXDATAL registers) until the Receive Complete Interrupt Flag (RXCIF in USART.RXDATAH) is cleared.

#### Asynchronous Data Reception

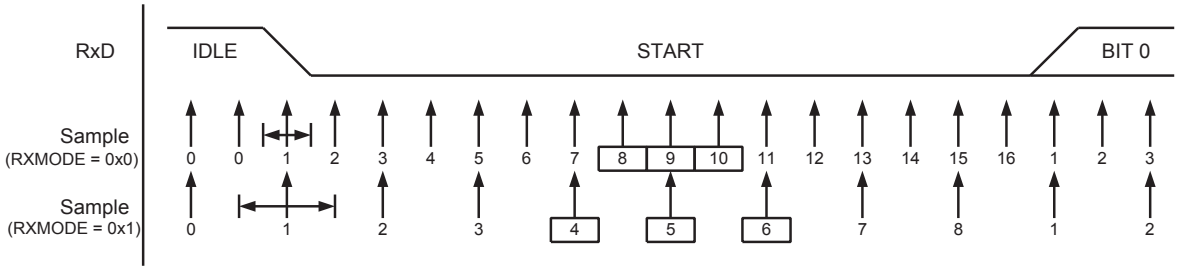
The USART includes a clock recovery and a data recovery unit for handling asynchronous data reception.

The clock recovery unit is used for synchronizing the incoming asynchronous serial frames at the RxD pin to the internally generated baud rate clock. It samples and low-pass filters each incoming bit, thereby improving the noise immunity of the receiver. The asynchronous reception operational range depends on the accuracy of the internal baud rate clock, the rate of the incoming frames, and the frame size in number of bits.

##### *Asynchronous Clock Recovery*

The clock recovery unit synchronizes the internal clock to the incoming serial frames. Figure 24-7 illustrates the sampling process for the start bit of an incoming frame. The sample rate is 16 times the baud rate for normal mode, and eight times the baud rate for double speed mode. The horizontal arrows illustrate the synchronization variation due to the sampling process. The larger time variation when using the double speed mode of operation. Samples denoted as zero are samples done when the RxD line is idle; i.e., when there is no communication activity.

**Figure 24-7. Start Bit Sampling**

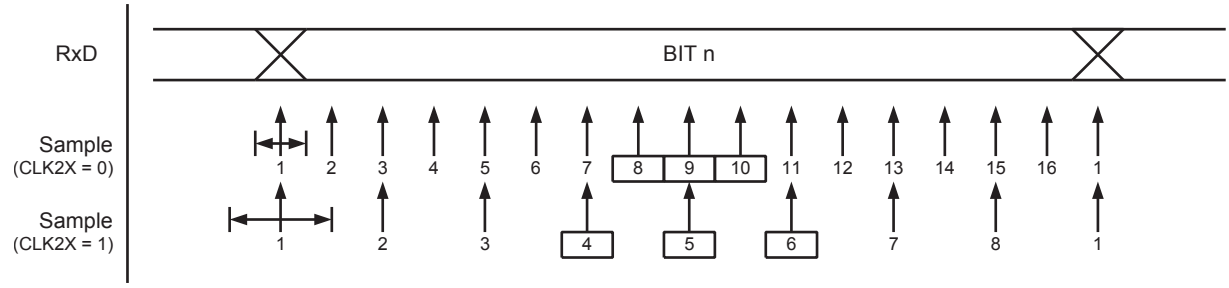



When the clock recovery logic detects a high (idle) to low (start) transition on the RxD line, the start bit detection sequence is initiated. Sample 1 denotes the first zero-sample, as shown in the figure. The clock recovery logic then uses samples 8, 9, and 10 for normal mode and samples 4, 5, and 6 for double speed mode to decide if a valid start bit is received. If two or three samples have a low level, the start bit is accepted. The clock recovery unit is synchronized, and the data recovery can begin. If two or three samples have a high level, the start bit is rejected as a noise spike, and the receiver looks for the next high-to-low transition. The process is repeated for each start bit.

##### *Asynchronous Data Recovery*

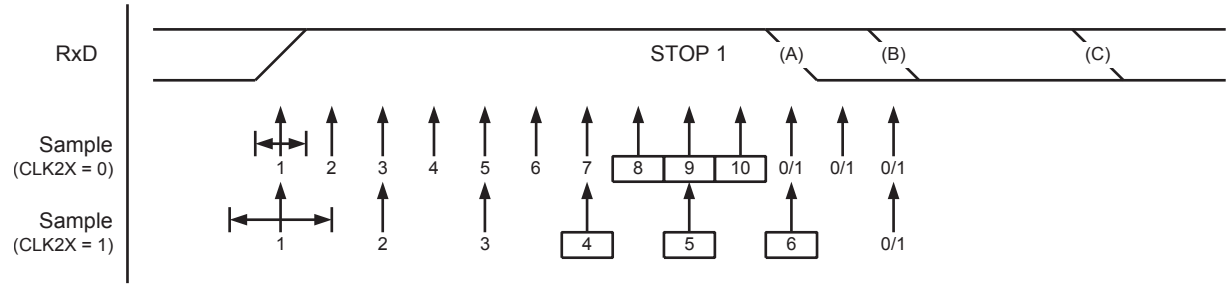
The data recovery unit uses sixteen samples in normal mode and eight samples in double speed mode for each bit. The following Figure shows the sampling process of data and parity bits.

**Figure 24-8. Sampling of Data and Parity Bits**




As for start bit detection, an identical majority voting technique is used on the three center samples for deciding of the logic level of the received bit. The process is repeated for each bit until a complete frame is received. It includes the first stop bit, but excludes additional ones. If the sampled stop bit is a '0' value, the Frame Error (FERR in USART.RXDATAH) flag will be set. The next Figure shows the sampling of the stop bit in relation to the earliest possible beginning of the next frame's start bit.

**Figure 24-9. Stop Bit and Next Start Bit Sampling**




A new high-to-low transition indicating the start bit of a new frame can come right after the last of the bits used for majority voting. For normal speed mode, the first low level sample can be at the point marked (A) in Stop Bit Sampling and Next Start Bit Sampling. For double speed mode, the first low level must be delayed to point (B). Point (C) marks a stop bit of full length at nominal baud rate. The early start bit detection influences the operational range of the receiver.

**Asynchronous Operational Range**

The operational range of the receiver is dependent on the mismatch between the received bit rate and the internally generated baud rate. If an external transmitter is sending using bit rates that are too fast or too slow, or if the internally generated baud rate of the receiver does not match the external source's base frequency, the receiver will not be able to synchronize the frames to the start bit.

The following equations can be used to calculate the ratio of the incoming data rate and internal receiver baud rate.

| | |
|---|---|
| $R_{SLOW} = \frac{16(D+1)}{16(D+1)+6}$ | $R_{FAST} = \frac{16(D+2)}{16(D+1)+8}$ |

*D* Sum of character size and parity size (D = 5 to 10 bit).

$R_{slow}$ 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. Normal Speed mode has higher toleration of baud rate variations.

**Table 24-4. Recommended Maximum Receiver Baud Rate Error for Normal Speed Mode (CLK2X = 0)**

| D #(Data + Parity bit) | $R_{slow}$ [%] | $R_{fast}$ [%] | Maximum total error [%] | Receiver max. receiver error [%] |
|---|---|---|---|---|
| 5 | 93.20 | 106.67 | +6.67/-6.80 | ±3.0 |
| 6 | 94.12 | 105.79 | +5.79/-5.88 | ±2.5 |
| 7 | 94.81 | 105.11 | +5.11/-5.19 | ±2.0 |
| 8 | 95.36 | 104.58 | +4.58/-4.54 | ±2.0 |
| 9 | 95.81 | 104.14 | +4.14/-4.19 | ±1.5 |
| 10 | 96.17 | 103.78 | +3.78/-3.83 | ±1.5 |

**Table 24-5. Recommended Maximum Receiver Baud Rate Error for Double Speed Mode (CLK2X = 1)**

| D #(Data + Parity bit) | $R_{slow}$ [%] | $R_{fast}$ [%] | Maximum total error [%] | Receiver 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 were made under the assumption that the receiver and transmitter equally divide the maximum total error.

#### 24.3.2.5 USART in Master SPI mode

Using the USART in master SPI mode requires the transmitter to be enabled. The receiver can optionally be enabled to serve as the serial input. The XCK pin will be used as the transfer clock.

As for the USART, a data transfer is initiated by writing to the DATA Register. This is the case for both sending and receiving data, since the transmitter controls the transfer clock. The data written to DATA are moved from the transmit buffer to the Shift Register when the Shift Register is ready to send a new frame.

The transmitter and receiver interrupt flags and corresponding USART interrupts used in master SPI mode are identical in function to their use in normal USART operation. The receiver error status flags are not in use and are always read as zero.

Disabling of the USART transmitter or receiver in master SPI mode is identical to their disabling in normal USART operation.

**Related Links**

CTRLC

**USART SPI vs. SPI**

The USART in master SPI mode is fully compatible with the standalone SPI module in that:

- Timing diagrams are the same

- UCPHA bit functionality is identical to that of the SPI CPHA bit
- UDORD bit functionality is identical to that of the SPI DORD bit

When the USART is set in master SPI mode, configuration and use are in some cases different from those of the standalone SPI module. In addition, the following difference exists:

- The USART in master SPI mode does not include the SPI (Write Collision) feature

The USART in master SPI mode does not include the SPI double speed mode feature, but this can be achieved by configuring the baud rate generator accordingly:

- Interrupt timing is not compatible
- Pin control differs due to the master-only operation of the USART in SPI master mode

A comparison of the USART in master SPI mode and the SPI pins is shown in Table 24-6.

**Table 24-6. Comparison of USART in master SPI mode and SPI pins.**

| USART | SPI | Comment |
|---|---|---|
| TxD | MOSI | Master out only |
| RxD | MISO | Master in only |
| XCK | SCK | Functionally identical |
| - | SS | Not supported by USART in master SPI mode |

**Related Links**

CTRLC

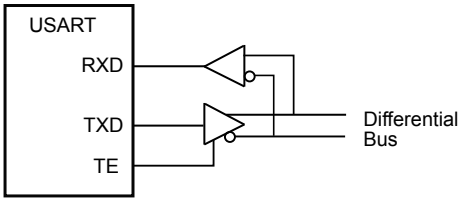
#### 24.3.2.6 RS485 Mode of Operation

The RS485 feature enables the support of external components to comply with the RS-485 standard.

Either an external line driver is supported as shown in the figure below (RS485=0x1 in USART.CTRLA), or control of the transmitter driving the TxD pin is provided (RS485=0x2).

While operating in RS485 mode, the Transmit Enable pin (TE) is driven high when the transmitter is active.

**Figure 24-10. RS485 Bus Connection**




The TE pin goes high one baud clock cycle in advance of data being shifted out, to allow some guard time to enable the external line driver. The TE pin will remain high for the complete frame including stop bit(s).

**Figure 24-11. TE Drive Timing**

**Related Links**

Signal Description

#### 24.3.2.7 Start Frame Detection

The start frame detection is supported in UART mode only. The UART start frame detector is limited to Standby sleep mode only and can wake up the system when a start bit is detected.

When a high-to-low transition is detected on RxDn, the oscillator is powered up and the UART clock is enabled. After start-up, the rest of the data frame can be received, provided that the baud rate is slow enough in relation to the oscillator start-up time. Start-up time of the oscillators varies with supply voltage and temperature. For details on oscillator start-up time characteristics, refer to the Electrical Characteristics.

If a false start bit is detected and if the system has not been waken-up by another source, the clock will automatically be turned off and the UART waits for the next transition.

The UART start frame detection works in asynchronous mode only. It is enabled by writing the Start Frame Detection bit (SFDEN) in CTRLB. If the start bit is detected while the device is in Standby sleep mode, the UART Start Interrupt Flag (RXSIF) bit is set.

In Active, Idle and Power Down sleep modes, the asynchronous detection is automatically disabled.

The UART receive complete flag and UART start interrupt flag share the same interrupt line, but each has its dedicated interrupt settings. The Table 21-5 shows the USART start frame detection modes, depending of interrupt setting.

**Table 24-7. USART Start Frame Detection Modes**

| SFDEN | RXSIF interrupt | RXCIF interrupt | Comment |
|---|---|---|---|
| 0 | x | x | Standard mode |
| 1 | Disabled | Disabled | Only the oscillator is powered during the frame reception. If the interrupts are disabled and buffer overflow is ignored, all incoming frames will be lost |
| 1 (1) | Disabled | Enabled | System/all clocks waked-up on Receive Complete interrupt |
| 1 (1) | Enabled | x | System/all clocks waked-up on UART Start Detection |

**Note:**

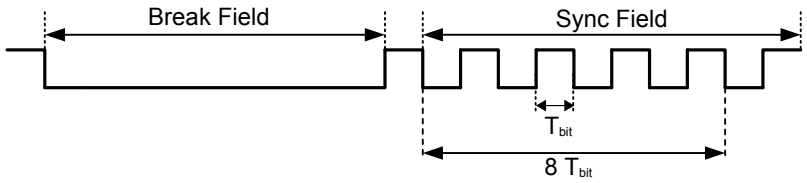
1. The SLEEP instruction will not shut down the oscillator if there is ongoing communication.

#### 24.3.2.8 Break Character Detection and Auto-baud

When USART receive mode is set to LINAUTO mode (RXMODE in USART.CTRLB), it follows the LIN format. All LIN Frames start with a Break Field followed by a Sync Field. The USART uses a break detection threshold of greater than 11 nominal bit times at the configured baud rate. At any time, if more than 11 consecutive dominant bits are detected on the bus, the USART detects a Break Field. When a

Break Field has been detected, the USART expects the Sync Field character to be 0x55. This field is used to update the actual baud rate in order to stay synchronized. If the received Sync character is not 0x55, then the Inconsistent Sync Field error flag (ISFIF in USART.STATUS) is set and the baud rate is unchanged.

**Figure 24-12. LIN Break and Sync Fields**




After a break field is detected and the start bit of the Sync Field is detected, a counter is started. The counter is then incremented for the next 8 $T_{bit}$ of the Sync Field. At the end of these 8 bit times, the counter is stopped. At this moment, the 10 most significant bits of the counter (value divided by 64) gives the new clock divider and the 6 least significant bits of this value (the remainder) gives the new fractional part. When the Sync Field has been received and all bits found valid, the clock divider and the fractional part are updated in the Baud Rate Generator register (USART.BAUD). After the Break and Sync Fields, n characters of data can be received.

When the USART receive mode is set to GENAUTO mode, a generic auto-baud mode is enabled. In this mode there no check of Sync character to equal 0x55. After detection of a Break Field the USART expects the next character to be a Sync field, counting 8 low and high bit times. If the measured Sync field result in a valid BAUD value (0x0064-0xffff), the BAUD register is updated. Setting the Wait for Break bit (WFB in USART.STATUS) before receiving the next Break character, the next negative plus positive edge of RxD line is detected as a Break. This makes it possible to set an arbitrary new Baud Rate without knowing the current Baud Rate.

#### 24.3.2.9 One-wire Mode

In this mode the TxD pin is connected to the RxD pin internally. If the receiver is enabled when transmitting it will receive what the transmitter is sending. This can be used to check that no one else is trying to transmit since received data will not be the same as the transmitted data.

#### 24.3.2.10 Multiprocessor Communication Mode

The multiprocessor communication mode (MCPM) effectively reduces the number of incoming frames that have to be handled by the receiver in a system with multiple microcontrollers communicating via the same serial bus. This mode is enabled by writing a '1' to the MCPM bit in the Control B register (USART.CTRLB). In this mode, a dedicated bit in the frames is used to indicate whether the frame is an address or data frame type.

If the receiver is set up to receive frames that contain five to eight data bits, the first stop bit is used to indicate the frame type. If the receiver is set up for frames with nine data bits, the ninth bit is used to indicate frame type. When the frame type bit is one, the frame contains an address. When the frame type bit is zero, the frame is a data frame. If 5-bit to 8-bit character frames are used, the transmitter must be set to use two stop bits, since the first stop bit is used for indicating the frame type.

If a particular slave MCU has been addressed, it will receive the following data frames as usual, while the other slave MCUs will ignore the frames until another address frame is received.

##### Using Multiprocessor Communication Mode

The following procedure should be used to exchange data in multiprocessor communication mode (MPCM):

1. All slave MCUs are in multiprocessor communication mode.
2. The master MCU sends an address frame, and all slaves receive and read this frame.

3. Each slave MCU determines if it has been selected.
4. The addressed MCU will disable MPCM and receive all data frames. The other slave MCUs will ignore the data frames.
5. When the addressed MCU has received the last data frame, it must enable MPCM again and wait for a new address frame from the master.

The process then repeats from step 2.

Using any of the 5- to 8-bit character frame formats is impractical, as the receiver must change between using n and n+1 character frame formats. This makes full-duplex operation difficult, since the transmitter and receiver must use the same character size setting.

#### 24.3.2.11 IRCOM Mode of Operation

The IRCOM mode enables IrDA 1.4 compliant modulation and demodulation for baud rates up to 115.2kbps. When IRCOM mode is enabled, double speed mode cannot be used for the USART.

**Overview**

The device contains one infrared communication module (IRCOM) that is IrDA compatible for baud rates up to 115.2kbps.

The IRCOM is automatically enabled when the USART is set in IRCOM mode (by writing 0x2 to CTRLC.CMODE). The signals between the USART and the RX/TX pins are then routed through the module. The data on the TX/RX pins are the inverted value of the transmitted/received infrared pulse. It is also possible to select an event channel from the event system as input for the IRCOM receiver. This will disable the RX input from the USART pin.

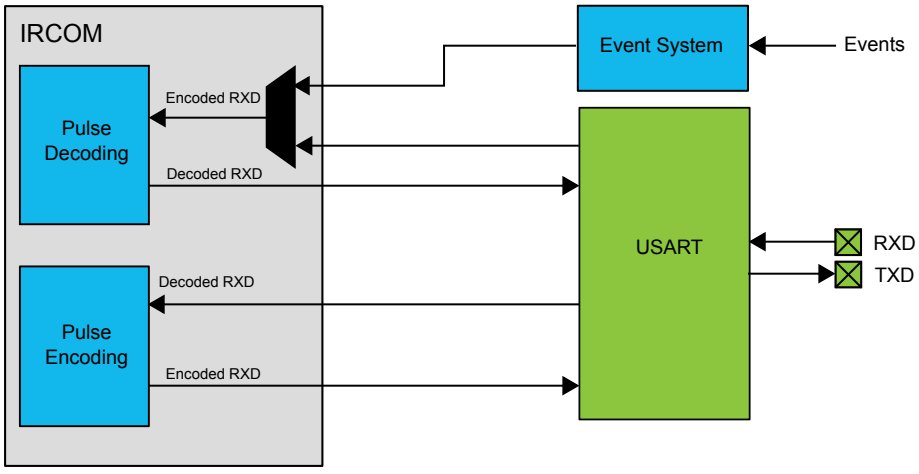
For transmission, three pulse modulation schemes are available:

- 3/16 of the baud rate period
- Fixed programmable pulse time based on the peripheral clock frequency
- Pulse modulation disabled

For reception, a fixed programmable minimum high-level pulse width for the pulse to be decoded as a logical '0' is used. Shorter pulses will then be discarded, and the bit will be decoded to logical '1' as if no pulse was received.

**Block Diagram**

**Figure 24-13. Block Diagram**



**IRCOM and Event System**

The event system can be used as the receiver input. This enables the IRCOM or USART input from the I/O pins or sources other than the corresponding RX pin. If event system input is enabled, input from the USART's RX pin is automatically disabled.

**Related Links**

EVSYS - Event System

### 24.3.3 Events

The USART can accept the following input Events:

- IREI - IrDA Event Input

The Event is enabled by writing a '1' to the IrDA Event Input bit (IREI) in the Event Control register (USART.EVCTRL).

**Related Links**

EVSYS - Event System

EVCTRL

### 24.3.4 Interrupts

**Table 24-8. Available Interrupt Vectors and Sources**

| Offset | Name | Vector Description | Conditions |
|---|---|---|---|
| 0x00 | RXC | Receive Complete interrupt | • There are unread data in the receive buffer (RXCIE)<br>• Receive of Start-of-Frame detected (RXSIE)<br>• Auto-Baud Error/ISFIF flag set (ABEIE) |
| 0x02 | DRE | Data Register Empty interrupt | The transmit buffer is empty/ready to receive new data (DREIE). |
| 0x04 | TXC | Transmit Complete interrupt | The entire frame in the Transmit Shift Register has been shifted out and there are no new data in the transmit buffer (TXCIE). |

When an interrupt condition occurs, the corresponding Interrupt Flag is set in the Status register (USART.STATUS).

An interrupt source is enabled or disabled by writing to the corresponding bit in the Control A register (USART.CTRLA).

An interrupt request is generated when the corresponding interrupt source is enabled and the Interrupt Flag is set. The interrupt request remains active until the Interrupt Flag is cleared. See the USART.STATUS register for details on how to clear Interrupt Flags.

**Related Links**

CPUINT - CPU Interrupt Controller

STATUS

CTRLA

### 24.3.5 Configuration Change Protection

Not applicable.

## 24.4 Register Summary - USART

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | RXDATAL | 7:0 | DATA[7:0] | | | | | | | |
| 0x01 | RXDATAH | 7:0 | RXCIF | BUFOVF | | | | FERR | PERR | DATA[8] |
| 0x02 | TXDATAL | 7:0 | DATA[7:0] | | | | | | | |
| 0x03 | TXDATAH | 7:0 | | | | | | | | DATA[8] |
| 0x04 | STATUS | 7:0 | RXCIF | TXCIF | DREIF | RXSIF | ISFIF | | BDF | WFB |
| 0x05 | CTRLA | 7:0 | RXCIE | TXCIE | DREIE | RXSIE | LBME | ABEIE | RS485[1:0] | |
| 0x06 | CTRLB | 7:0 | RXEN | TXEN | | SFDEN | ODME | RXMODE[1:0] | | MPCM |
| 0x07 | CTRLC | 7:0 | CMODE[1:0] | | PMODE[1:0] | | SBMODE | CHSIZE[2:0] | | |
| 0x08 | BAUD | 7:0 | BAUD[7:0] | | | | | | | |
| 0x09 | | 15:8 | BAUD[15:8] | | | | | | | |
| 0x0A | Reserved | | | | | | | | | |
| 0x0B | DBGCTRL | 7:0 | | | | | | | | DBGRUN |
| 0x0C | EVCTRL | 7:0 | | | | | | | | IREI |
| 0x0D | TXPLCTRL | 7:0 | TXPL[7:0] | | | | | | | |
| 0x0E | RXPLCTRL | 7:0 | | RXPL[6:0] | | | | | | |

## 24.5 Register Description

### 24.5.1 Receiver Data Register Low Byte

Reading the RXDATAL Register location will return the contents of the Receive Data Buffer Register (RXB).
The receive buffer consists of a two level FIFO. The FIFO and the corresponding flags in the high byte of RXDATA will change state whenever the receive buffer is accessed (read). If CHSIZE in USART.CTRLC is set to 9BIT Low byte first, read USART.RXDATAL before USART.RXDATAH, otherwise always read USART.RXDATAH before USART.RXDATAL in order to get the correct flags.

**Name:** RXDATAL
**Offset:** 0x00
**Reset:** 0x00
**Property:** R

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

**Bits 7:0 – DATA[7:0]: Receiver Data Register**

### 24.5.2 Receiver Data Register High Byte

Reading the RXDATAH Register location will return the contents of the ninth DATA bit plus status bits.

The receive buffer consists of a two level FIFO. The FIFO and the corresponding flags in the high byte of USART.RXDATAH will change state whenever the receive buffer is accessed (read). If CHSIZE in

USART.CTRLC is set to 9BIT Low byte first, read USART.RXDATAL before USART.RXDATAH, otherwise always read USART.RXDATAH before USART.RXDATAL in order to get the correct flags.

**Name:** RXDATAH
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RXCIF | BUFOVF | | | | FERR | PERR | DATA[8] |
| Access | R | R | | | | R | R | R |
| Reset | 0 | 0 | | | | 0 | 0 | 0 |

**Bit 7 – RXCIF: USART Receive Complete Interrupt Flag**
This flag is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). When the Receiver is disabled, the receive buffer will be flushed and consequently the RXCIF will become zero.

**Bit 6 – BUFOVF: Buffer Overflow**
The BUFOVF flag indicates data loss due to a receiver buffer full condition. This flag is set if a Buffer Overflow condition is detected. A Buffer Overflow occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. This flag is valid until the receive buffer (USART.RXDATAL) is read.

This flag is not used in Master SPI mode of operation.

**Bit 2 – FERR: Frame Error**
The FERR flag indicates the state of the first stop bit of the next readable frame stored in the receive buffer. The bit is set if the received character had a Frame Error, i.e. when the first stop bit was zero, and cleared when the stop bit of the received data is '1'. This bit is valid until the receive buffer (USART.RXDATAL) is read. The FERR is not affected by the SBMODE bit in USART.CTRLC since the Receiver ignores all, except for the first stop bit.

This flag is not used in Master SPI mode of operation.

**Bit 1 – PERR: Parity Error**
If parity checking is enabled and the next character in the receive buffer has a Parity Error this flag is set. If Parity Check is not enabled the PERR will always be read as zero. This bit is valid until the receive buffer (USART.RXDATAL) is read. For details on parity calculation refer to Parity . If USART is set to LINAUTO mode, this bit will be a Parity Check of the Protected identifier field and will be valid when DATA[8] in USART.RXDATAH reads low.

This flag is not used in Master SPI mode of operation.

**Bit 0 – DATA[8]: Receiver Data Register**
When USART receiver is set to LINAUTO mode, this bit indicates if the received data is within the response space of a LIN frame. If the received data is the Protected identifier field, this bit will be read as zero. Otherwise the bit will be read as one. For receiver mode other than LINAUTO mode, DATA[8] holds the ninth data bit in the received character when operating with serial frames with nine data bits.

### 24.5.3 Transmit Data Register Low Byte

The Transmit Data Buffer Register (TXB) will be the destination for data written to the USART.TXDATAL Register location.

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 DREIF Flag in the USART.STATUS Register is set. Data written to DATA when the DREIF 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. The data is then transmitted on the TxD pin.

**Name:** TXDATAL
**Offset:** 0x02
**Reset:** 0x00
**Property:** R/W

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DATA[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 – DATA[7:0]: Transmit Data Register**

### 24.5.4 Transmit Data Register High Byte

USART.TXDATAH holds the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. When used this bit must be written before writing to USART.TXDATAL except if CHSIZE in USART.CTRLC is set to 9BIT Low byte first where USART.TXDATAL should be written first. This bit is unused in Master SPI mode of operation.

**Name:** TXDATAH
**Offset:** 0x03
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | DATA[8] |
| Access | | | | | | | | W |
| Reset | | | | | | | | 0 |

**Bit 0 – DATA[8]: Transmit Data Register**
This bit is used when CHSIZE=9BIT in USART.CTRLC.

### 24.5.5 USART Status Register

**Name:** STATUS
**Offset:** 0x04
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RXCIF | TXCIF | DREIF | RXSIF | ISFIF | | BDF | WFB |
| Access | R | R/W | R | R/W | R/W | | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | | 0 | 0 |

**Bit 7 – RXCIF: USART Receive Complete Interrupt Flag**
This flag is set to '1' when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). When the Receiver is disabled, the receive buffer will be flushed and consequently the RXCIF will become zero.

When interrupt-driven data reception is used, the receive complete interrupt routine must read the received data from RXDATA in order to clear the RXCIF. If not, a new interrupt will occur directly after the return from the current interrupt.

**Bit 6 – TXCIF: USART Transmit Complete Interrupt Flag**
This flag is set when the entire frame in the Transmit Shift Register has been shifted out and there are no new data in the transmit buffer (TXDATA).

This flag is automatically cleared when the transmit complete interrupt vector is executed. The flag can also be cleared by writing a '1' to its bit location.

**Bit 5 – DREIF: USART Data Register Empty Flag**
The DREIF indicates if the transmit buffer (TXDATA) is ready to receive new data. The flag is set to '1' when the transmit buffer is empty, and is '0' when the transmit buffer contains data to be transmitted that has not yet been moved into the Shift Register. DREIF is set after a reset to indicate that the Transmitter is ready. Always write this bit to '0' when writing the STATUS register.

DREIF is cleared to '0' by writing TXDATAL. When interrupt-driven data transmission is used, the Data Register Empty interrupt routine must either write new data to TXDATA in order to clear DREIF or disable the Data Register Empty interrupt. If not, a new interrupt will occur directly after the return from the current interrupt.

**Bit 4 – RXSIF: USART Receive Start Interrupt Flag**
The RXSIF flag indicates a valid start condition on RxD line. The flag is set when the system is in standby modes and a high (IDLE) to low (START) valid transition is detected on the RxD line. If the start detection is not enabled, the RXSIF will always be read as zero. This flag can only be cleared by writing a '1' to its bit location. This flag is not used in master SPI mode operation.

**Bit 3 – ISFIF: Inconsistent Sync Field Interrupt Flag**
This bit is set when the auto-baud is enabled and the sync field bit time are too fast or too slow to give a valid baud setting. It will also be set when USART is set to LINAUTO mode and the SYNC character differ from data value 0x55.

Writing a '1' to this bit will clear the flag and bring the USART back to idle state.

**Bit 1 – BDF: Break Detected Flag**
This bit is intended for USART configured to LINAUTO receive mode, see CTRLB. The break detector has a fixed threshold of 11 bits low for a BREAK to be detected. The BDF bit is set after a valid BREAK and SYNC character is detected. The bit is automatically cleared when next data is received. The bit will behave identically when USART is set to GENAUTO mode. In NORMAL or CLK2X receive mode, the BDF bit is unused.

This bit is cleared by writing a '1' to it.

**Bit 0 – WFB: Wait For Break**
Writing this bit to '1' will register the next low and high transition on RxD line as a break character. This can be used to wait for a BREAK character of arbitrary width. Combined with USART set to GENAUTO mode, this allow the user to set any BAUD rate trough BREAK and SYNC as long as it falls within valid range of the USART.BAUD register. This bit will always read '0'.

### 24.5.6 Control A

**Name:** CTRLA
**Offset:** 0x05
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RXCIE | TXCIE | DREIE | RXSIE | LBME | ABEIE | RS485[1:0] | |
| 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 – RXCIE: Receive Complete Interrupt Enable**
The bit enables the Receive Complete Interrupt (interrupt vector RXC). The enabled interrupt will be triggered when RXCIF in the USART.STATUS register is set.

**Bit 6 – TXCIE: Transmit Complete Interrupt Enable**
This bit enables the Transmit Complete Interrupt (interrupt vector TXC). The enabled interrupt will be triggered when the TXCIF in the USART.STATUS register is set.

**Bit 5 – DREIE: Data Register Empty Interrupt Enable**
This bit enables the Data Register Empty Interrupt (interrupt vector DRE). The enabled interrupt will be triggered when the DREIF in the USART.STATUS register is set.

**Bit 4 – RXSIE: Receiver Start Frame Interrupt Enable**
Writing a '1' to this bit enables the Start Frame Detector to generate an interrupt on interrupt vector RXC when a start of frame condition is detected.

**Bit 3 – LBME: Loop-back Mode Enable**
Writing this bit to '1' enables an internal connection between TxD and RxD pin.

**Bit 2 – ABEIE: Auto-baud Error Interrupt Enable**
Writing this bit to '1' enables the auto-baud error interrupt on interrupt vector RXC. The enabled interrupt will trigger for conditions where ISFIF flag is set.

**Bits 1:0 – RS485[1:0]: RS485 Mode**
These bits enable the RS485 and select the operation mode.

| Value | Name | Description |
|---|---|---|
| 0x0 | OFF | Disabled. |
| 0x1 | EXT | Enables RS485 mode with control of an external line driver through a dedicated Transmit Enable (TE) pin. |
| 0x2 | INT | Enables RS485 mode with control of the internal USART transmitter. |
| 0x3 | - | Reserved. |

### 24.5.7 Control B

**Name:** CTRLB
**Offset:** 0x06
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RXEN | TXEN | | SFDEN | ODME | RXMODE[1:0] | | MPCM |
| 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 – RXEN: Receiver Enable**
Writing this bit to '1' enables the USART Receiver. The Receiver will override normal port operation for the RxD pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FERR, BUFOVF, and PERR flags. In GENAUTO and LINAUTO mode, disabling the receiver will reset the auto-baud detection logic.

**Bit 6 – TXEN: Transmitter Enable**
Writing this bit to '1' enables the USART Transmitter. The Transmitter will override normal port operation for the TxD pin when enabled. Disabling the Transmitter (writing TXEN to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When the transmitter is disabled, it will no longer override the TxDn pin, and the pin direction is set as input automatically by hardware, even if it was configured as output by the user.

**Bit 4 – SFDEN: Start Frame Detection Enable**
Writing this bit to '1' enables the USART Start Frame Detection mode. The start frame detector is able to wake up the system from idle or standby sleep modes when a high (IDLE) to low (START) transition is detected on the RxD line.

**Bit 3 – ODME: Open Drain Mode Enable**
Writing this bit to '1' will make the TxD pin to have open-drain functionality. A pull-up resistor is needed to prevent the line from floating when a logic one is output to TxD pin.

**Bits 2:1 – RXMODE[1:0]: Receiver Mode**
In CLK2X mode, the divisor of the baud rate divider will be reduced from 16 to 8 effectively doubling the transfer rate for asynchronous communication modes. For synchronous operation the CLK2X mode has no effect and RXMODE should always be written to zero. RXMODE must be zero when the USART Communication Mode is configured to IRCOM. Setting RXMODE to GENAUTO enables generic auto-baud where the SYNC character is valid when eight low and high bits has been registered. In this mode any SYNC character that gives a valid BAUD rate will be accepted. In LINAUTO mode the SYNC character is constrained and found valid if each two bits falls within 32 +/- 6 baud samples of the internal baud rate and match data value 0x55. The GENAUTO and LINAUTO mode is only supported for USART operated in asynchronous slave mode.

| Value | Name | Description |
|---|---|---|
| 0x0 | NORMAL | Normal USART Mode, Standard Transmission Speed |
| 0x1 | CLK2X | Normal USART Mode, Double Transmission Speed |

© 2017 Microchip Technology Inc.

| Value | Name | Description |
|---|---|---|
| 0x2 | GENAUTO | Generic Auto-baud Mode |
| 0x3 | LINAUTO | LIN Constrained Auto-baud Mode |

**Bit 0 – MPCM: Multi-Processor Communication Mode**
Writing a '1' to this bit enables the Multi-Processor Communication mode: the USART Receiver ignores all the incoming frames that do not contain address information. The Transmitter is unaffected by the MPCM setting. For more detailed information see Multiprocessor Communication Mode.

### 24.5.8 Control C - Async Mode

This register description is valid for all modes except Master SPI mode. When the USART Communication Mode bits (CMODE) in this register are written to 'MSPI', see Control C - Master SPI Mode for the correct description.

**Name:** CTRLC
**Offset:** 0x07
**Reset:** 0x03
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CMODE[1:0] | | PMODE[1:0] | | SBMODE | CHSIZE[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 | 1 | 1 |

**Bits 7:6 – CMODE[1:0]: USART Communication Mode**
Writing these bits selects the communication mode of the USART.
Writing a 0x3 to these bits alters the available bit fields in this register, see Control C - Master SPI Mode- Master SPI Mode .

| Value | Name | Description |
|---|---|---|
| 0x0 | ASYNCHRONOUS | Asynchronous USART |
| 0x1 | SYNCHRONOUS | Synchronous USART |
| 0x2 | IRCOM | Infrared Communication |
| 0x3 | MSPI | Master SPI |

**Bits 5:4 – PMODE[1:0]: Parity Mode**
Writing these bits enables and selects the type of parity generation.

When enabled, the Transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data, compare it to the PMODE setting, and set the Parity Error flag (PERR) in the Status register (USART.STATUS) if a mismatch is detected.

| Value | Name | Description |
|---|---|---|
| 0x0 | DISABLED | Disabled |
| 0x1 | - | Reserved |
| 0x2 | EVEN | Enabled, Even Parity |
| 0x3 | ODD | Enabled, Odd Parity |

**Bit 3 – SBMODE: Stop Bit Mode**
Writing this bit selects the number of stop bits to be inserted by the Transmitter.
The Receiver ignores this setting.

| Value | Description |
|---|---|
| 0 | 1 stop bit |
| 1 | 2 stop bits |

**Bits 2:0 – CHSIZE[2:0]: Character Size**
Writing these bits select the number of data bits in a frame. The Receiver and Transmitter use the same setting. For 9BIT character size, the order of which byte to read or write first, low or high byte of RXDATA or TXDATA is selectable.

| Value | Name | Description |
|---|---|---|
| 0x0 | 5BIT | 5-bit |
| 0x1 | 6BIT | 6-bit |
| 0x2 | 7BIT | 7-bit |
| 0x3 | 8BIT | 8-bit |
| 0x4 | - | Reserved |
| 0x5 | - | Reserved |
| 0x6 | 9BIT | 9-bit (Low byte first) |
| 0x7 | 9BIT | 9-bit (High byte first) |

### 24.5.9 Control C - Master SPI Mode

This register description is only valid when the USART is in Master SPI mode (CMODE written to MSPI). For other CMODE values, see Control C - Async Mode the correct description.

See USART in Master SPI mode for full description of the Master SPI Mode operation.

**Name:** CTRLC
**Offset:** 0x07
**Reset:** 0x00
**Property:** -

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

**Bits 7:6 – CMODE[1:0]: USART Communication Mode**
Writing these bits selects the communication mode of the USART.
Writing a value different than 0x3 to these bits alters the available bit fields in this register, see Control C - Async Mode.

| Value | Name | Description |
|---|---|---|
| 0x0 | ASYNCHRONOUS | Asynchronous USART |
| 0x1 | SYNCHRONOUS | Synchronous USART |
| 0x2 | IRCOM | Infrared Communication |
| 0x3 | MSPI | Master SPI |

**Bit 2 – UDORD: Data Order**
Writing this bit selects the frame format.

The Receiver and Transmitter use the same setting. Changing the setting of UDORD will corrupt all ongoing communication for both receiver and transmitter.

| Value | Description |
|---|---|
| 0 | MSB of the data word is transmitted first |
| 1 | LSB of the data word is transmitted first |

**Bit 1 – UCPHA: Clock Phase**
The UCPHA bit setting determine if data is sampled on the leading (first) edge or tailing (last) edge of XCKn. Refer to the Master SPI Mode Clock Generation for details.

### 24.5.10 Baud Register

The USART.BAUDL and USART.BAUDH register pair represents the 16-bit value, USART.BAUD. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

Ongoing transmissions of the Transmitter and Receiver will be corrupted if the baud rate is changed. Writing this register will trigger an immediate update of the baud rate prescaler. For more information of how to set the baud rate, see Table 24-2

**Name:** BAUD
**Offset:** 0x08
**Reset:** 0x00
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | BAUD[15:8] | | | | | | | |
| 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 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | BAUD[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 15:8 – BAUD[15:8]: USART Baud Rate high byte**
These bits hold the MSB of the 16-bit Baud register.

**Bits 7:0 – BAUD[7:0]: USART Baud Rate low byte**
These bits hold the LSB of the 16-bit Baud register.

### 24.5.11 Debug Control Register

**Name:** DBGCTRL
**Offset:** 0x0B
**Reset:** 0x00
**Property:** -

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

**Bit 0 – DBGRUN: Debug Run**

| Value | Description |
|---|---|
| 0 | The peripheral is halted in break debug mode and ignores events. |
| 1 | The peripheral will continue to run in break debug mode when the CPU is halted. |

### 24.5.12 IrDA Control Register

**Name:** EVCTRL
**Offset:** 0x0C
**Reset:** 0x00
**Property:** -

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

**Bit 0 – IREI: IrDA Event Input Enable**
This bit enables the event source for the IRCOM Receiver. If event input is selected for the IRCOM Receiver, the input from the USART's RX pin is automatically disabled.

### 24.5.13 IRCOM Transmitter Pulse Length Control Register

**Name:** TXPLCTRL
**Offset:** 0x0D
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TXPL[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 – TXPL[7:0]: Transmitter Pulse Length**
The 8-bit value sets the pulse modulation scheme for the transmitter. Setting this register will only have effect if IRCOM mode is selected by a USART. By leaving this register value to zero, 3/16 of baud rate

period pulse modulation is used. Setting this value from 1 to 254 will give a fixed pulse length coding. The 8-bit value sets the number of system clock periods for the pulse. The start of the pulse will be synchronized with the rising edge of the baud rate clock. Setting the value to 255 (0xFF) will disable pulse coding, letting the RX and TX signals pass through the IRCOM Module unaltered. This enables other features through the IRCOM Module, such as half-duplex USART, Loop-back testing and USART RX input from an Event Channel.

TXPL must be configured before USART transmitter is enabled (TXEN).

### 24.5.14 IRCOM Receiver Pulse Length Control Register

**Name:** RXPLCTRL
**Offset:** 0x0E
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | RXPL[6:0] | | | | | | |
| 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 – RXPL[6:0]: Receiver Pulse Length**
The 8-bit value sets the filter coefficient for the IRCOM transceiver. Setting this register will only have effect if IRCOM mode is selected by a USART.
By leaving this register value to zero, filtering is disabled. Setting this value between 0x01 and 0xFF will enable filtering, where x+1 equal samples are required for the pulse to be accepted.

RXPL must be configured before USART receiver is enabled (RXEN).

# 25. SPI - Serial Peripheral Interface

## 25.1 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

## 25.2 Overview

The Serial Peripheral Interface (SPI) is a high-speed synchronous data transfer interface using three or four pins. It allows full duplex communication between an AVR device and peripheral devices or between several microcontrollers. The SPI peripheral can be configured as either Master or Slave. The master initiates and controls all data transactions.

The interconnection between master and slave devices with SPI is shown in the block diagram. The system consists of two shift registers and a master clock generator. The SPI master initiates the communication cycle by pulling the desired slave's slave select ($\overline{SS}$) signal low. 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 exchange data. Data is always shifted from master to slave on the master output, slave input (MOSI) line, and from slave to master on the master input, slave output (MISO) line.

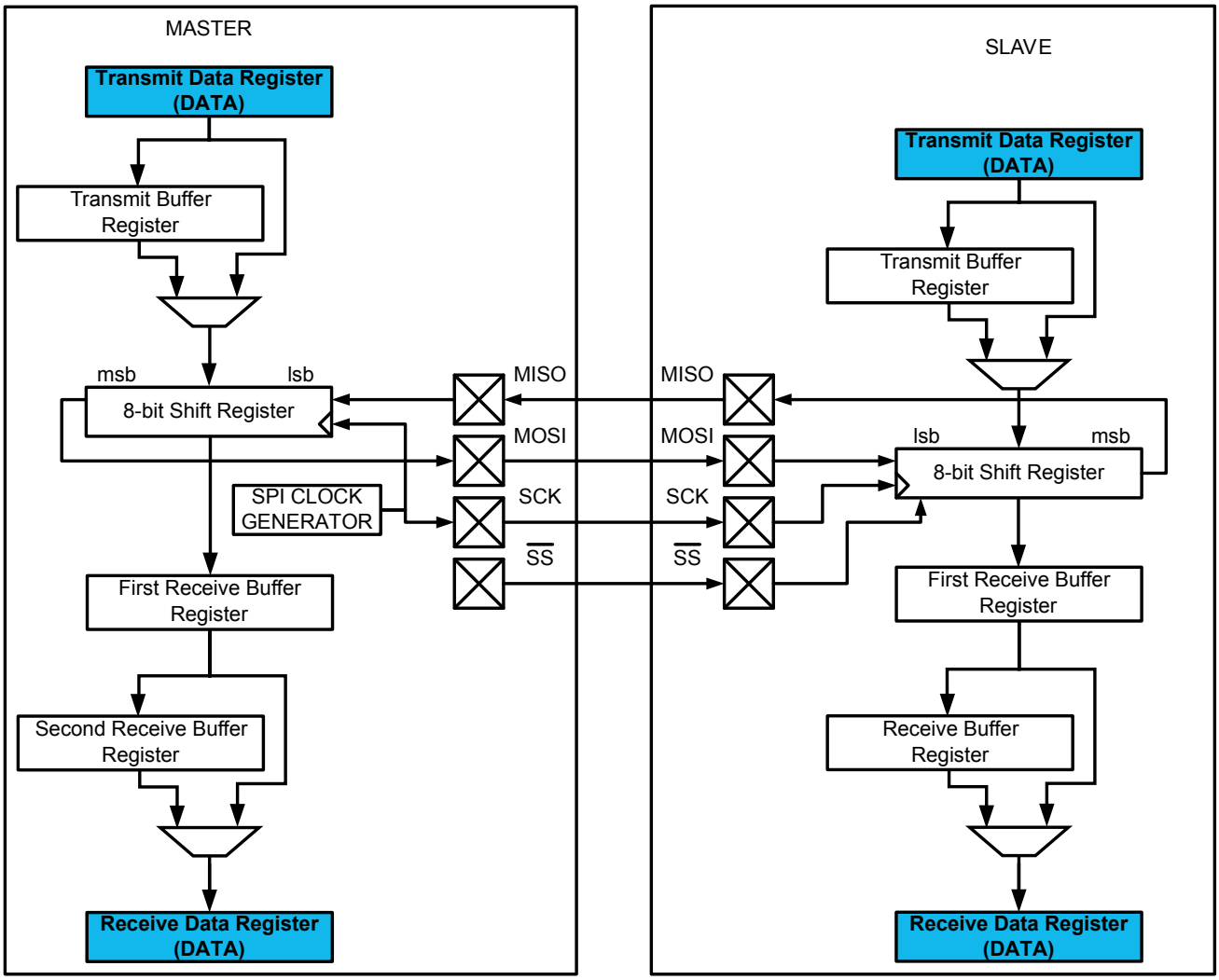
This device provides one instance of the SPI peripheral, SPI0.

**Related Links**

Block Diagram

### 25.2.1 Block Diagram

**Figure 25-1. SPI Block Diagram**




The SPI is built around an 8-bit shift register that will shift out and in data at the same time. The Transmit Data Register and the Receive Data Register are not physical registers, but are mapped to other registers when written or read: Writing the Transmit Data Register (DATA) will write the shift register in Normal mode and the Transmit Buffer Register in Buffer mode. Reading the Receive Data Register (DATA) will read the First Receive Buffer Register in normal mode and the Second Receive Data Register in buffer mode.

In master mode the SPI has a clock generator to generate the SCK clock. In slave mode the received SCK clock is synchronized and sampled to trigger the shifting of data in the shift register.

### 25.2.2 Signal Description

**Table 25-1. Signals in Master and Slave Mode**

| Signal | Description | Pin configuration | |
|---|---|---|---|
| | | Master Mode | Slave Mode |
| MOSI | Master Out Slave In | User defined | Input |
| MISO | Master In Slave Out | Input | User defined |

| Signal | Description | Pin configuration | |
|---|---|---|---|
| | | Master Mode | Slave Mode |
| SCK | Slave clock | User defined | Input |
| $\overline{SS}$ | Slave select | User defined | Input |

When the SPI module is enabled, the data direction of the MOSI, MISO, SCK, and SS pins is overridden according to Table 25-1.

The data direction of the pins with "User defined" pin configuration is not controlled by the SPI: The data direction is controlled by the application software configuring the Port peripheral. If these pins are configured with data direction as input, they can be used as regular I/O pin inputs. If these pins are configured with data direction as output, their output value is controlled by the SPI. The MISO pin has a special behavior: When the SPI is in slave mode and the MISO pin is configured as an output, the $\overline{SS}$ pin controls the output buffer of the pin: If $\overline{SS}$ is low, the output buffer drives the pin, if $\overline{SS}$ is high, the pin is tristated.

The data direction of the pins with "Input" pin configuration is controlled by the SPI hardware.

**Related Links**

I/O Multiplexing and Considerations

### 25.2.3 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 25-2. SPI System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | Yes | PORT |
| Interrupts | Yes | CPUINT |
| Events | Yes | EVSYS |
| Debug | Yes | UPDI |

**Related Links**

I/O Lines and Connections

Debug Operation

Interrupts

Clocks

#### 25.2.3.1 Clocks

This peripheral depends on the peripheral clock.

**Related Links**

CLKCTRL - Clock Controller

#### 25.2.3.2 I/O Lines and Connections

The SPI signals (MOSI, MISO, SCK, $\overline{SS}$) are either input or outputs, depending on whether the SPI is in Master or Slave Mode, as described in the Signal Description.

Using the I/O lines requires configuration of the I/O pins as described in the Signal Description.

**Related Links**

I/O Multiplexing and Considerations

PORT - I/O Pin Configuration

Signal Description

#### 25.2.3.3 Interrupts

Using the interrupts of this peripheral requires the Interrupt Controller to be configured first.

**Related Links**

CPUINT - CPU Interrupt Controller

SREG

Interrupts

#### 25.2.3.4 Events

Not applicable.

#### 25.2.3.5 Debug Operation

When run-time debugging, this peripheral will continue normal operation. Halting the CPU in debugging mode will halt normal operation of the peripheral.

If the peripheral is configured to require periodical service by the CPU through interrupts or similar, improper operation or data loss may result during halted debugging.

**Related Links**

UPDI - Unified Program and Debug Interface

## 25.3 Functional Description

### 25.3.1 Initialization

Initialize the SPI to a basic functional state by following these steps:

1. Configure the $\overline{SS}$ pin in the Port peripheral.
2. Select SPI Master / Slave operation by writing the Master/Slave Select bit (MASTER) in the Control A register (SPI.CTRLA).
3. In Master mode, select the clock speed by writing the Prescaler bits (PRESC) and the Clock Double bit (CLK2X) in SPI.CTRLA.
4. Optional: Select the data transfer mode by writing to the MODE bits in the Control B register (SPI.CTRLB).
5. Optional: Write the Data Order bit (DORD) in SPI.CTRLA.
6. Optional: Setup buffer mode by writing BUFEN and BUFWR bits in the Control B register (SPI.CTRLB).
7. Optional: To disable the multimaster support in Master mode, write '1' to the Slave Select Disable bit (SSD) in SPI.CTRLB.
8. Enable the SPI by writing a '1' to the ENABLE bit in SPI.CTRLA.

**Related Links**

I/O Multiplexing and Considerations

PORT - I/O Pin Configuration

Signal Description

### 25.3.2 Operation

#### 25.3.2.1 Master Mode Operation

When the SPI is configured in master mode, a write to the DATA register will start a new transfer. The SPI clock generator starts and the hardware shifts the eight bits into the selected slave. After the byte is shifted out the interrupt flag (INTFLAGS.IF) in SPI.INTFLAGS is set. The SPI Master can operate in two modes, Normal and Buffered, as explained below.

**$\overline{SS}$ Pin Functionality in Master Mode - Multimaster support**

In master mode, the Slave Select Disable bit (CTRLB.SSD) in Control Register B (SPI.CTRLB) controls how the SPI uses the $\overline{SS}$ line.

- If CTRLB.SSD is zero, the SPI can use the $\overline{SS}$ pin to transition from Master to Slave mode. This allows multiple SPI masters on the same SPI bus.
- If CTRLB.SSD is one, the SPI does not use the $\overline{SS}$ pin, and it can be used as a regular IO pin, or by other peripheral modules.

If the CTRLB.SSD bit is zero and the $\overline{SS}$ is configured as an output pin, it can be used as a regular IO pin, or by other peripheral modules, and will not affect the SPI system.

If the CTRLB.SSD bit is zero and the $\overline{SS}$ is configured as an input pin, the $\overline{SS}$ pin must be held high to ensure master SPI operation. A low level will be interpreted as another master trying to take control of the bus. This will switch the SPI into slave mode and the hardware of the SPI will perform the following actions:

1. The master bit (CTRLA.MASTER) in the SPI Control A Register (SPI.CTRLA) is cleared and the SPI system becomes a slave. The direction of the pins will be switched according to Table 25-3.
2. The Interrupt Flag (INTFLAGS.IF) in the Interrupt Flags Register (SPI.INTFLAGS) will be set. If the interrupt is enabled and the global interrupts are enabled the interrupt routine will be executed.

**Table 25-3. Overview of the $\overline{SS}$ Pin Functionality when CTRLB.SSD is zero**

| $\overline{SS}$ Configuration | $\overline{SS}$ Pin-level | Description |
|---|---|---|
| Input | High | Master activated (selected) |
| | Low | Master deactivated, switched to slave mode |
| Output | High | Master activated (selected) |
| | Low | |

**Note:**
If the AVR is configured for master mode and it can not be ensured that the $\overline{SS}$ pin will stay high between two transmissions, the status of the Master bit (CTRLA.MASTER) has to be checked before a new byte is written. After the Master bit has been cleared by a low level on the $\overline{SS}$ line, it must be set by the application to re-enable SPI master mode.

**Normal Mode**

In Normal mode, the system is single buffered in the transmit direction and double buffered in the receive direction. This influences the data handling in the following ways:

1. New bytes to be sent cannot be written to the data register (SPI.DATA) before the entire transfer has completed. A premature write will cause corruption of the transmitted data, and the hardware will set the Write Collision Flag (INTFLAGS.WRCOL).
2. Received bytes are written to First Receive Buffer Register immediately after the transmission is completed.

3. The First Receive Buffer Register has to be read before the next transmission is completed or data will be lost. This register is read by reading SPI.DATA.
4. The Transmit Buffer Register and Second Receive Buffer Register are not used in Normal Mode.

After a transfer has completed, the Interrupt Flag (INTFLAGS.IF) will be set in the Interrupt Flags Register (SPI.INTFLAGS). This will cause the corresponding interrupt to be executed if this interrupt and the global interrupts are enabled. Setting the Interrupt Enable (IE) bit in the Interrupt Control Register (SPI.INTCTRL) will enable the interrupt.

**Buffer Mode**

The Buffer Mode is enabled by setting the BUFEN bit in SPI.CTRLB. The BUFWR bit in SPI.CTRLB has no effect in Master mode. In Buffer Mode the system is double buffered in the transmit direction and triple buffered in the receive direction. This influences the data handling the following ways:

1. New bytes to be sent can be written to the data register (DATA) as long as the Data Register Empty Interrupt Flag (DREIF) in the Interrupt Flag Register (INTFLAGS) is set. The first write will be transmitted right away and a following write will go to the transmit buffer register.
2. A received byte is placed in a two-entry RX FIFO comprised of the First and Second Receive Buffer Registers immediately after the transmission is completed.
3. The Data Register is used to read from the RX FIFO. The RX FIFO must be read at least every second transfer to avoid any loss of data.

If both the Shift Register and the Transmit Buffer Register becomes empty, the Transfer Complete Interrupt Flag (TXCIF) in the Interrupt Flags Register (INTFLAGS) will be set. This will cause the corresponding interrupt to be executed if this interrupt and the global interrupts are enabled. Setting the Transfer Complete Interrupt Enable (TXCIE) in the Interrupt Control Register (INTCTRL) enables the Transfer Complete Interrupt.

#### 25.3.2.2 Slave Mode

In Slave Mode, the SPI peripheral receives SPI clock and Slave Select from a Master. Slave Mode supports three operational modes: One unbuffered mode and two buffered modes. In Slave Mode, the control logic will sample the incoming signal on the SCK pin. To ensure correct sampling of this clock signal, the minimum low and high periods must each be longer than two peripheral clock cycles.

**$\overline{SS}$ Pin Functionality in Slave Mode**

The Slave Select ($\overline{SS}$) pin plays a central role in operation of the SPI. Depending on the mode the SPI is in and the configuration of this pin, it can be used to activate or deactivate devices. The $\overline{SS}$ pin is used as a chip select pin.

In slave mode, $\overline{SS}$, MOSI and SCK are always inputs. The behavior of the MISO pin depends on the configured data direction of the pin in the port peripheral and the value of $\overline{SS}$: When $\overline{SS}$ is driven low, the SPI is activated and will respond to received SCK pulses by clocking data out on MISO if the user has configured the data direction of the MISO pin as output. When $\overline{SS}$ is driven high the SPI is deactivated, meaning that it will not receive incoming data. If the MISO pin data direction is configured as output, the MISO pin will be tristated. The following table shows an overview of the $\overline{SS}$ pin functionality.

**Table 25-4. Overview of the $\overline{SS}$ Pin Functionality**

| $\overline{SS}$ Configuration | $\overline{SS}$ Pin-level | Description | MISO pin mode | |
|---|---|---|---|---|
| | | | Port direction = output | Port direction = input |
| Always Input | High | Slave deactivated (deselected) | Tristated | Input |
| | Low | Slave activated (selected) | Output | Input |

**Note:**
In slave mode, the SPI state machine will be reset when the $\overline{SS}$ pin is brought high. If the $\overline{SS}$ is brought high during a transmission, the SPI will stop sending and receiving immediately and both data received and data sent must be considered as lost. As the $\overline{SS}$ pin is used to signal the start and end of a transfer, it is useful for achieving packet/byte synchronization, and keeping the slave bit counter synchronized with the master clock generator.

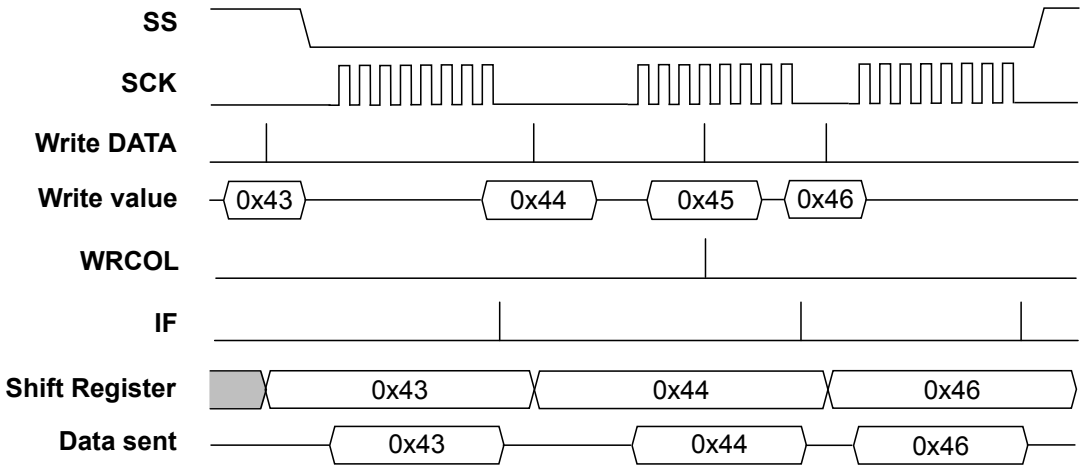
#### Normal Mode

In Normal Mode, the SPI peripheral will remain idle as long as the $\overline{SS}$ pin is driven high. In this state, software may update the contents of the DATA register, but the data will not be shifted out by incoming clock pulses on the SCK pin until the $\overline{SS}$ pin is driven low. If $\overline{SS}$ is driven low, the slave will start to shift out data on the first SCK clock pulse. When one byte has been completely shifted, the SPI Interrupt flag (IF) in SPI.INTFLAGS is set.

The user application may continue placing new data to be sent into the SPI.DATA register before reading the incoming data. New bytes to be sent cannot be written to SPI.DATA before the entire transfer has completed. A premature write will be ignored, and the hardware will set the Write Collision Flag (INTFLAGS.WRCOL).

When $\overline{SS}$ is driven high, the SPI logic is halted, and the SPI slave will not receive any new data. Any partially received packet in the shift register will be lost.

**Figure 25-2. SPI timing diagram in normal mode (buffer mode not enabled)**



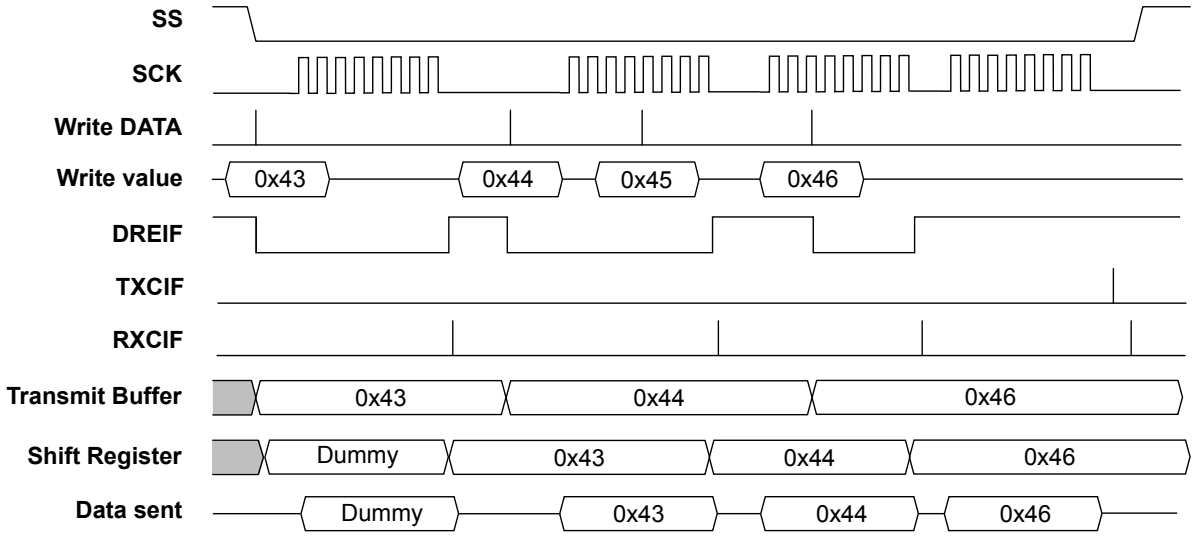

The figure shows three transfers and one write to the DATA register while the SPI is busy with a transfer. This write will be ignored and the Write Collision Flag (INTFLAGS.WRCOL) is set.

#### Buffer Mode

To avoid data collisions, the SPI peripheral can be configured in buffered mode by writing a '1' to the Buffer Mode Enable bit (CTRLB.BUFEN) in the Control B register (SPI.CTRLB). In this mode, the SPI has

additional interrupt flags and extra buffers. The extra buffers are shown in Figure 25-1. There are two different modes for the buffer mode, selected with the Buffer Mode Wait for Receive bit (BUFWR). The two different modes are described below with timing diagrams.

**Figure 25-3. SPI Timing Diagram in buffer mode with CTRLB.BUFWR set to zero**

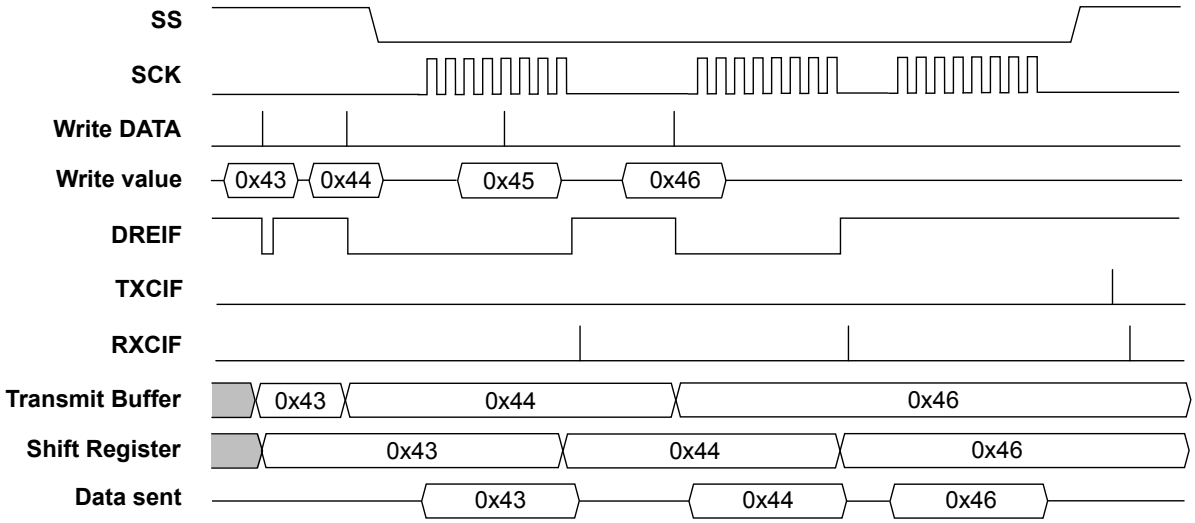



All writes to the Data Register (DATA) goes to the Transmit Buffer Register. As the figure above shows 0x43 is written to the Data Register (DATA), but it is not immediately transferred to the shift register so the first byte sent will be a dummy byte. The value of the dummy byte is whatever was in the shift register at the time, usually the last received byte. After the first dummy transfer is completed the value 0x43 is transferred to the shift register. Then 0x44 is written to the Data Register (DATA) and goes to the Transmit Buffer Register. A new transfer is started and 0x43 will be sent. The value 0x45 is written to the Data Register (DATA), but the Transmit Buffer Register is not updated since it is already full containing 0x44 and the Data Register Empty Interrupt Flag (INTFLAGS.DREIF) is low. The value 0x45 will be lost. After the transfer the value 0x44 is moved to the shift register. During the next transfer 0x46 is written to the Data Register (DATA) and 0x44 is sent out. After the transfer is complete 0x46 is copied into the shift register and it is sent out in the next transfer.

The Data Register Empty Interrupt Flag (INTFLAGS.DREIF) goes low every time the Transmit Buffer Register is written and goes high after a transfer when the previous value in the Transmit Buffer Register is copied into the shift register. The Receive Complete Interrupt Flag (INTFLAGS.RXCIF) is set one cycle after the Data Register Empty Interrupt Flag (INTFLAGS.DREIF) goes high. The Transfer Complete Interrupt Flag (INTFLAGS.TXCIF) is set one cycle after the Receive Complete Interrupt Flag (INTFLAGS.RXCIF) is set when both the value in the shift register and the Transmit Buffer Register has been sent.

© 2017 Microchip Technology Inc.

**Figure 25-4. SPI Timing Diagram in buffer mode with CTRLB.BUFWR set to one**




All writes to the Data Register (DATA) goes to the transmit buffer. As the figure above shows 0x43 is written to the Data Register (DATA) and since the Slave Select pin is high it is copied to the shift register the next cycle. Then the next write (0x44) will go to the Transmit Buffer Register. During the first transfer the value 0x43 will be shifted out. In the figure the value 0x45 is written to the Data Register (DATA), but the Transmit Buffer Register is not updated since the Data Register Empty Interrupt Flag (INTFLAGS.DREIF) is low. After the transfer is completed the value 0x44 form the Transmit Buffer Register is copied over to the shift register. The value 0x46 is written to the Transmit Buffer Register. During the next two transfers 0x44 and 0x46 is shifted out. The Flags behaves the same as with Buffer Mode Wait for Receive Bit (CTRLB.BUFWR) set to zero.

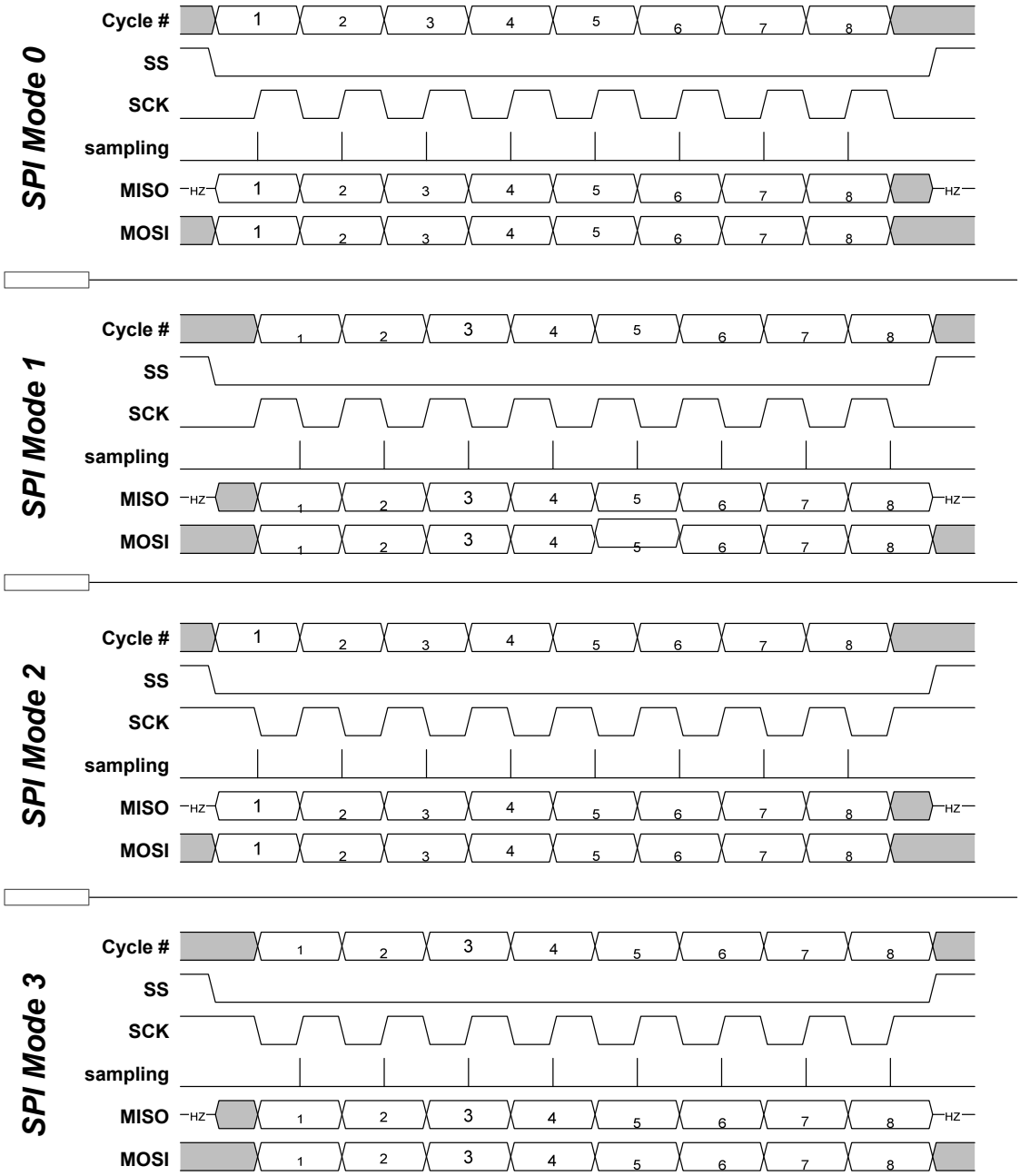
#### 25.3.2.3 Data Modes

There are four combinations of SCK phase and polarity with respect to serial data. The desired combination is selected by writing to the MODE bits in the Control B register (SPI.CTRLB).

The SPI data transfer formats are shown below. Data bits are shifted out and latched in on opposite edges of the SCK signal, ensuring sufficient time for data signals to stabilize.

The leading edge is the first clock edge of a clock cycle. The trailing edge is the last clock edge of a clock cycle.

**Figure 25-5. SPI Data Transfer Modes**




### 25.3.3 Interrupts

**Table 25-5. Available Interrupt Vectors and Sources**

| Offset | Name | Vector Description | Conditions |
|---|---|---|---|
| 0x00 | SPI | SPI interrupt | • SSI: Slave Select Trigger Interrupt<br>• DRE: Data Register Empty Interrupt<br>• TXC: Transfer Complete Interrupt<br>• RXC: Receive Complete Interrupt |

When an interrupt condition occurs, the corresponding Interrupt Flag is set in the Interrupt Flags register of the peripheral (*peripheral*.INTFLAGS).

An interrupt source is enabled or disabled by writing to the corresponding enable bit in the peripheral's Interrupt Control register (*peripheral*.INTCTRL).

An interrupt request is generated when the corresponding interrupt source is enabled and the Interrupt Flag is set. The interrupt request remains active until the Interrupt Flag is cleared. See the peripheral's INTFLAGS register for details on how to clear Interrupt Flags.

**Related Links**

SREG

CPUINT - CPU Interrupt Controller

### 25.3.4 Sleep Mode Operation

The SPI will continue working in Idle sleep mode. When entering any deeper sleep mode, an active transaction will be stopped.

**Related Links**

SLPCTRL - Sleep Controller

### 25.3.5 Configuration Change Protection

Not applicable.

## 25.4 Register Summary - SPI

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | | DORD | MASTER | CLK2X | | PRESC[1:0] | | ENABLE |
| 0x01 | CTRLB | 7:0 | BUFEN | BUFWR | | | | SSD | MODE[1:0] | |
| 0x02 | INTCTRL | 7:0 | RXCIE | TXCIE | DREIE | SSIE | | | | IE |
| 0x03 | INTFLAGS | 7:0 | RXCIF/IF | TXCIF/ WRCOL | DREIF | SSIF | | | | BUFOVF |
| 0x04 | DATA | 7:0 | DATA[7:0] | | | | | | | |

## 25.5 Register Description

### 25.5.1 Control A

**Name:** CTRLA
**Offset:** 0x00 [ID-000004d0]
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | DORD | MASTER | CLK2X | | PRESC[1:0] | | ENABLE |
| Access | | R/W | R/W | R/W | | R/W | R/W | R/W |
| Reset | | 0 | 0 | 0 | | 0 | 0 | 0 |

**Bit 6 – DORD: Data Order**

| Value | Description |
|---|---|
| 0 | The MSB of the data word is transmitted first. |
| 1 | The LSB of the data word is transmitted first. |

**Bit 5 – MASTER: Master/Slave Select**
Write this bit to configure SPI in desired mode.

If $\overline{SS}$ is configured as input and driven low while this bit is '1', this bit is cleared, and the IF flag in SPI.INTFLAGS is set. The user has to write MASTER=1 again to re-enable SPI Master mode.

This behavior is controlled by the Slave Select Disable bit (SSD) in SPI.CTRLB.

| Value | Description |
|---|---|
| 0 | SPI Slave mode selected |
| 1 | SPI Master mode selected |

**Bit 4 – CLK2X: Clock Double**
When this bit is written to '1' the SPI speed (SCK frequency, after internal prescaler) is doubled in Master mode.

| Value | Description |
|---|---|
| 0 | SPI speed (SCK frequency) is not doubled. |
| 1 | SPI speed (SCK frequency) is doubled in Master mode |

**Bits 2:1 – PRESC[1:0]: Prescaler**
This bit field controls the SPI clock rate configured in master mode. These bits have no effect in slave mode. The relationship between SCK and the peripheral clock frequency ($f_{CLK_PER}$) is shown below.

The output of the SPI prescaler can be doubled by writing the CLK2X bit to '1'.

| Value | Name | Description |
|---|---|---|
| 0x0 | DIV4 | CLK_PER/4 |
| 0x1 | DIV16 | CLK_PER/16 |
| 0x2 | DIV64 | CLK_PER/64 |
| 0x3 | DIV128 | CLK_PER/128 |

**Bit 0 – ENABLE: SPI Enable**

| Value | Description |
|---|---|
| 0 | SPI is disabled. |
| 1 | SPI is enabled. |

### 25.5.2 Control B

**Name:** CTRLB
**Offset:** 0x01 [ID-000004d0]
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | BUFEN | BUFWR | | | | SSD | MODE[1:0] | |
| Access | R/W | R/W | | | | R/W | R/W | R/W |
| Reset | 0 | 0 | | | | 0 | 0 | 0 |

**Bit 7 – BUFEN: Buffer Mode Enable**
Writing this bit to '1' enables Buffer Mode, meaning two buffers in receive direction, one buffer in transmit direction, and separate interrupt flags for both transmit complete and receive complete.

**Bit 6 – BUFWR: Buffer Mode Wait for Receive**
When writing this bit to '0' the first data transferred will be a dummy sample.

| Value | Description |
|---|---|
| 0 | One SPI transfer must be completed before the data is copied into the shift register. |
| 1 | When writing to the data register when the SPI is enabled and $\overline{SS}$ is high, the first write will go directly to the shift register. |

**Bit 2 – SSD: Slave Select Disable**
When this bit is set and when operating as SPI Master (MASTER=1 in SPI.CTRLA), $\overline{SS}$ does not disable Master Mode.

| Value | Description |
|---|---|
| 0 | Enable the Slave Select line when operating as SPI Master. |
| 1 | Disable the Slave Select line when operating as SPI Master. |

**Bits 1:0 – MODE[1:0]: Mode**
These bits select the transfer mode. The four combinations of SCK phase and polarity with respect to the serial data are shown in the table below. These bits decide whether the first edge of a clock cycle (leading edge) is rising or falling, and whether data setup and sample occur on the leading or trailing edge. When the leading edge is rising, the SCK signal is low when idle, and when the leading edge is falling, the SCK signal is high when idle.

| Value | Name | Description |
|---|---|---|
| 0x0 | 0 | Leading edge: Rising, sample<br>Trailing edge: Falling, setup |
| 0x1 | 1 | Leading edge: Rising, setup<br>Trailing edge: Falling, sample |
| 0x2 | 2 | Leading edge: Falling, sample<br>Trailing edge: Rising, setup |
| 0x3 | 3 | Leading edge: Falling, setup<br>Trailing edge: Rising, sample |

**Related Links**
Data Modes

### 25.5.3 Interrupt Control

**Name:** INTCTRL
**Offset:** 0x02 [ID-000004d0]
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RXCIE | TXCIE | DREIE | SSIE | | | | IE |
| Access | R/W | R/W | R/W | R/W | | | | R/W |
| Reset | 0 | 0 | 0 | 0 | | | | 0 |

**Bit 7 – RXCIE: Receive Complete Interrupt Enable**
In buffer mode this bit enables the receive complete interrupt. The enabled interrupt will be triggered when the RXCIF flag in the INTFLAG register is set. In non-buffer mode this bit is zero.

**Bit 6 – TXCIE: Transfer Complete Interrupt Enable**
In buffer mode this bit enables the transfer complete interrupt. The enabled interrupt will be triggered when the TXCIF flag in the INTFLAG register is set. In non-buffer mode this bit is zero.

**Bit 5 – DREIE: Data Register Empty Interrupt Enable**
In buffer mode this bit enables the data register empty interrupt. The enabled interrupt will be triggered when the DREIF flag in the INTFLAG register is set. In non-buffer mode this bit is zero.

**Bit 4 – SSIE: Slave Select Trigger Interrupt Enable**
In buffer mode this bit enables the Slave Select interrupt. The enabled interrupt will be triggered when the SSIF flag in the INTFLAG register is set. In non-buffer mode this bit is zero.

**Bit 0 – IE: Interrupt Enable**
This bit enables the SPI interrupt when the SPI is not in buffer mode. The enabled interrupt will be triggered when RXCIF/IF is set in the INTFLAG register.

### 25.5.4 Interrupt Flags

**Name:** INTFLAGS
**Offset:** 0x03 [ID-000004d0]
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RXCIF/IF | TXCIF/WRCOL | DREIF | SSIF | | | | BUFOVF |
| Access | R/W | R/W | R/W | R/W | | | | R/W |
| Reset | 0 | 0 | 0 | 0 | | | | 0 |

**Bit 7 – RXCIF/IF: Receive Complete Interrupt Flag/Interrupt Flag**
**RXCIF:** In buffer mode this flag is set when there is unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). In non-buffer mode this bit does not have any effect.

When interrupt-driven data reception is used, the receive complete interrupt routine must read the received data from DATA in order to clear RXCIF. If not, a new interrupt will occur directly after the return from the current interrupt. This flag can also be cleared by writing a one to its bit location.

**IF:** This flag is set when a serial transfer is complete and one byte is completely shifted in/out of the DATA register. If $\overline{SS}$ is configured as input and is driven low when the SPI is in master mode, this will also set this flag. IF is cleared by hardware when executing the corresponding interrupt vector. Alternatively, the IF flag can be cleared by first reading the SPI.INTFLAGS register when IF is set, and then accessing the DATA register.

**Bit 6 – TXCIF/WRCOL: Transfer Complete Interrupt Flag/Write Collision Flag**
**TXCIF:** In buffer mode this flag is set when all the data in the transmit shift register has been shifted out and there are no new data in the transmit buffer (DATA). The flag is cleared by writing a one to its bit location. In non-buffer mode this bit does not have any effect.

**WRCOL:** The WRCOL flag is set if the DATA register is written to before a complete byte has been shifted out. This flag is cleared by first reading the SPI.INTFLAGS register when WRCOL is set, and then accessing the DATA register.

**Bit 5 – DREIF: Data Register Empty Interrupt Flag**
In buffer mode this flag indicates whether the transmit buffer (DATA) is ready to receive new data. The flag is one when the transmit buffer is empty and zero when the transmit buffer contains data to be transmitted that has not yet been moved into the shift register. DREIF is set to '0' after a reset to indicate that the transmitter is ready. In non-buffer mode this bit is always zero.

DREIF is cleared by writing DATA. When interrupt-driven data transmission is used, the data register empty interrupt routine must either write new data to DATA in order to clear DREIF or disable the data register empty interrupt. If not, a new interrupt will occur directly after the return from the current interrupt.

**Bit 4 – SSIF: Slave Select Trigger Interrupt Flag**
In buffer mode this flag indicates that the SPI has been in master mode and the SS line has been pulled low externally so the SPI is now working in slave mode. The flag will only be set if the Slave Select Disable bit (SSD) is not '1'. The flag is cleared by writing a one to its bit location. In non-buffer mode this bit is always zero.

**Bit 0 – BUFOVF: Buffer Overflow**
This flash is only used in buffer mode. This flag indicates data loss due to a receiver buffer full condition. This flag is set if a buffer overflow condition is detected. A buffer overflow occurs when the receive buffer is full (two characters) and a third byte has been received in the shift register. If there is no transmit data the buffer overflow will not be set before the start of a new serial transfer. This flag is valid until the receive buffer (DATA) is read. Always write this bit location to zero when writing the SPI.INTFLAGS register. In non-buffer mode this bit is always zero

### 25.5.5 Data

**Name:** DATA
**Offset:** 0x04 [ID-000004d0]
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DATA[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 – DATA[7:0]: SPI Data**
The DATA register is used for sending and receiving data. Writing to the register initiates the data transmission, and the byte written to the register will be shifted out on the SPI output line.

Reading this register in buffer mode will read the second receive buffer and the contents of the first receive buffer will be moved to the second receive buffer.

# 26. TWI - Two Wire Interface

## 26.1 Features

- Bidirectional, two-wire communication interface
  - Philips $I^2C$ compatible
  - System Management Bus (SMBus) compatible
- Bus master and slave operation supported
  - Slave operation
  - Single bus master operation
  - Bus master in multi-master bus environment
  - Multi-master arbitration
- Flexible slave address match functions
  - 7-bit and general call address recognition in hardware
  - 10-bit addressing supported
  - Address mask register for dual address match or address range masking
  - Optional software address recognition for unlimited number of addresses
- Slave can operate in all sleep modes, including power-down
- Slave address match can wake device from all sleep modes
- Up to 1MHz bus frequency support
- Input filter for bus noise and spike suppression
- Support arbitration between start/repeated start and data bit (SMBus)
- Slave arbitration allows support for address resolution protocol (ARP) (SMBus)
- Supports SMBus Layer 1 timeouts
- Configurable timeout values

## 26.2 Overview

The Two-Wire Interface (TWI) peripheral is a bidirectional, two-wire communication interface. It is $I^2C$ and System Management Bus (SMBus) compatible. The only external hardware needed to implement the bus is one pull-up resistor on each bus line.

Any device connected to the bus must act as a master or a slave. The master initiates a data transaction by addressing a slave on the bus and telling whether it wants to transmit or receive data. One bus can have many slaves and one or several masters that can take control of the bus. An arbitration process handles priority if more than one master tries to transmit data at the same time. Mechanisms for resolving bus contention are inherent in the protocol.

The TWI peripheral supports master and slave functionality. The master and slave functionality are separated from each other, and can be enabled and configured separately. The master module supports multi-master bus operation and arbitration. It contains the baud rate generator. All 100kHz, 400kHz, and 1MHz bus frequencies are supported. Quick command and smart mode can be enabled to auto-trigger operations and reduce software complexity.

The slave module implements 7-bit address match and general address call recognition in hardware. 10-bit addressing is also supported. A dedicated address mask register can act as a second address match register or as a register for address range masking. The slave continues to operate in all sleep modes,

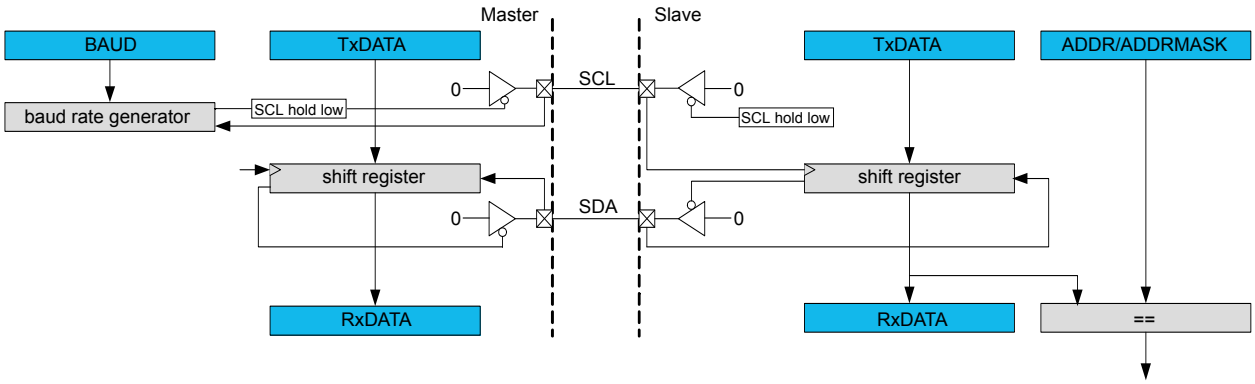
including power-down mode. This enables the slave to wake up the device from all sleep modes on TWI address match. It is possible to disable the address matching to let this be handled in software instead.

The TWI peripheral will detect START and STOP conditions, bus collisions, and bus errors. Arbitration lost, errors, collision, and clock hold on the bus are also detected and indicated in separate status flags available in both master and slave modes.

This device provides one instance of the TWI peripheral, TWI0.

### 26.2.1 Block Diagram

**Figure 26-1. TWI Block Diagram**




### 26.2.2 Signal Description

| Signal | Description | Type |
|---|---|---|
| SCL | Serial clock line | Digital I/O |
| SDA | Serial data line | Digital I/O |

**Related Links**

I/O Multiplexing and Considerations

### 26.2.3 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 26-1. TWI System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | Yes | PORT |
| Interrupts | Yes | CPUINT |
| Events | No | - |
| Debug | Yes | UPDI |

**Related Links**

Clocks

Debug Operation

I/O Lines and Connections

Interrupts

#### 26.2.3.1 Clocks

This peripheral requires the system clock (CLK_PER). The relationship between CLK_PER and the TWI bus clock (SCL) is explained in the TWI.MBAUD register.

**Related Links**

CLKCTRL - Clock Controller

MBAUD

#### 26.2.3.2 I/O Lines and Connections

Using the I/O lines of the peripheral requires configuration of the I/O pins.

#### 26.2.3.3 Interrupts

Using the interrupts of this peripheral requires the Interrupt Controller to be configured first.

#### 26.2.3.4 Events

Not applicable.

#### 26.2.3.5 Debug Operation

When run-time debugging, this peripheral will continue normal operation. Halting the CPU in debugging mode will halt normal operation of the peripheral.

This peripheral can be forced to operate with halted CPU by writing a '1' to the Debug Run bit (DBGRUN) in the Debug Control register of the peripheral (*peripheral*.DBGCTRL).

When the CPU is halted in debug mode and DBGRUN=1, that reading/writing the DATA register will neither trigger a bus operation nor cause transmit and clear flags.

If the peripheral is configured to require periodical service by the CPU through interrupts or similar, improper operation or data loss may result during halted debugging.

**Related Links**

UPDI - Unified Program and Debug Interface

## 26.3 Functional Description

### 26.3.1 Initialization

To start the TWI as Master, write a '1' to the ENABLE bit in the Master Control A register (TWI.MCTRLA), followed by writing the slave address to the Master Address (TWI.MADDR) register. The TWI.MADDR register also has a R/$\overline{W}$ bit which indicates whether the Master is transmitting or receiving. The Master DATA register (TWI.MDATA) is written in case master is transmitting data.

To enable the TWI as Slave, write the Slave Address (ADDR) in TWI.SADDR, and write a '1' to the ENABLE bit in the Slave Control A register (TWI.SCTRLA). The TWI peripheral will wait to receive a byte addressed to it.

### 26.3.2 General TWI Bus Concepts

The TWI provides a simple, bidirectional, two-wire communication bus consisting of a serial clock line (SCL) and a serial data line (SDA). The two lines are open-collector lines (wired-AND), and pull-up resistors (Rp) are the only external components needed to drive the bus. The pull-up resistors provide a high level on the lines when none of the connected devices are driving the bus.

The TWI bus is a simple and efficient method of interconnecting multiple devices on a serial bus. A device connected to the bus can be a master or slave, where the master controls the bus and all communication.

Figure 26-2 illustrates the TWI bus topology.

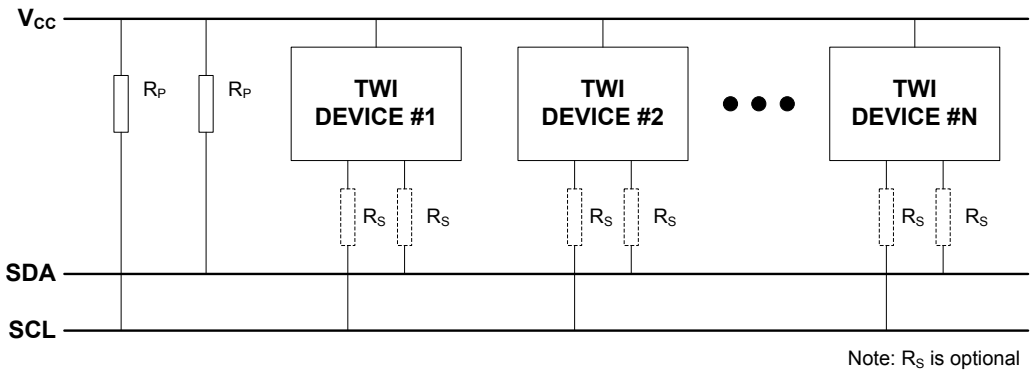



**Figure 26-2. TWI Bus Topology**

An unique address is assigned to all slave devices connected to the bus, and the master will use this to address a slave and initiate a data transaction.

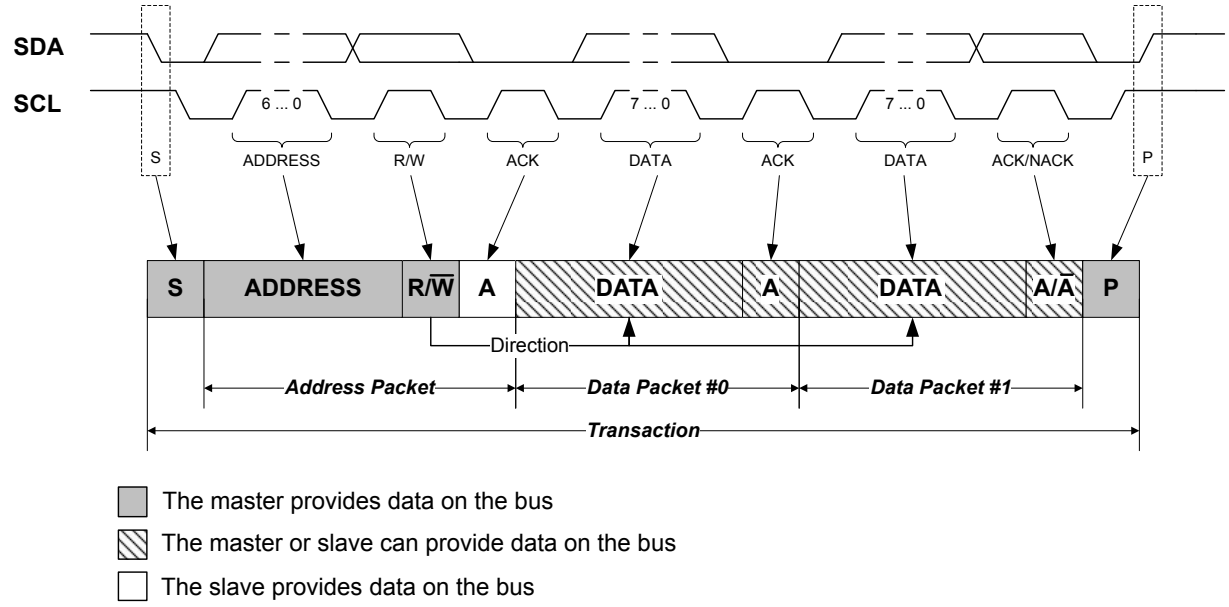
Several masters can be connected to the same bus, called a multi-master environment. An arbitration mechanism is provided for resolving bus ownership among masters, since only one master device may own the bus at any given time.

A device can contain both master and slave logic, and can emulate multiple slave devices by responding to more than one address.

A master indicates the start of a transaction by issuing a START condition (S) on the bus. An address packet with a slave address (ADDRESS) and an indication whether the master wishes to read or write data (R/$\overline{W}$) are then sent. After all data packets (DATA) are transferred, the master issues a STOP condition (P) on the bus to end the transaction. The receiver must acknowledge (A) or not-acknowledge ($\overline{A}$) each byte received.

Figure 26-3 shows a TWI transaction.

**Figure 26-3. Basic TWI Transaction Diagram Topology for a 7-bit Address Bus**

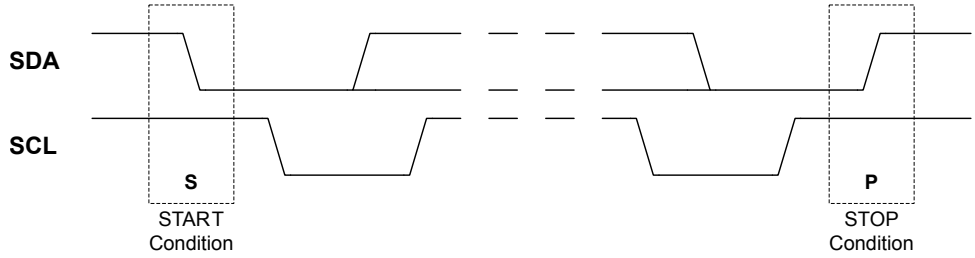



The master provides the clock signal for the transaction, but a device connected to the bus is allowed to stretch the low-level period of the clock to decrease the clock speed.

#### 26.3.2.1 START and STOP Conditions

Two unique bus conditions are used for marking the beginning (START) and end (STOP) of a transaction. The master issues a START condition (S) by indicating a high-to-low transition on the SDA line while the SCL line is kept high. The master completes the transaction by issuing a STOP condition (P), indicated by a low-to-high transition on the SDA line while SCL line is kept high.

**Figure 26-4. START and STOP Conditions**

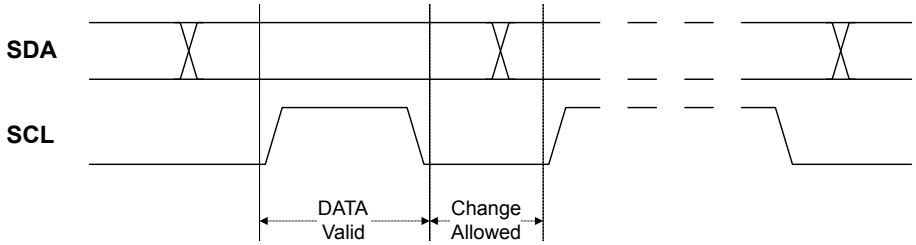



Multiple START conditions can be issued during a single transaction. A START condition that is not directly following a STOP condition is called a repeated START condition (Sr).

#### 26.3.2.2 Bit Transfer

As illustrated by Figure 26-5, a bit transferred on the SDA line must be stable for the entire high period of the SCL line. Consequently the SDA value can only be changed during the low period of the clock. This is ensured in hardware by the TWI module.

**Figure 26-5. Data Validity**




Combining bit transfers results in the formation of address and data packets. These packets consist of eight data bits (one byte) with the most-significant bit transferred first, plus a single-bit not-acknowledge (NACK) or acknowledge (ACK) response. The addressed device signals ACK by pulling the SCL line low during the ninth clock cycle, and signals NACK by leaving the line SCL high.

#### 26.3.2.3 Address Packet

After the START condition, a 7-bit address followed by a read/write (R/$\overline{W}$) bit is sent. This is always transmitted by the master. A slave recognizing its address will ACK the address by pulling the data line low for the next SCL cycle, while all other slaves should keep the TWI lines released and wait for the next START and address. The address, R/$\overline{W}$ bit, and acknowledge bit combined is the address packet. Only one address packet for each START condition is allowed, also when 10-bit addressing is used.

The R/$\overline{W}$ bit specifies the direction of the transaction. If the R/$\overline{W}$ bit is low, it indicates a master write transaction, and the master will transmit its data after the slave has acknowledged its address. If the R/$\overline{W}$ bit is high, it indicates a master read transaction, and the slave will transmit its data after acknowledging its address.

#### 26.3.2.4 Data Packet

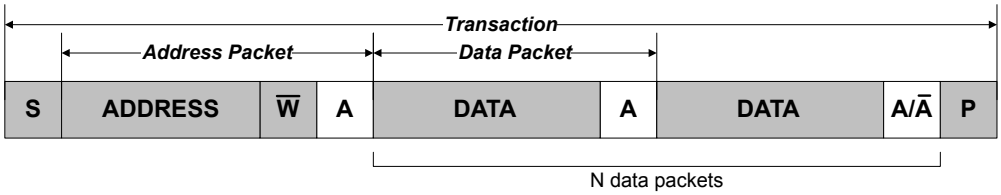
An address packet is followed by one or more data packets. All data packets are nine bits long, consisting of one data byte and an acknowledge bit. The direction bit in the previous address packet determines the direction in which the data are transferred.

#### 26.3.2.5 Transaction

A transaction is the complete transfer from a START to a STOP condition, including any repeated START conditions in between. The TWI standard defines three fundamental transaction modes: Master write, master read, and a combined transaction.

Figure 26-6 illustrates the master write transaction. The master initiates the transaction by issuing a START condition (S) followed by an address packet with the direction bit set to zero (ADDRESS+$\overline{W}$).

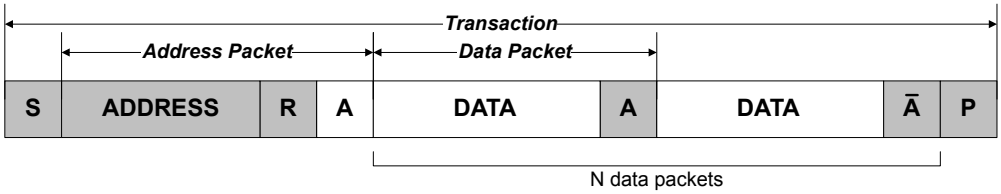
**Figure 26-6. Master Write Transaction**




Assuming the slave acknowledges the address, the master can start transmitting data (DATA) and the slave will ACK or NACK (A/$\overline{A}$) each byte. If no data packets are to be transmitted, the master terminates the transaction by issuing a STOP condition (P) directly after the address packet. There are no limitations to the number of data packets that can be transferred. If the slave signals a NACK to the data, the master must assume that the slave cannot receive any more data and terminate the transaction.

Figure 26-7 illustrates the master read transaction. The master initiates the transaction by issuing a START condition followed by an address packet with the direction bit set to one (ADDRESS+R). The addressed slave must acknowledge the address for the master to be allowed to continue the transaction.

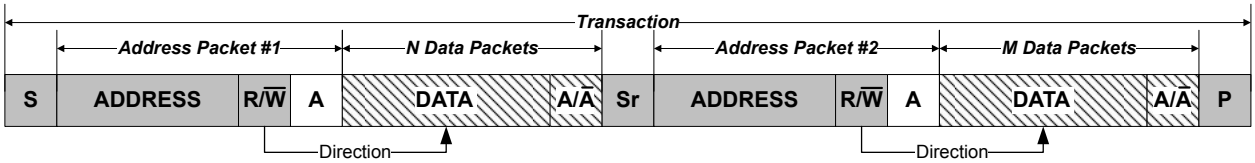
**Figure 26-7. Master Read Transaction**




Assuming the slave acknowledges the address, the master can start receiving data from the slave. There are no limitations to the number of data packets that can be transferred. The slave transmits the data while the master signals ACK or NACK after each data byte. The master terminates the transfer with a NACK before issuing a STOP condition.

Figure 26-8 illustrates a combined transaction. A combined transaction consists of several read and write transactions separated by repeated START conditions (Sr).

**Figure 26-8. Combined Transaction**

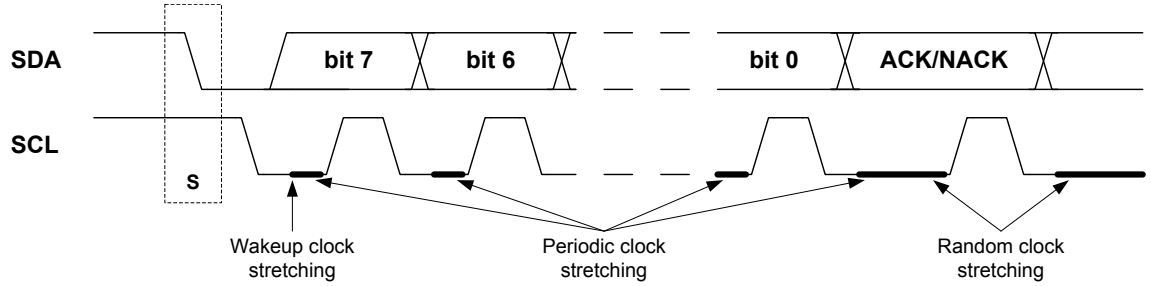



#### 26.3.2.6 Clock and Clock Stretching

All devices connected to the bus are allowed to stretch the low period of the clock to slow down the overall clock frequency or to insert wait states while processing data. A device that needs to stretch the clock can do this by holding/forcing the SCL line low after it detects a low level on the line.

Three types of clock stretching can be defined, as shown in Figure 26-9.

**Figure 26-9. Clock Stretching (1)**




**Note:** Clock stretching is not supported by all $I^2C$ slaves and masters.

If a slave device is in sleep mode and a START condition is detected, the clock stretching normally works during the wake-up period. For AVR devices, the clock stretching will be either directly before or after the ACK/NACK bit, as AVR devices do not need to wake up for transactions that are not addressed to it.

A slave device can slow down the bus frequency by stretching the clock periodically on a bit level. This allows the slave to run at a lower system clock frequency. However, the overall performance of the bus will be reduced accordingly. Both the master and slave device can randomly stretch the clock on a byte level basis before and after the ACK/NACK bit. This provides time to process incoming or prepare outgoing data, or perform other time-critical tasks.

In the case where the slave is stretching the clock, the master will be forced into a wait state until the slave is ready, and vice versa.

#### 26.3.2.7 Arbitration

A master can start a bus transaction only if it has detected that the bus is idle. As the TWI bus is a multi-master bus, it is possible that two devices may initiate a transaction at the same time. This results in multiple masters owning the bus simultaneously. This is solved using an arbitration scheme where the master loses control of the bus if it is not able to transmit a high level on the SDA line. The masters who lose arbitration must then wait until the bus becomes idle (i.e., wait for a STOP condition) before attempting to reacquire bus ownership. Slave devices are not involved in the arbitration procedure.

**Figure 26-10. TWI Arbitration**

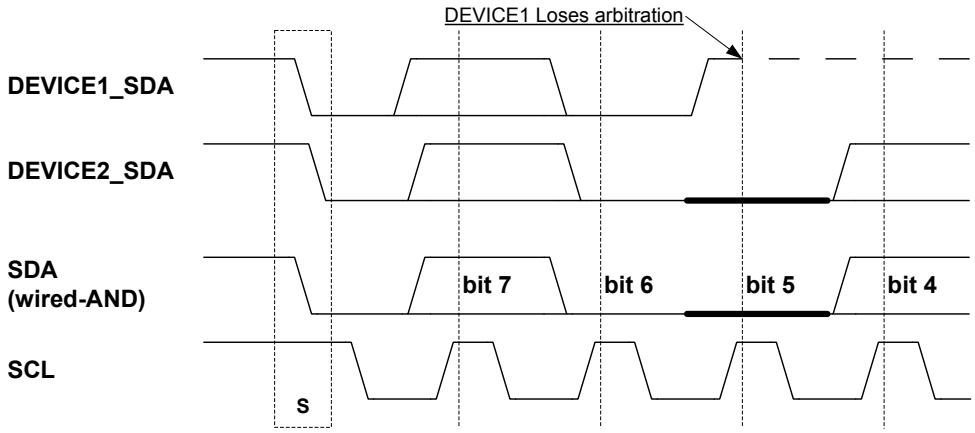


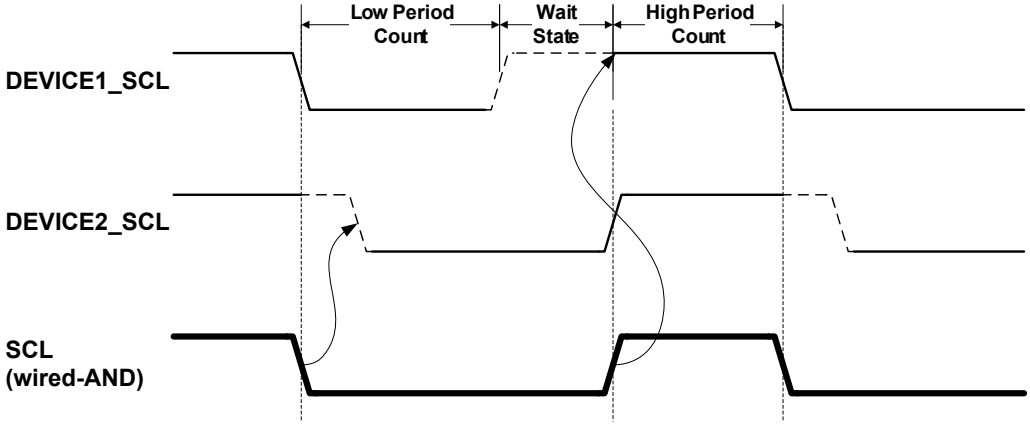

Figure 26-10 shows an example where two TWI masters are contending for bus ownership. Both devices are able to issue a START condition, but DEVICE1 loses arbitration when attempting to transmit a high level (bit 5) while DEVICE2 is transmitting a low level.

Arbitration between a repeated START condition and a data bit, a STOP condition and a data bit, or a repeated START condition and a STOP condition are not allowed and will require special handling by software.

#### 26.3.2.8 Synchronization

A clock synchronization algorithm is necessary for solving situations where more than one master is trying to control the SCL line at the same time. The algorithm is based on the same principles used for the clock stretching previously described. Figure 26-11shows an example where two masters are competing for control over the bus clock. The SCL line is the wired-AND result of the two masters clock outputs.

**Figure 26-11. Clock Synchronization**



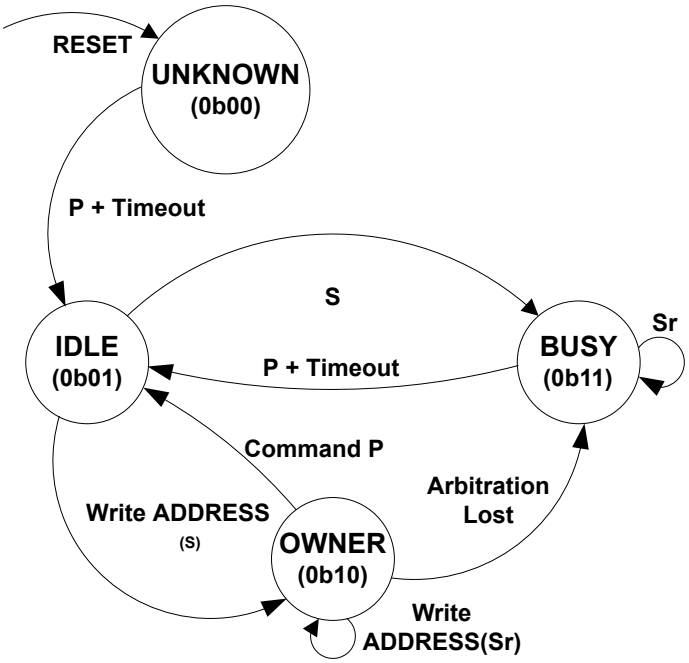

A high-to-low transition on the SCL line will force the line low for all masters on the bus, and they will start timing their low clock period. The timing length of the low clock period can vary among the masters. When a master (DEVICE1 in this case) has completed its low period, it releases the SCL line. However, the SCL line will not go high until all masters have released it. Consequently, the SCL line will be held low by the device with the longest low period (DEVICE2). Devices with shorter low periods must insert a wait state until the clock is released. All masters start their high period when the SCL line is released by all devices and has gone high. The device which first completes its high period (DEVICE1) forces the clock line low, and the procedure is then repeated. The result is that the device with the shortest clock period determines the high period, while the low period of the clock is determined by the device with the longest clock period.

### 26.3.3 TWI Bus State Logic

The bus state logic continuously monitors the activity on the TWI bus lines when the master is enabled. It continues to operate in all sleep modes, including power-down.

The bus state logic includes START and STOP condition detectors, collision detection, inactive bus timeout detection, and a bit counter. These are used to determine the bus state. Software can get the current bus state by reading the bus state bits in the master status register. The bus state can be unknown, idle, busy, or owner, and is determined according to the state diagram shown in Figure 26-12. The values of the bus state bits according to state are shown in binary in the figure.

**Figure 26-12. Bus State, State Diagram**




After a system reset and/or TWI master enable, the bus state is unknown. The bus state machine can be forced to enter idle by writing to the bus state bits accordingly. If no state is set by the application software, the bus state will become idle when the first STOP condition is detected. If the master inactive bus timeout is enabled, the bus state will change to idle on the occurrence of a timeout. After a known bus state is established, only a system reset or disabling of the TWI master will set the state to unknown.

When the bus is idle, it is ready for a new transaction. If a START condition generated externally is detected, the bus becomes busy until a STOP condition is detected. The STOP condition will change the bus state to idle. If the master inactive bus timeout is enabled, the bus state will change from busy to idle on the occurrence of a timeout.

If a START condition is generated internally while in idle state, the owner state is entered. If the complete transaction was performed without interference, i.e., no collisions are detected, the master will issue a STOP condition and the bus state will change back to idle. If a collision is detected, the arbitration is assumed lost and the bus state becomes busy until a STOP condition is detected. A repeated START condition will only change the bus state if arbitration is lost during the issuing of the repeated START. Arbitration during repeated START can be lost only if the arbitration has been ongoing since the first START condition. This happens if two masters send the exact same ADDRESS+DATA before one of the masters issues a repeated START (Sr).

### 26.3.4 Operation

#### 26.3.4.1 Electrical Characteristics

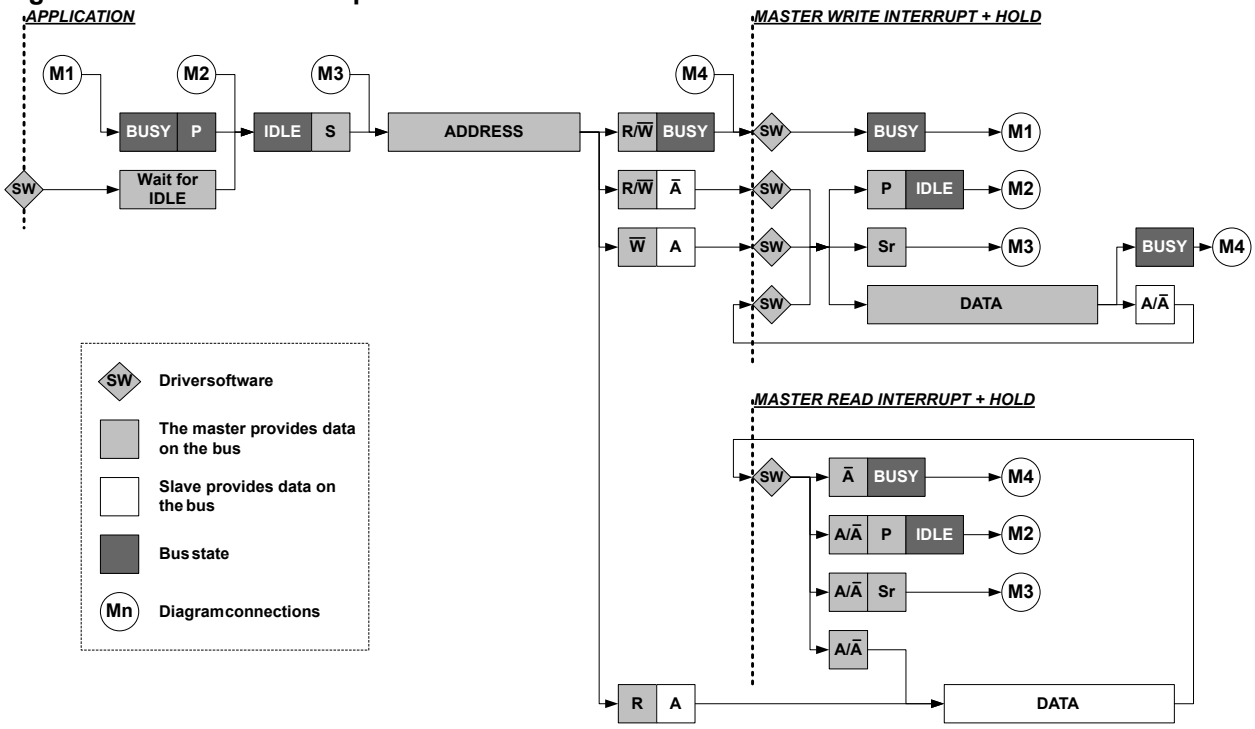
The TWI module in AVR devices follows the electrical specifications and timing of $I^2C$ bus and SMBus. These specifications are not 100% compliant, and so to ensure correct behavior, the inactive bus timeout period should be set in TWI master mode. Refer to TWI Master Operation for more details.

#### 26.3.4.2 TWI Master Operation

The TWI master is byte-oriented, with an optional interrupt after each byte. There are separate interrupt flags for master write and master read. Interrupt flags can also be used for polled operation. There are dedicated status flags for indicating ACK/NACK received, bus error, arbitration lost, clock hold, and bus state.

When an interrupt flag is set, the SCL line is forced low. This will give the master time to respond or handle any data, and will in most cases require software interaction. Figure 26-13 shows the TWI master operation. The diamond shaped symbols (SW) indicate where software interaction is required. Clearing the interrupt flags releases the SCL line.

**Figure 26-13. TWI Master Operation**



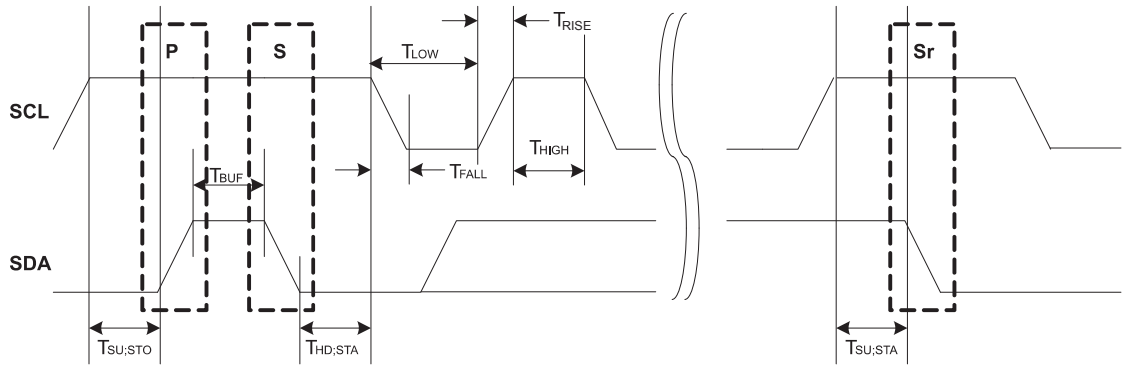

The number of interrupts generated is kept to a minimum by automatic handling of most conditions.

### Clock Generation

The BAUD must be set to a value that results in a TWI bus clock frequency ($f_{SCL}$) equal or less than 100kHz/400kHz/1MHz, dependent on the mode used by the application (Standard mode Sm/Fast mode Fm/Fast mode plus Fm+).

The low ($T_{LOW}$) and high ($T_{HIGH}$) times are determined by the Baud Rate register (BAUD), while the rise ($T_{RISE}$) and fall ($T_{FALL}$) times are determined by the bus topology. Because of the wired-AND logic of the bus, $T_{FALL}$ will be considered as part of $T_{LOW}$. Likewise, $T_{RISE}$ will be in a state between $T_{LOW}$ and $T_{HIGH}$ until a high state has been detected.

**Figure 26-14. SCL Timing**




- $T_{LOW}$ – Low period of SCL clock
- $T_{SU;STO}$ – Set-up time for stop condition
- $T_{BUF}$ – Bus free time between stop and start conditions
- $T_{HD;STA}$ – Hold time (repeated) start condition
- $T_{SU;STA}$ – Set-up time for repeated start condition
- $T_{HIGH}$ is timed using the SCL high time count from TWI.MBAUD
- $T_{RISE}$ is determined by the bus impedance; for internal pull-ups. Refer to *Electrical Characteristics*.
- $T_{FALL}$ is determined by the open-drain current limit and bus impedance; can typically be regarded as zero. Refer to *Electrical Characteristics* for details.

The SCL frequency is given by:

$$f_{\mathrm{SCL}} = \frac{1}{T_{\mathrm{LOW}} + T_{\mathrm{HIGH}} + T_{\mathrm{RISE}}}$$

The TWI.MBAUD value is used to time both SCL high and SCL low which gives the following formula of SCL frequency:

$$f_{\mathrm{SCL}} = \frac{f_{\mathrm{CLK_PER}}}{10 + 2BAUD + f_{\mathrm{CLK_PER}} \cdot T_{\mathrm{RISE}}}$$

If the TWI is in Fm+ mode, only TWI.MBAUD value of 3 or higher is supported. This means that for Fm+ mode to achieve baud rate of 1MHz, the peripheral clock (CLK_PER) has to run at 16MHz or faster.

**Transmitting Address Packets**

After issuing a START condition, the master starts performing a bus transaction when the master address register is written with the 7-bit slave address and direction bit. If the bus is busy, the TWI master will wait until the bus becomes idle before issuing the START condition.

Depending on arbitration and the R/$\overline{W}$ direction bit, one of four distinct cases (M1 to M4) arises following the address packet. The different cases must be handled in software.

***Case M1: Arbitration Lost or Bus Error during Address Packet***

If arbitration is lost during the sending of the address packet, the Master Write Interrupt Flag (WIF in TWI.MSTATUS) and Arbitration Lost Flag (ARBLOST in TWI.MSTATUS) are both set. Serial data output to the SDA line is disabled, and the SCL line is released. The master is no longer allowed to perform any operation on the bus until the bus state has changed back to idle.

A bus error will behave in the same way as an arbitration lost condition, but the Bus Error Flag (BUSERR in TWI.MSTATUS) is set in addition to the write interrupt and arbitration lost flags.

***Case M2: Address Packet Transmit Complete - Address not Acknowledged by Slave***
If no slave device responds to the address, the Master Write Interrupt Flag (WIF in TWI.MSTATUS) and the Master Received Acknowledge Flag (RXACK in TWI.MSTATUS) are set. The RXACK flag reflects the physical state of the ACK bit, i.e. no slave did pull the ACK bit low. The clock hold is active at this point, preventing further activity on the bus.

***Case M3: Address Packet Transmit Complete - Direction Bit Cleared***
If the master receives an ACK from the slave, the Master Write Interrupt Flag (WIF in TWI.MSTATUS) is set and the Master Received Acknowledge Flag (RXACK in TWI_MSTATUS) is cleared. The clock hold is active at this point, preventing further activity on the bus.

***Case M4: Address Packet Transmit Complete - Direction Bit Set***
If the master receives an ACK from the slave, the master proceeds to receive the next byte of data from the slave. When the first data byte is received, the Master Read Interrupt Flag (RIF in TWI.MSTATUS) is set and the Master Received Acknowledge Flag (RXACK in TWI.MSTATUS) is cleared. The clock hold is active at this point, preventing further activity on the bus.

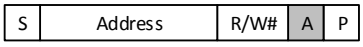
**Transmitting Data Packets**
The slave will know when an address packet with R/$\overline{W}$ direction bit set has been successfully received. It can then start sending data by writing to the slave data register. When a data packet transmission is completed, the data interrupt flag is set. If the master indicates NACK, the slave must stop transmitting data and expect a STOP or repeated START condition.

**Receiving Data Packets**
The slave will know when an address packet with R/$\overline{W}$ direction bit cleared has been successfully received. After acknowledging this, the slave must be ready to receive data. When a data packet is received, the data interrupt flag is set and the slave must indicate ACK or NACK. After indicating a NACK, the slave must expect a STOP or repeated START condition.

**Quick Command Mode**
With Quick Command enabled (QCEN in MCTRLA), the R/W# bit of the slave address denotes the command. This is a SMBus specific command where the R/W# bit may be used to simply turn a device function on or off, or enable/disable a low power standby mode. There is no data sent or received.

After the master receives an acknowledge from the slave, either RIF or WIF flag in MSTATUS will be set depending on polarity of R/W#. When either RIF or WIF flag is set after issuing a Quick Command, the TWI will accept a stop command through writing the CMD bits in MCTRLB.

**Figure 26-15. Quick Command Frame Format**

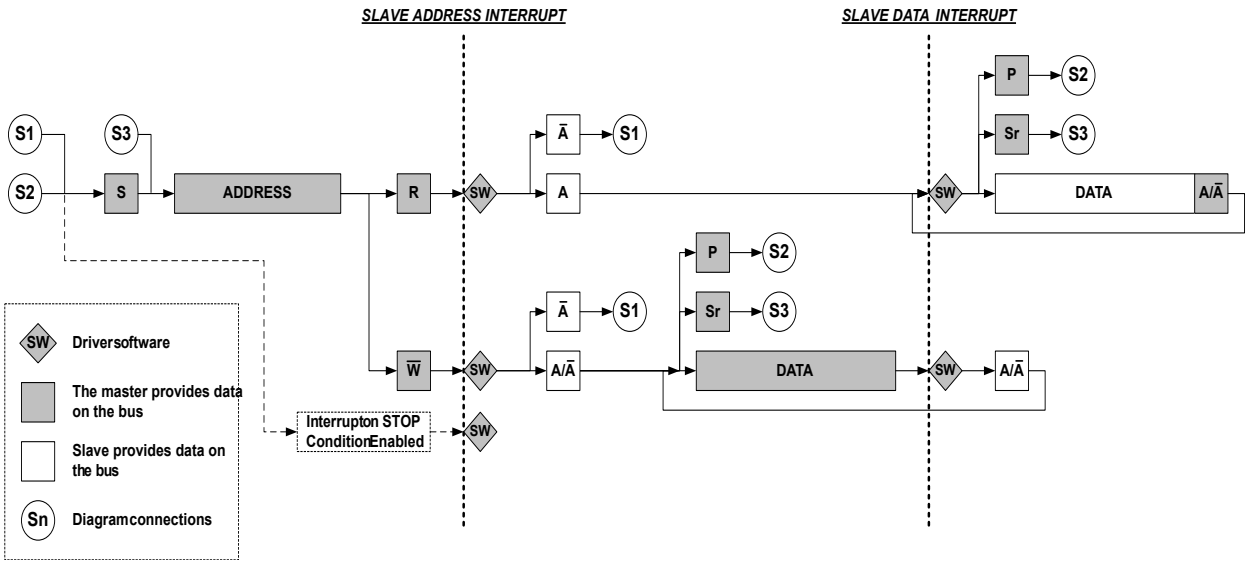



### 26.3.4.3 TWI Slave Operation

The TWI slave is byte-oriented with optional interrupts after each byte. There are separate slave data and address/stop interrupt flags. Interrupt flags can also be used for polled operation. There are dedicated status flags for indicating ACK/NACK received, clock hold, collision, bus error, and read/write direction.

When an interrupt flag is set, the SCL line is forced low. This will give the slave time to respond or handle data, and will in most cases require software interaction. Figure 26-16. shows the TWI slave operation. The diamond shapes symbols (SW) indicate where software interaction is required.

**Figure 26-16. TWI Slave Operation**




The number of interrupts generated is kept to a minimum by automatic handling of most conditions. Quick command can be enabled to auto-trigger operations and reduce software complexity.

Address Recognition Mode can be enabled to allow the slave to respond to all received addresses.

**Receiving Address Packets**

When the TWI slave is properly configured, it will wait for a START condition to be detected. When this happens, the successive address byte will be received and checked by the address match logic, and the slave will ACK a correct address and store the address in the DATA register. If the received address is not a match, the slave will not acknowledge and store address, and will wait for a new START condition.

The slave address/stop interrupt flag is set when a START condition succeeded by a valid address byte is detected. A general call address will also set the interrupt flag.

A START condition immediately followed by a STOP condition is an illegal operation, and the bus error flag is set.

The R/$\overline{W}$ direction flag reflects the direction bit received with the address. This can be read by software to determine the type of operation currently in progress.

Depending on the R/$\overline{W}$ direction bit and bus condition, one of four distinct cases (S1 to S4) arises following the address packet. The different cases must be handled in software.

***Case S1: Address Packet Accepted - Direction Bit Set***

If the R/$\overline{W}$ direction flag is set, this indicates a master read operation. The SCL line is forced low by the slave, stretching the bus clock. If ACK is sent by the slave, the slave hardware will set the data interrupt flag indicating data is needed for transmit. Data, repeated START, or STOP can be received after this. If NACK is sent by the slave, the slave will wait for a new START condition and address match.

***Case S2: Address Packet Accepted - Direction Bit Cleared***

If the R/$\overline{W}$ direction flag is cleared, this indicates a master write operation. The SCL line is forced low, stretching the bus clock. If ACK is sent by the slave, the slave will wait for data to be received. Data, repeated START, or STOP can be received after this. If NACK is sent, the slave will wait for a new START condition and address match.

***Case S3: Collision***
If the slave is not able to send a high level or NACK, the collision flag is set, and it will disable the data and acknowledge output from the slave logic. The clock hold is released. A START or repeated START condition will be accepted.

***Case S4: STOP Condition Received***
When the STOP condition is received, the slave address/stop flag will be set, indicating that a STOP condition, and not an address match, occurred.

**Receiving Data Packets**
The slave will know when an address packet with R/$\overline{W}$ direction bit cleared has been successfully received. After acknowledging this, the slave must be ready to receive data. When a data packet is received, the data interrupt flag is set and the slave must indicate ACK or NACK. After indicating a NACK, the slave must expect a STOP or repeated START condition.

**Transmitting Data Packets**
The slave will know when an address packet with R/$\overline{W}$ direction bit set has been successfully received. It can then start sending data by writing to the slave data register. When a data packet transmission is completed, the data interrupt flag is set. If the master indicates NACK, the slave must stop transmitting data and expect a STOP or repeated START condition.

#### 26.3.4.4 Smart Mode

The TWI interface has a smart mode that simplifies application code and minimizes the user interaction needed to adhere to the $I^2C$ protocol. For TWI Master, smart mode accomplishes this by automatically sending an ACK as soon as data register TWI.MDATA is read. This feature is only active when the ACKACT bit in TWI.MCTRLA register is set to ACK. If ACKACT is set to NACK, the TWI Master will not generate a NACK bit followed by reading the data register.

With smart mode enabled for TWI Slave (SMEN bit in TWI.SCTRLA), DIF (Data Interrupt Flag) will automatically be cleared if data register (TWI.SDATA) is read or written.

### 26.3.5 Events

Not applicable.

### 26.3.6 Interrupts

**Table 26-2. Available Interrupt Vectors and Sources**

| Offset | Name | Vector Description | Conditions |
|---|---|---|---|
| 0x00 | Slave | TWI Slave interrupt | • DIF: Data Interrupt Flag in SSTATUS set<br>• APIF: Address or Stop Interrupt Flag in SSTATUS set |
| 0x02 | Master | TWI Master interrupt | • RIF: Read Interrupt Flag in MSTATUS set<br>• WIF: Write Interrupt Flag in MSTATUS set |

When an interrupt condition occurs, the corresponding Interrupt Flag is set in the Master register (TWI.MSTATUS) or Slave Status register (TWI.SSTATUS).
When several interrupt request conditions are supported by an interrupt vector, the interrupt requests are ORed together into one combined interrupt request to the Interrupt Controller. The user must read the peripheral's INTFLAGS register to determine which of the interrupt conditions are present.

**Related Links**

CPUINT - CPU Interrupt Controller

SREG

### 26.3.7 Sleep Mode Operation

The bus state logic and Slave continue to operate in all sleep modes, including Power Down sleep mode. If a Slave device is in sleep mode and a START condition is detected, clock stretching is active during the wake-up period until the system clock is available. Master will stop operation in all sleep modes.

### 26.3.8 Synchronization

Not applicable.

### 26.3.9 Configuration Change Protection

Not applicable.

## 26.4 Register Summary - TWI

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | | | | SDASETUP | SDAHOLD[1:0] | | FMPEN | |
| 0x01 | Reserved | | | | | | | | | |
| 0x02 | DBGCTRL | 7:0 | | | | | | | | DBGRUN |
| 0x03 | MCTRLA | 7:0 | RIEN | WIEN | | QCEN | TIMEOUT[1:0] | | SMEN | ENABLE |
| 0x04 | MCTRLB | 7:0 | | | | | FLUSH | ACKACT | CMD[1:0] | |
| 0x05 | MSTATUS | 7:0 | RIF | WIF | CLKHOLD | RXACK | ARBLOST | BUSERR | BUSSTATE[1:0] | |
| 0x06 | MBAUD | 7:0 | BAUD[7:0] | | | | | | | |
| 0x07 | MADDR | 7:0 | ADDR[7:0] | | | | | | | |
| 0x08 | MDATA | 7:0 | DATA[7:0] | | | | | | | |
| 0x09 | SCTRLA | 7:0 | DIEN | APIEN | PIEN | | | PMEN | SMEN | ENABLE |
| 0x0A | SCTRLB | 7:0 | | | | | | ACKACT | CMD[1:0] | |
| 0x0B | SSTATUS | 7:0 | DIF | APIF | CLKHOLD | RXACK | COLL | BUSERR | DIR | AP |
| 0x0C | SADDR | 7:0 | ADDR[7:0] | | | | | | | |
| 0x0D | SDATA | 7:0 | DATA[7:0] | | | | | | | |
| 0x0E | SADDRMASK | 7:0 | ADDRMASK[6:0] | | | | | | | ADDREN |

## 26.5 Register Description

### 26.5.1 Control A

**Name:** CTRLA
**Offset:** 0x00
**Reset:** 0x00
**Property:** -

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

**Bit 4 – SDASETUP: SDA Setup Time**
By default, there are 4 clock cycles of setup time on SDA out signal while reading from slave part of the TWI module. Writing this bit to '1' will change the setup time to 8 clocks.

| Value | Name | Description |
|---|---|---|
| 0 | 4CYC | SDA setup time is 4 clock cycles |
| 1 | 8CYC | SDA setup time is 8 clock cycle |

**Bits 3:2 – SDAHOLD[1:0]: SDA Hold Time**
Writing these bits selects the SDA hold time.

**Table 26-3. SDA Hold Time**

| SDAHOLD[1:0] | Nominal Hold Time | Hold Time Range across All Corners (ns) | Description |
|---|---|---|---|
| 0x0 | OFF | 0 | Hold time off. |
| 0x1 | 50ns | 36 - 131 | Backward compatible setting. |
| 0x2 | 300ns | 180 - 630 | Meets SMBus specification under typical conditions. |
| 0x3 | 500ns | 300 - 1050 | Meets SMBus specification across all corners. |

**Bit 1 – FMPEN: FM Plus Enable**
Writing these bits selects the 1MHz bus speed (Fast mode plus, Fm+) for the TWI in default configuration.

| Value | Description |
|---|---|
| 0 | Fm+ disabled |
| 1 | Fm+ enabled |

### 26.5.2 Debug Control

**Name:** DBGCTRL
**Offset:** 0x02
**Reset:** 0x00
**Property:** -

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

**Bit 0 – DBGRUN: Debug Run**

| Value | Description |
|---|---|
| 0 | The peripheral is halted in break debug mode and ignores events. |
| 1 | The peripheral will continue to run in break debug mode when the CPU is halted. |

### 26.5.3 Master Control A

**Name:** MCTRLA
**Offset:** 0x03
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RIEN | WIEN | | QCEN | TIMEOUT[1:0] | | SMEN | ENABLE |
| 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 – RIEN: Read Interrupt Enable**
Writing this bit to '1' enables interrupt on the Master Read Interrupt Flag (RIF) in the Master Status register (TWI.MSTATUS). A TWI Master read interrupt would be generated only if this bit, the RIF, and the Global Interrupt Flag (I) in CPU.SREG are all '1'.

**Bit 6 – WIEN: Write Interrupt Enable**
Writing this bit to '1' enables interrupt on the Master Write Interrupt Flag (WIF) in the Master Status register (TWI.MSTATUS). A TWI Master write interrupt will be generated only if this bit, the WIF, and the Global Interrupt Flag (I) in CPU.SREG are all '1'.

**Bit 4 – QCEN: Quick Command Enable**
Writing this bit to '1' enables Quick Command. When Quick Command is enabled, the corresponding interrupt flag is set immediately after the slave acknowledges the address. At this point the software can either issue a Stop command or a repeated Start by writing either the Command bits (CMD) in the Master Control B register (TWI.MCTRLB) or the Master Address register (TWI.MADDR).

**Bits 3:2 – TIMEOUT[1:0]: Inactive Bus Timeout**

| Value | Name | Description |
|---|---|---|
| 0x0 | DISABLED | Bus timeout disabled. $I^2C$. |
| 0x1 | 50US | 50µs - SMBus (assume baud is set to 100kHz) |
| 0x2 | 100US | 100µs (assume baud is set to 100kHz) |
| 0x3 | 200US | 200µs (assume baud is set to 100kHz) |

**Bit 1 – SMEN: Smart Mode Enable**
Writing this bit to '1' enables the Master smart mode. When smart mode is enabled, the acknowledge action is sent immediately after reading the Master Data (TWI.MDATA) register.

**Bit 0 – ENABLE: Enable TWI Master**
Writing this bit to '1' enables the TWI as Master.

### 26.5.4 Master Control B

**Name:** MCTRLB
**Offset:** 0x04
**Reset:** 0x00
**Property:** -

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

**Bit 3 – FLUSH: Flush**
Writing a '1' to this bit generates a strobe for one clock cycle disabling and then enabling the master.

Writing '0' has no effect.

The purpose is to clear the internal state of master: For TWI master to transmit successfully, it is recommended to write the Master Address register (TWI.MADDR) first and then the Master Data register (TWI.MDATA).

The peripheral will transmit invalid data if TWI.MDATA is written before TWI.MADDR. To avoid this invalid transmission, write '1' to this bit to clear both registers.

**Bit 2 – ACKACT: Acknowledge Action**
This bit defines the master's behavior under certain conditions defined by the bus protocol state and software interaction. The acknowledge action is performed when DATA is read, or when an execute command is written to the CMD bits.

The ACKACT bit is not a flag or strobe, but an ordinary read/write accessible register bit. The default ACKACT for master read interrupt is "Send ACK" (0). For master write, the code will know that no acknowledge should be sent since it is itself sending data.

| Value | Description |
|---|---|
| 0 | Send ACK |
| 1 | Send NACK |

**Bits 1:0 – CMD[1:0]: Command**
The master command bits are strobes. These bits are always read as zero.

Writing to these bits triggers a master operation as defined by the table below.

**Table 26-4. Command Settings**

| CMD[1:0] | DIR | Description |
|---|---|---|
| 0x0 | X | NOACT |
| 0x1 | X | REPSTART - Execute Acknowledge Action succeeded by repeated Start. |
| 0x2 | 0 | RECVTRANS - Execute Acknowledge Action succeeded by a byte read operation. |
| | 1 | Execute Acknowledge Action (no action) succeeded by a byte send operation.(1) |
| 0x3 | X | STOP - Execute Acknowledge Action succeeded by issuing a STOP condition. |

**Note:**

1. For a master being a sender, it will normally wait for new data written to the Master Data Register (TWI.MDATA).

The acknowledge action bits and command bits can be written at the same time.

### 26.5.5 Master Status

Normal TWI operation dictates that this register is regarded purely as a read-only register. Clearing any of the status flags is done indirectly by accessing the Master transmits address (TWI.MADDR), Master Data register (TWI.MDATA), or the Command bits (CMD) in the Master Control B register (TWI.MCTRLB).

**Name:** MSTATUS
**Offset:** 0x05
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RIF | WIF | CLKHOLD | RXACK | ARBLOST | BUSERR | BUSSTATE[1:0] | |
| Access | R/W | R/W | R/W | R | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bit 7 – RIF: Read Interrupt Flag**
This bit is set to '1' when the master byte read operation is successfully completed, i.e. no arbitration lost or bus error occurred during the operation. The read operation is triggered by software reading DATA or writing to ADDR registers with bit ADDR[0] written to '1'. A slave device must have responded with an ACK to the address and direction byte transmitted by the master for this flag to be set.

Writing a '1' to this bit will clear the RIF. However, normal use of the TWI does not require the flag to be cleared by this method.

Clearing the RIF bit will follow the same software interaction as the CLKHOLD flag.

The RIF flag can generate a master read interrupt (see description of the RIEN control bit in the TWI.MCTRLA register).

**Bit 6 – WIF: Write Interrupt Flag**
This bit is set when a master transmit address or byte write is completed, regardless of the occurrence of a bus error or an arbitration lost condition.

Writing a '1' to this bit will clear the WIF. However, normal use of the TWI does not require the flag to be cleared by this method.

Clearing the WIF bit will follow the same software interaction as the CLKHOLD flag.

The WIF flag can generate a master write interrupt (see description of the WIEN control bit in the TWI.MCTRLA register).

**Bit 5 – CLKHOLD: Clock Hold**
If read as '1', this bit indicates that the master is currently holding the TWI clock (SCL) low, stretching the TWI clock period.

Writing a '1' to this bit will clear the CLKHOLD flag. However, normal use of the TWI does not require the CLKHOLD flag to be cleared by this method, since the flag is automatically cleared when accessing several other TWI registers. The CLKHOLD flag can be cleared by:

1. Writing a '1' to it.
2. Writing to the TWI.MADDR register.
3. Writing to the TWI.MDATA register.
4. Reading the TWI.DATA register while the ACKACT control bits in TWI.MCTRLB are set to either send ACK or NACK.
5. Writing a valid command to the TWI.MCTRLB register.

**Bit 4 – RXACK: Received Acknowledge**
This bit is read-only and contains to the most recently received acknowledge bit from slave.

**Bit 3 – ARBLOST: Arbitration Lost**
If read as '1' this bit indicates that the master has lost arbitration while transmitting a high data or NACK bit, or while issuing a start or repeated start condition (S/Sr) on the bus.

Writing a '1' to it will clear the ARBLOST flag. However, normal use of the TWI does not require the flag to be cleared by this method. However, as for the CLKHOLD flag, clearing the ARBLOST flag is not required during normal use of the TWI.

Clearing the ARBLOST bit will follow the same software interaction as the CLKHOLD flag.

Given the condition where the bus ownership is lost to another master, the software must either abort operation or resend the data packet. Either way, the next required software interaction is in both cases to write to the TWI.MADDR register. A write access to the TWI.MADDR register will then clear the ARBLOST flag.

**Bit 2 – BUSERR: Bus Error**
The BUSERR flag indicates that an illegal bus condition has occurred. An illegal bus condition is detected if a protocol violating start (S), repeated start (Sr), or stop (P) is detected on the TWI bus lines. A start condition directly followed by a stop condition is one example of protocol violation.

Writing a '1' to this bit will clear the BUSERR. However, normal use of the TWI does not require the BUSERR to be cleared by this method.

A robust TWI driver software design will treat the bus error flag similarly to the ARBLOST flag, assuming the bus ownership is lost when the bus error flag is set. As for the ARBLOST flag, the next software operation of writing the TWI.MADDR register will consequently clear the BUSERR flag. For bus error to be detected, the bus state logic must be enabled and the system frequency must be 4x the SCL frequency.

**Bits 1:0 – BUSSTATE[1:0]: Bus State**
These bits indicate the current TWI bus state as defined in the table below. After a System Reset or re-enabling, the TWI master bus state will be unknown. The change of bus state is dependent on bus activity.

Writing 0x1 to the BUSSTATE bits forces the bus state logic into its 'idle' state. However, the bus state logic cannot be forced into any other state. When the master is disabled, the bus-state is 'unknown'.

| Value | Name | Description |
|---|---|---|
| 0x0 | UNKNOWN | Unknown bus state |
| 0x1 | IDLE | Bus is idle |
| 0x2 | OWNER | This TWI controls the bus |
| 0x3 | BUSY | The bus is busy |

### 26.5.6 Master Baud Rate

**Name:** MBAUD
**Offset:** 0x06
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | BAUD[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 – BAUD[7:0]: Baud Rate**
This bit field is used to derive the SCL high and low time and should be written while the master is disabled (ENABLE bit in TWI.MCTRLA is '0').

For more information on how to calculate the frequency, see Clock Generation.

### 26.5.7 Master Address

**Name:** MADDR
**Offset:** 0x07
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ADDR[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 – ADDR[7:0]: Address**
When this bit field is written, a START condition and slave address protocol sequence is initiated dependent on the bus state.

If the bus state is unknown the Master Write Interrupt Flag (WIF) and bus error flag (BUSERR) in the Master Status register (TWI.MSTATUS) are set and the operation is terminated.

If the bus is busy the master awaits further operation until the bus becomes idle. When the bus is or becomes idle, the master generates a START condition on the bus, copies the ADDR value into the data shift register (TWI.MDATA) and performs a byte transmit operation by sending the contents of the data register onto the bus. The master then receives the response i.e. the acknowledge bit from the slave. After completing the operation the bus clock (SCL) is forced and held low only if arbitration was not lost. The CLKHOLD bit in the Master Setup register (TWI.MSETUP) is set accordingly. Completing the operation sets the WIF in the Master Status register (TWI.MSTATUS).

If the bus is already owned, a repeated start (Sr) sequence is performed. In two ways the repeated start (Sr) sequence deviates from the start sequence. Firstly, since the bus is already owned by the master, no wait for idle bus state is necessary. Secondly, if the previous transaction was a read, the acknowledge action is sent before the repeated start bus condition is issued on the bus.

The master receives one data byte from the slave before the master sets the Master Read Interrupt Flag (RIF) in the Master Status register (TWI.MSTATUS). All TWI Master flags are cleared automatically when this bit field is written. This includes bus error, arbitration lost, and both master interrupt flags.

This register can be read at any time without interfering with ongoing bus activity, since a read access does not trigger the master logic to perform any bus protocol related operations.

The master control logic uses bit 0 of the TWI.MADDR register as the bus protocol's read/write flag (R/$\overline{W}$).

### 26.5.8 Master DATA

**Name:** MDATA
**Offset:** 0x08
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DATA[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 – DATA[7:0]: Data**
The bit field gives direct access to the masters physical shift register which is used both to shift data out onto the bus (write) and to shift in data received from the bus (read).

The direct access implies that the data register cannot be accessed during byte transmissions. Build-in logic prevents any write access to this register during the shift operations. Reading valid data or writing data to be transmitted can only be successfully done when the bus clock (SCL) is held low by the master, i.e. when the CLKHOLD bit in the Master Status register (TWI.MSTATUS) is set. However, it is not necessary to check the CLKHOLD bit in software before accessing this register if the software keeps track of the present protocol state by using interrupts or observing the interrupt flags.

Accessing this register assumes that the master clock hold is active, auto-triggers bus operations dependent of the state of the acknowledge action command bit (ACKACT) in TWI.MSTATUS and type of register access (read or write).

A write access to this register will, independent of ACKACT in TWI.MSTATUS, command the master to perform a byte transmit operation on the bus directly followed by receiving the acknowledge bit from the slave. When the acknowledge bit is received, the Master Write Interrupt Flag (WIF) in TWI.MSTATUS is set regardless of any bus errors or arbitration. If operating in a multi-master environment, the interrupt handler or application software must check the Arbitration Lost Status Flag (ARBLOST) in TWI.MSTATUS before continuing from this point. If the arbitration was lost, the application software must decide to either abort or to resend the packet by rewriting this register. The entire operation is performed (i.e. all bits are clocked), regardless of winning or losing arbitration before the write interrupt flag is set. When arbitration is lost, only '1's are transmitted for the remainder of the operation, followed by a write interrupt with ARBLOST flag set.

Both TWI master interrupt flags are cleared automatically when this register is written. However, the Master Arbitration Lost and Bus Error flags are left unchanged.

Reading this register triggers a bus operation, dependent on the setting of the acknowledge action command bit (ACKACT) in TWI.MSTATUS. Normally the ACKACT bit is preset to either ACK or NACK before the register read operation. If ACK or NACK action is selected, the transmission of the acknowledge bit precedes the release of the clock hold. The clock is released for one byte, allowing the slave to put one byte of data on the bus. The Master Read Interrupt flag RIF in TWI.MSTATUS is then set if the procedure was successfully executed. However, if arbitration was lost when sending NACK, or a bus error occurred during the time of operation, the Master Write Interrupt flag (WIF) is set instead. Observe that the two master interrupt flags are mutual exclusive, i.e. both flags will not be set simultaneously.

© 2017 Microchip Technology Inc.

Both TWI master interrupt flags are cleared automatically if this register is read while ACKACT is set to either ACK or NACK. However, arbitration lost and bus error flags are left unchanged.

### 26.5.9 Slave Control A

**Name:** SCTRLA
**Offset:** 0x09
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DIEN | APIEN | PIEN | | | PMEN | SMEN | ENABLE |
| Access | R/W | R/W | R/W | | | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | | | 0 | 0 | 0 |

**Bit 7 – DIEN: Data Interrupt Enable**
Writing this bit to '1' enables interrupt on the Slave Data Interrupt Flag (DIF) in the Slave Status register (TWI.SSTATUS). A TWI slave data interrupt will be generated only if this bit, the DIF, and the Global Interrupt Flag (I) in CPU.SREG are all '1'.

**Bit 6 – APIEN: Address or Stop Interrupt Enable**
Writing this bit to '1' enables interrupt on the Slave Address or Stop Interrupt Flag (APIF) in the Slave Status register (TWI.SSTATUS). A TWI slave address or stop interrupt will be generated only if the this bit, APIF, PIEN in this register, and the Global Interrupt Flag (I) in CPU.SREG are all '1'.

The slave stop interrupt shares the interrupt vector with slave address interrupt. The AP bit determines which caused the interrupt.

**Bit 5 – PIEN: Stop Interrupt Enable**
Writing this bit to '1' enables APIF to be set when a STOP condition occurs. To use this feature the system frequency must be 4x the SCL frequency.

**Bit 2 – PMEN: Address Recognition Mode**
If this bit is written to '1', the slave address match logic responds to all received addresses.

If this bit is written to '0', the address match logic uses the slave address register (TWI.SADDR) to determine which address to recognize as the slaves own address.

**Bit 1 – SMEN: Smart Mode Enable**
Writing this bit to '1' enables the slave smart mode. When smart mode is enabled, issuing a command with CMD or reading/writing DATA resets the interrupt and operation continues. If smart mode is disabled, the slave always waits for a CMD command before continuing.

**Bit 0 – ENABLE: Enable TWI Slave**
Writing this bit to '1' enables the TWI slave.

### 26.5.10 Slave Control B

**Name:** SCTRLB
**Offset:** 0x0A
**Reset:** 0x00
**Property:** -

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

**Bit 2 – ACKACT: Acknowledge Action**
This bit defines the slave's behavior under certain conditions defined by the bus protocol state and software interaction. The table below lists the acknowledge procedure performed by the slave if action is initiated by software. The acknowledge action is performed when TWI.SDATA is read or written, or when an execute command is written to the CMD bits in this register.

The ACKACT bit is not a flag or strobe, but an ordinary read/write accessible register bit.

| Value | Name | Description |
|---|---|---|
| 0 | ACK | Send ACK |
| 1 | NACK | Send NACK |

**Bits 1:0 – CMD[1:0]: Command**
Unlike the acknowledge action bits, the slave command bits are strobes. These bits always read as zero. Writing to these bits trigger a slave operation as defined in the table below.

**Table 26-5. Command Settings**

| CMD[1:0] | DIR | Description |
|---|---|---|
| 0x0 - NOACT | X | No action |
| 0x1 | X | Reserved |
| 0x2 - COMPTRANS | Used to complete a transaction. | |
| | 0 | Execute Acknowledge Action succeeded by waiting for any START (S/Sr) condition. |
| | 1 | Wait for any START (S/Sr) condition. |
| 0x3 - RESPONSE | Used in response to an address interrupt (APIF). | |
| | 0 | Execute Acknowledge Action succeeded by reception of next byte. |
| | 1 | Execute Acknowledge Action succeeded by slave data interrupt. |
| | Used in response to a data interrupt (DIF). | |
| | 0 | Execute Acknowledge Action succeeded by reception of next byte. |
| | 1 | Execute a byte read operation followed by Acknowledge Action. |

The acknowledge action bits and command bits can be written at the same time.

### 26.5.11 Slave Status

Normal TWI operation dictates that the slave status register should be regarded purely as a read-only register. Clearing any of the status flags will indirectly be done when accessing the slave data (TWI.SDATA) register or the CMD bits in the Slave Control B register (TWI.SCTRLB).

**Name:** SSTATUS
**Offset:** 0x0B
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DIF | APIF | CLKHOLD | RXACK | COLL | BUSERR | DIR | AP |
| Access | R/W | R/W | R | R | R/W | R/W | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bit 7 – DIF: Data Interrupt Flag**
This flag is set when a slave byte transmit or byte receive operation is successfully completed without any bus error. The flag can be set with an unsuccessful transaction in case of collision detection (see description of the COLL status bit). Writing a '1' to its bit location will clear the DIF. However, normal use of the TWI does not require the DIF flag to be cleared by using this method, since the flag is automatically cleared when:

1. Writing to the slave DATA register.
2. Reading the slave DATA register.
3. Writing a valid command to the CTRLB register.

The DIF flag can be used to generate a slave data interrupt (see description of the DIEN control bit in TWI.CTRLA).

**Bit 6 – APIF: Address or Stop Interrupt Flag**
This flag is set when the slave address match logic detects that a valid address has been received or by a stop condition. Writing a '1' to its bit location will clear the APIF. However, normal use of the TWI does not require the flag to be cleared by this method since the flag is cleared using same software interactions as described for the DIF flag.

The APIF flag can be used to generate a slave address or stop interrupt (see description of the AIEN control bit in TWI.CTRLA). Take special note of that the slave stop interrupt shares the interrupt vector with slave address interrupt.

**Bit 5 – CLKHOLD: Clock Hold**
If read as '1', the slave clock hold flag indicates that the slave is currently holding the TWI clock (SCL) low, stretching the TWI clock period. This is a read only bit that is set when an address or data interrupt is set. Resetting the corresponding interrupt will indirectly reset this flag.

**Bit 4 – RXACK: Received Acknowledge**
This bit is read only and contains to the most recently received acknowledge bit from the master.

**Bit 3 – COLL: Collision**
If read as '1', the transmit collision flag indicates that the slave has not been able to transmit a high data or NACK bit. If a slave transmit collision is detected, the slave will commence its operation as normal, except no low values will be shifted out onto the SDA line (i.e., when the COLL flag is set to '1' it disables the data and acknowledge output from the slave logic). DIF flag will be set to '1' at the end as a result of

© 2017 Microchip Technology Inc.

internal completion of unsuccessful transaction. Similarly when collision occurs because slave is not been able to transmit NACK bit, it means address match already happened and APIF flag is set as a result. APIF/DIF flags can only generate interrupt whose handlers can be used to check for the collision. Writing a '1' to its bit location will clear the COLL flag. However, the flag is automatically cleared if any START condition (S/Sr) is detected.

This flag is intended for systems where address resolution protocol (ARP) is employed. However, a detected collision in non-ARP situations indicates that there has been a protocol violation and should be treated as a bus error.

**Bit 2 – BUSERR: Bus Error**
The BUSERR flag indicates that an illegal bus condition has occurred. An illegal bus condition is detected if a protocol violating start (S), repeated start (Sr), or stop (P) is detected on the TWI bus lines. A start condition directly followed by a stop condition is one example of protocol violation. Writing a '1' to its bit location will clear the BUSERR flag. However, normal use of the TWI does not require the BUSERR to be cleared by this method. A robust TWI driver software design will assume that the entire packet of data has been corrupted and restart by waiting for a new START condition (S). The TWI bus error detector is part of the TWI Master circuitry. For bus errors to be detected, the TWI Master must be enabled (ENABLE bit in TWI.MCTRLA is '1'), and the system clock frequency must be at least four times the SCL frequency.

**Bit 1 – DIR: Read/Write Direction**
This bit is read only and indicates the current bus direction state. The DIR bit reflects the direction bit value from the last address packet received from a master TWI device. If this bit is read as '1', a master read operation is in progress. Consequently a '0' indicates that a master write operation is in progress.

**Bit 0 – AP: Address or Stop**
When the TWI slave address or stop interrupt flag (APIF) is set, this bit determines whether the interrupt is due to address detection or a stop condition.

| Value | Name | Description |
|---|---|---|
| 0 | STOP | A stop condition generated the interrupt on APIF. |
| 1 | ADR | Address detection generated the interrupt on APIF. |

### 26.5.12 Slave Address

**Name:** SADDR
**Offset:** 0x0C
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ADDR[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 – ADDR[7:0]: Address**
The slave address register in combination with the slave address mask register (TWI.SADDRMASK) is used by the slave address match logic to determine if a master TWI device has addressed the TWI slave. The slave address interrupt flag (APIF) is set to one if the received address is recognized. The slave address match logic supports recognition of 7- and 10-bits addresses, and general call address.

When using 7-bit or 10-bit address recognition mode, the upper 7-bits of the address register (ADDR[7:1]) represents the slave address and the least significant bit (ADDR[0]) is used for general call address recognition. Setting the ADDR[0] bit in this case enables the general call address recognition logic. The TWI slave address match logic only support recognition of the first byte of a 10-bit address i.e. by setting ADDRA[7:1] = “0b11110aa” where “aa” represents bit 9 and 8 or the slave address. The second 10-bit address byte must be handled by software.

### 26.5.13 Slave Data

**Name:** SDATA
**Offset:** 0x0D
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DATA[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 – DATA[7:0]: Data**
The slave data register I/O location (DATA) provides direct access to the slave's physical shift register, which is used both to shift data out onto the bus (transmit) and to shift in data received from the bus (receive). The direct access implies that the data register cannot be accessed during byte transmissions. Built-in logic prevents any write accesses to the data register during the shift operations. Reading valid data or writing data to be transmitted can only be successfully done when the bus clock (SCL) is held low by the slave, i.e. when the slave CLKHOLD bit is set. However, it is not necessary to check the CLKHOLD bit in software before accessing the slave DATA register if the software keeps track of the present protocol state by using interrupts or observing the interrupt flags. Accessing the slave DATA register, assumed that clock hold is active, auto-trigger bus operations dependent of the state of the slave acknowledge action command bits (ACKACT) and type of register access (read or write).

### 26.5.14 Slave Address Mask

**Name:** SADDRMASK
**Offset:** 0x0E
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ADDRMASK[6:0] | | | | | | | ADDREN |
| 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 7:1 – ADDRMASK[6:0]: Address Mask**
The ADDRMASK register acts as a second address match register, or an address mask register depending on the ADDREN setting.

If ADDREN is written to '0', ADDRMASK can be loaded with a 7-bit Slave Address mask. Each of the bits in ADDRMASK register can mask (disable) the corresponding address bits in the TWI slave Address

Register (ADDR). If the mask bit is written to '1' then the address match logic ignores the compare between the incoming address bit and the corresponding bit in slave ADDR register. In other words, masked bits will always match.

If ADDREN is written to '1', the slave ADDRMASK can be loaded with a second slave address in addition to the ADDR register. In this mode, the slave will match on 2 unique addresses, one in ADDR and the other in ADDRMASK.

**Bit 0 – ADDREN: Address Mask Enable**
If this bit is written to '1', the slave address match logic responds to the 2 unique addresses in slave ADDR and ADDRMASK.

If this bit is '0', the ADDRMASK register acts as a mask to the slave ADDR register.

# 27. CRCSCAN - Cyclic Redundancy Check Memory Scan

## 27.1 Features

- CRC-16-CCITT
- Can check full Flash, application code and/or boot section
- Single/continuous background or priority check selectable
- Selectable NMI trigger on failure
- User configurable check during internal reset initialization
- Paused in all CPU sleep modes

## 27.2 Overview

A Cyclic Redundancy Check (CRC) takes a data stream of bytes from the NVM (either entire Flash, only Boot section, or both application code and Boot section) and generates a checksum. The CRC peripheral (CRCSCAN) can be used to detect errors in program memory:

The last location in the section to check has to contain the correct pre-calculated 16-bit checksum for comparison. If the checksum calculated by the CRCSCAN and the pre-calculated checksums match, a status bit in the CRCSCAN is set. If they do not match, the status register will indicate that it failed. The user can choose to let the CRCSCAN generate a non-maskable interrupt (NMI) if the checksums do not match.

An *n*-bit CRC, applied to a data block of arbitrary length, will detect any single alteration (error burst) up to *n* bits in length. For longer error bursts, a fraction $1-2^{-n}$ will be detected.

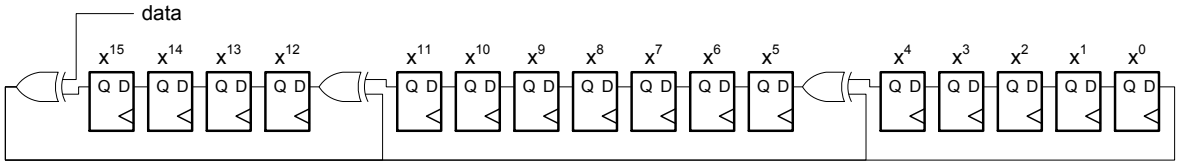
The CRC-generator supports CRC-16-CCITT.

Polynomial:

- CRC-16-CCITT: $x^{16} + x^{12} + x^5 + 1$

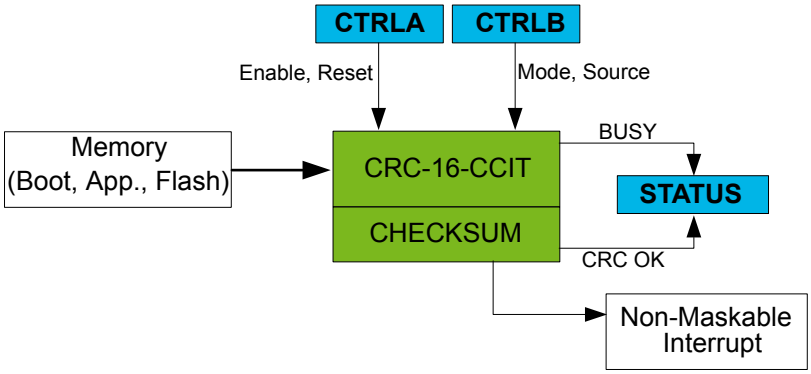
The CRC reads in byte by byte of the content of the section(s) it is set up to check, starting with byte 0, and generates a new checksum per byte. The byte is sent through an implementation corresponding to Figure 27-1, starting with the most significant bit. If the last two bytes in the section contain the correct 16-bit checksum, the CRC will pass. See Checksum for how to place the checksum. The initial value of the checksum register is 0xFFFF.

**Figure 27-1. CRC Implementation Description**



### 27.2.1 Block Diagram

**Figure 27-2. Cyclic Redundancy Check Block Diagram**




### 27.2.2 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 27-1. System Product Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | No | - |
| Interrupts | Yes | CPUINT |
| Events | No | - |
| Debug | Yes | UPDI |

**Related Links**

Clocks

Interrupts

#### 27.2.2.1 Clocks

This peripheral depends on the peripheral clock.

**Related Links**

CLKCTRL - Clock Controller

#### 27.2.2.2 I/O Lines and Connections

Not applicable.

#### 27.2.2.3 Interrupts

Using the interrupts of this peripheral requires the Interrupt Controller to be configured first.

**Related Links**

CPUINT - CPU Interrupt Controller

SREG

Interrupts

#### 27.2.2.4 Events

Not applicable.

#### 27.2.2.5 Debug Operation

Whenever the debugger accesses the device, for instance reading or writing a peripheral or memory location, the CRCSCAN peripheral will be disabled.

If the CRCSCAN is busy when the debugger accesses the device, the CRCSCAN will restart the ongoing operation when the debugger accesses an internal register or when the debugger disconnects.

The BUSY bit in the Status register (CRCSCAN.STATUS) will read '1' if the CRCSCAN was busy when the debugger caused it to disable, but it will not actively check any section as long as the debugger keeps it disabled. There are synchronized CRC status bits in the debugger's internal register space, which can be read by the debugger without disabling the CRCSCAN. Reading the debugger's internal CRC status bits will make sure that the CRCSCAN is enabled.

It is possible to write the CRCSCAN.STATUS register directly from the debugger:

- BUSY bit in CRCSCAN.STATUS:
  - Writing the BUSY bit to '0' will stop the ongoing CRC operation (so that the CRCSCAN does not restart its operation when the debugger allows it).
  - Writing the BUSY bit to '1' will make the CRC start a single check with the settings in the Control B register (CRCSCAN.CTRLB), but not until the debugger allows it.

  As long as the BUSY bit in CRCSCAN.STATUS is '1', CRCSCAN.CRCTRLB and the Non-Maskable Interrupt Enable bit (NMIEN) in the Control A register (CRCSCAN.CTRLA) cannot be altered.
- OK bit in CRCSCAN.STATUS:
  - Writing the OK bit to '0' can trigger a non-maskable interrupt (NMI) if the NMIEN bit in CRCSCAN.CTRLA is '1'. If an NMI has been triggered, no writes to the CRCSCAN are allowed.
  - Writing the OK bit to '1' will make the OK bit read as '1' when the BUSY bit in CRCSCAN.STATUS is '0'.

Writes to CRCSCAN.CTRLA and CRCSCAN.CTRLB from the debugger are treated in the same way as writes from the CPU.

**Related Links**

UPDI - Unified Program and Debug Interface

CTRLA

CTRLB

## 27.3 Functional Description

### 27.3.1 Initialization

To enable a CRC in software (or via the debugger):

1. Write the MODE and Source (SRC) bit fields of the Control B register (CRCSCAN.CTRLB) to select the desired mode and source settings.
2. Enable the CRCSCAN by writing a '1' to the ENABLE bit in the Control A register (CRCSCAN.CTRLA).
3. The CRC will start after three cycles, and the CPU will continue executing during these three cycles.

The CRCSCAN can be enabled during the internal reset initialization to ensure the Flash is OK before letting the CPU execute code. If the CRCSCAN fails during the internal reset initialization, the CPU is not

allowed to start normal code execution. The full source settings are available during the internal reset initialization. See the Fuse description for more information.

If the CRCSCAN was enabled during the internal reset initialization, the CRC Control A and B registers will reflect this when normal code execution is started:

- The ENABLE bit in CRCSCAN.CTRLA will be '1'
- The MODE bit field in CRCSCAN.CTRLB will be BACKGROUND
- The SRC bit field in CRCSCAN.CTRLB will reflect the checked section(s).

**Related Links**

CTRLA

CTRLB

FUSES - Configuration and User Fuses

Reset time

### 27.3.2 Operation

The CRC can run in these modes:

- Continuous mode in the background: the CRC restarts checking from the beginning of the selected section. The CPU can continue executing code and the CRC fetches data when it can.
- Single mode in the background: A single CRC is executed. The CPU can continue executing code and the CRC fetches data when it can.
- Priority mode: the CRC peripheral has priority access to the Flash and will stall the CPU until completed.

Although the CPU has priority under the continuous mode and single mode in the background, there can still be a "collision". Therefore, the CPU will have to wait one additional cycle when this condition occurs. It is important that it should not run continuous background checks during executing or running safety critical code. Otherwise, it is hard to get test coverage.

The CRC fetches a new word (16-bit) on every third main clock cycle when in priority mode, or when the CRC module is set to do a scan from startup. In background mode, scan time can not be guaranteed.

#### 27.3.2.1 Checksum

The pre-calculated checksum must be present in the last location of the section to be checked. If the BOOT section should be checked, the 16-bit checksum must be saved in the last two bytes of the BOOT section, and similarly for APPLICATION and entire Flash. Table 27-2 shows explicitly how the checksum should be stored for the different sections. Also see CTRLB for how to configure which section to check and the device fuse description for how to configure the BOOTEND and APPEND fuses.

**Table 27-2. How to place the pre-calculated 16-bit checksum in Flash**

| Section to Check | CHECKSUM[15:8] | CHECKSUM[7:0] |
|---|---|---|
| BOOT | FUSE_BOOTEND*256-2 | FUSE_BOOTEND*256-1 |
| BOOT and APPLICATION | FUSE_APPEND*256-2 | FUSE_APPEND*256-1 |
| Full Flash | FLASHEND-1 | FLASHEND |

#### 27.3.2.2 Background Scan Timing

As long as the CPU is fetching continuously, the CRC will not get access because the CPU has priority to the Flash. In certain multi-cycle instructions, however, the CPU does not fetch in all cycles: This allows the CRC to get a single access during these instructions. These instructions include:

- All LD instructions (LD, LDS, LDD with X/Y/Z pointer)
- POP
- RET/RETI
- CALL/JMP (not RJMP/RCALL/IJMP/ICALL)
- All MUL instructions (MUL, MULS, MULSU, FMUL...)
- ADIW/SBIW

If the cache is enabled, the CRC will get access as long as the CPU is executing code from a cached loop, with the same timing assumptions as for priority mode.

### 27.3.3 Interrupts

**Table 27-3. Available Interrupt Vectors and Sources**

| Offset | Name | Vector Description | Conditions |
|---|---|---|---|
| 0x00 | NMI | Non-Maskable Interrupt | Generated on CRC failure |

When the interrupt condition occurs, the OK flag in the Status register (CRCSCAN.STATUS) is cleared to '0'.

An interrupt is enabled by writing a '1' to the respective Enable bit (NMIEN) in the Control A register (CRCSCAN.CTRLA), but can only be disabled with a system Reset. An NMI is generated when the OK flag in CRCSCAN.STATUS is cleared and the NMIEN bit is '1'. The NMI request remains active until a system Reset, and can not be disabled.

A non-maskable interrupt can be triggered even if interrupts are not globally enabled.

**Related Links**

CTRLA

STATUS

CPUINT - CPU Interrupt Controller

### 27.3.4 Sleep Mode Operation

In all CPU sleep modes, the CRCSCAN peripheral is halted, and will resume operation when the CPU wakes up.

It is possible to enter sleep mode after setting up the CRCSCAN to start a PRIORITY check (see CTRLB for more information), but before the actual check is started. If the CPU is able enter sleep mode before the check starts and a Priority check was configured, the check will start and complete (halting the CPU) immediately after waking up, and before entering any interrupt handler.

### 27.3.5 Configuration Change Protection

Not applicable.

## 27.4 Register Summary - CRCSCAN

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | RESET | | | | | | NMIEN | ENABLE |
| 0x01 | CTRLB | 7:0 | | | MODE[1:0] | | | | SRC[1:0] | |
| 0x02 | STATUS | 7:0 | | | | | | | OK | BUSY |

## 27.5 Register Description

### 27.5.1 Control A

If an NMI has been triggered, this register is not writable.

**Name:** CTRLA
**Offset:** 0x00
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RESET | | | | | | NMIEN | ENABLE |
| Access | R/W | | | | | | R/W | R/W |
| Reset | 0 | | | | | | 0 | 0 |

**Bit 7 – RESET: Reset CRCSCAN**
Writing this bit to '1' resets the CRCSCAN peripheral: The CRCSCAN Control registers and Status register (CTRLA, CTRLB, STATUS) will be cleared one clock cycle after the RESET bit was written to '1'.

If NMIEN is '0', this bit is writable both when the CRCSCAN is busy (BUSY bit in CRCSCAN.STATUS is '1') and not busy (BUSY bit is '0'), and will take effect immediately.

If NMIEN is '1', this bit is only writable when the CRCSCAN is not busy (BUSY bit in CRCSCAN.STATUS is '0').

The RESET bit is a strobe bit.

**Bit 1 – NMIEN: Enable NMI Trigger**
When this bit is written to '1', any CRC failure will trigger an NMI.

This can only be cleared by a system Reset - it is not cleared by a write to the RESET bit.

This bit can only be written to '1' when the CRCSCAN is not busy (BUSY bit in CRCSCAN.STATUS is '0').

**Bit 0 – ENABLE: Enable CRCSCAN**
Writing this bit to '1' enables the CRCSCAN peripheral with the current settings. It will stay '1' even after a CRC check has completed, but writing it to 1 again will start a new check.

Writing the bit to '0' will disable the CRCSCAN after the ongoing check is completed (after reaching the end of the section it is set up to check). This is the preferred way to stop a continuous background check. A failure in the ongoing check will still be detected and can cause an NMI if the NMIEN bit is '1'.

The CRCSCAN can be enabled during the internal reset initialization to verify Flash sections before letting the CPU start normal code execution (see device datasheet fuse description). If the CRCSCAN is

enabled during the internal reset initialization, the ENABLE bit will read as '1' when normal code execution starts.

To see whether the CRCSCAN peripheral is busy with an ongoing check, poll the Busy bit (BUSY) in the Status register (CRCSCAN.STATUS).

**Related Links**

FUSES - Configuration and User Fuses

Reset time

### 27.5.2 Control B

The CTRLB register contains the mode and source settings for the CRC. It is not writable when the CRC is busy or when an NMI has been triggered.

**Name:** CTRLB
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

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

**Bits 5:4 – MODE[1:0]: CRC Flash Access Mode**

The MODE bit field selects the priority of the CRC module in the system when accessing Flash, either in the background or completely stalling the CPU until the CRC is completed. It is also possible to select the CONTINUOUS mode, which will make the CRC module restart at the beginning of the selected section after finishing a section. The CONTINUOUS mode will stop if a failure occurs. To otherwise stop a CONTINUOUS mode, write the ENABLE or RESET bit in the CRCSCAN.CTRLA register.

The CRC can be enabled during internal reset initialization to verify Flash sections before letting the CPU start (see device datasheet fuse description). If the CRC is enabled during internal reset initialization, the MODE bit field will read out as BACKGROUND when normal code execution starts.

| Value | Name | Description |
|---|---|---|
| 0x0 | PRIORITY | The CRC module runs a single check with priority to Flash. The CPU is halted until the CRC completes. |
| 0x2 | BACKGROUND | The CRC module runs a single check with lowest priority to Flash. |
| 0x3 | CONTINUOUS | The CRC module runs continuous checks in the background. After completing a successful check it restarts. |
| other | - | Reserved |

**Bits 1:0 – SRC[1:0]: CRC Source**

The SRC bit field selects which section of the Flash the CRC module should check. To set up section sizes, refer to the fuse description.

The CRC can be enabled during internal reset initialization to verify Flash sections before letting the CPU start (see fuse description). If the CRC is enabled during internal reset initialization, the SRC bit field will read out as FLASH, BOOTAPP or BOOT when normal code execution starts (depending on the configuration).

© 2017 Microchip Technology Inc.

| Value | Name | Description |
|---|---|---|
| 0x0 | FLASH | The CRC is performed on the entire Flash (boot, application code and application data sections). |
| 0x1 | BOOTAPP | The CRC is performed on the boot and application code sections of Flash. |
| 0x2 | BOOT | The CRC is performed on the boot section of Flash. |
| 0x3 | - | Reserved. |

**Related Links**

FUSES - Configuration and User Fuses

Reset time

### 27.5.3 Status

The status register contains the busy and OK information. It is not writable, only readable.

**Name:** STATUS
**Offset:** 0x02
**Reset:** 0x02
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | OK | BUSY |
| Access | | | | | | | R | R |
| Reset | | | | | | | 1 | 0 |

**Bit 1 – OK: CRC OK**
When this bit is read as 1, the previous CRC completed successfully. The bit is set to '1' from Reset, but is cleared to '0' when enabling. As long as the CRC module is busy, it will be read '0'. When running continuously, the CRC status must be assumed OK until it fails or is stopped by the user.

**Bit 0 – BUSY: CRC Busy**
When this bit is read as 1, the CRC module is busy. As long as the module is busy, the access to the control registers are limited. See CTRLA and CTRLB for more information.

# 28. CCL – Configurable Custom Logic

## 28.1 Features

- Glue logic for general purpose PCB design
- Up to two Programmable LookUp Tables LUT[1:0]
- Combinatorial Logic Functions: Any logic expression which is a function of up to three inputs.
- Sequential Logic Functions:
  Gated D Flip-Flop, JK Flip-Flop, gated D Latch, RS Latch
- Flexible Lookup Table Inputs Selection:
  - I/Os
  - Events
  - Subsequent LUT Output
  - Internal Peripherals
    - Analog Comparator
    - Timer/Counters
    - USART
    - SPI
- Clocked by system clock or other peripherals
- Output can be connected to IO pins or Event System
- Optional synchronizer, filter, or edge detector available on each LUT output

## 28.2 Overview

The Configurable Custom Logic (CCL) is a programmable logic peripheral which can be connected to the device pins, to events, or to other internal peripherals. The CCL can serve as "glue logic" between the device peripherals and external devices. The CCL can eliminate the need for external logic components, and can also help the designer to overcome real-time constraints by combining core independent peripherals to handle the most time critical parts of the application independent of the CPU.

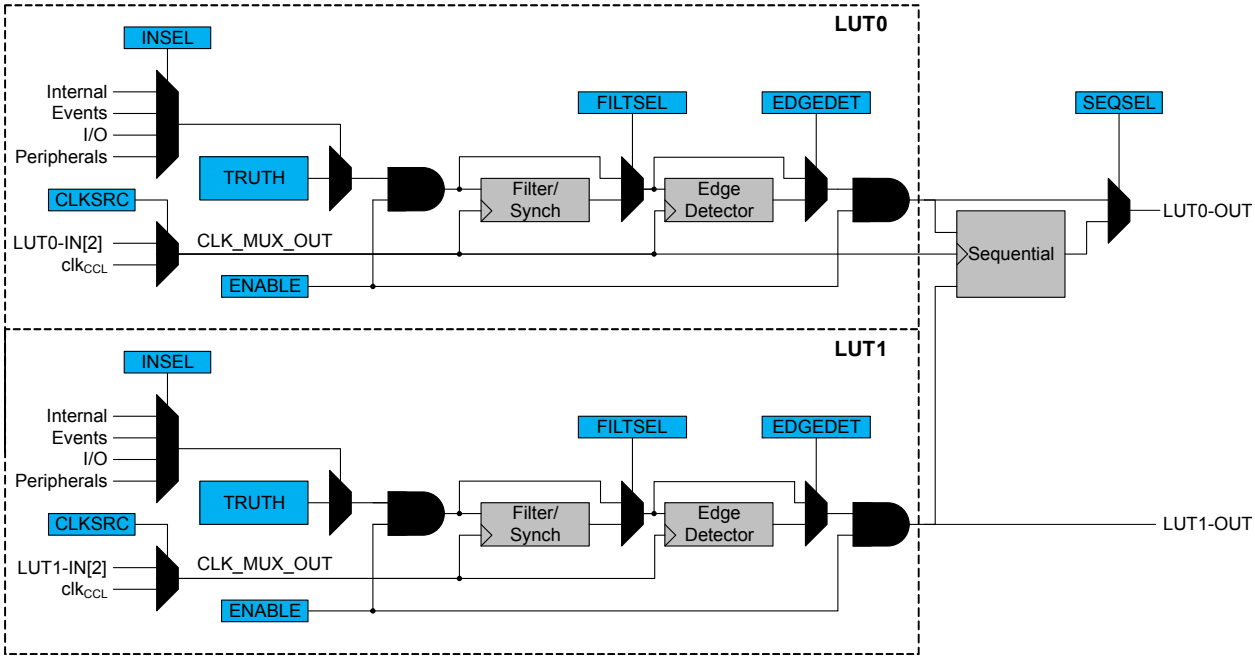
The CCL peripheral has one pair of Lookup Tables (LUT). Each LUT consists of three inputs, a truth table, and a filter/edge detector. Each LUT can generate an output as a user programmable logic expression with three inputs. Inputs can be individually masked.

The output can be generated from the inputs combinatorially, and can be filtered to remove spikes. An optional Sequential module can be enabled. The inputs to the Sequential module are individually controlled by two independent, adjacent LUT (LUT0/LUT1) outputs, enabling complex waveform generation.

© 2017 Microchip Technology Inc.

### 28.2.1 Block Diagram

**Figure 28-1. Configurable Custom Logic**




### 28.2.2 Signal Description

| Pin Name | Type | Description |
|---|---|---|
| LUTn-OUT | Digital output | Output from lookup table |
| LUTn-IN[2:0] | Digital input | Input to lookup table |

Refer to *I/O Multiplexing and Considerations* for details on the pin mapping for this peripheral. One signal can be mapped on several pins.

**Related Links**

I/O Multiplexing and Considerations

### 28.2.3 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 28-1. CCL System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | Yes | PORT |
| Interrupts | Yes | CPUINT |
| Events | Yes | EVSYS |
| Debug | Yes | UPDI |

#### 28.2.3.1 Clocks

By default, the CCL is using the peripheral clock of the device (CLK_PER).

Alternatively, the CCL can be clocked by a peripheral input that is available on LUT n input line 2 (LUTn_IN[2]). This is configured by writing a '1' to the Clock Source Selection bit (CLKSRC) in the LUTn Control A register (CCL.LUTnCTRLA). The sequential block is clocked by the same clock as that of its even LUT in the LUT pair (SEQn.clk = LUT2n.clk). It is advised to disable the peripheral by writing a '0' to the Enable bit (ENABLE) in the Control A register (CCL.CTRLA) before configuring the CLKSRC bit in CCL.LUTnCTRLA.

Alternatively, the input line 2 (IN[2]) of a LUT can be used to clock the LUT and the corresponding Sequential block. This is enabled by writing a '1' to the Clock Source Selection bit (CLKSRC) in the LUTn Control A register (CCL.LUTnCTRLA).

The CCL must be disabled before changing the LUT clock source: write a '0' to the Enable bit (ENABLE) in Control A register (CCL.CTRLA).

**Related Links**

CLKCTRL - Clock Controller

#### 28.2.3.2 I/O Lines

The CCL can take inputs and generate output through I/O pins. For this to function properly, the I/O pins must be configured to be used by a Look-up Table (LUT).

**Related Links**

PORT - I/O Pin Configuration

#### 28.2.3.3 Interrupts

Not applicable.

#### 28.2.3.4 Events

The CCL can use events from other peripherals and generate events that can be used by other peripherals. For this feature to function, the Events have to be configured properly. Refer to the Related Links below for more information about the Event Users and Event Generators.

**Related Links**

EVSYS - Event System

#### 28.2.3.5 Debug Operation

When the CPU is halted in Debug mode the CCL continues normal operation. However, the CCL cannot be halted when the CPU is halted in Debug mode. If the CCL is configured in a way that requires it to be periodically serviced by the CPU, improper operation or data loss may result during debugging.

## 28.3 Functional Description

### 28.3.1 Initialization

The following bits are enable-protected, meaning that they can only be written when the corresponding even LUT is disabled (ENABLE=0 in CCL.LUT0CTRLA):

- Sequential Selection (SEQSEL) in Sequential Control 0 register (CCL.SEQCTRL0)

The following registers are enable-protected, meaning that they can only be written when the corresponding LUT is disabled (ENABLE=0 in CCL.LUT0CTRLA):

- LUT n Control x register, except ENABLE bit (CCL.LUTnCTRLx)

Enable-protected bits in the CCL.LUTnCTRLx registers can be written at the same time as ENABLE in CCL.LUTnCTRLx is written to '1', but not at the same time as ENABLE is written to '0'.

Enable-protection is denoted by the Enable-Protected property in the register description.

### 28.3.2 Operation

#### 28.3.2.1 Enabling, Disabling and Resetting

The CCL is enabled by writing a '1' to the ENABLE bit in the Control register (CCL.CTRLA). The CCL is disabled by writing a '0' to that ENABLE bit.

Each LUT is enabled by writing a '1' to the LUT Enable bit (ENABLE) in the LUT n Control A register (CCL.LUTnCTRLA). Each LUT is disabled by writing a '0' to the ENABLE bit in CCL.LUTnCTRLA.

#### 28.3.2.2 Lookup Table Logic

The lookup table in each LUT unit can generate a combinational logic output as a function of up to three inputs IN[2:0]. Unused inputs can be masked (tied low). The truth table for the combinational logic expression is defined by the bits in the CCL.TRUTHn registers. Each combination of the input bits (IN[2:0]) corresponds to one bit in the TRUTHn register, as shown in the table:

**Table 28-2. Truth Table of LUT**

| IN[2] | IN[1] | IN[0] | OUT |
|---|---|---|---|
| 0 | 0 | 0 | TRUTH[0] |
| 0 | 0 | 1 | TRUTH[1] |
| 0 | 1 | 0 | TRUTH[2] |
| 0 | 1 | 1 | TRUTH[3] |
| 1 | 0 | 0 | TRUTH[4] |
| 1 | 0 | 1 | TRUTH[5] |
| 1 | 1 | 0 | TRUTH[6] |
| 1 | 1 | 1 | TRUTH[7] |

#### 28.3.2.3 Truth Table Inputs Selection

**Input Overview**

The inputs can be individually:

- Masked
- Driven by peripherals:
  - Analog comparator output (AC)
  - Timer/Counters waveform outputs (TC)
- Driven by internal events from Event System
- Driven by other CCL sub-modules

The Input Selection for each input y of LUT n is configured by writing the Input y Source Selection bit in the LUT n Control x=[B,C] registers

- INSEL0 in CCL.LUTnCTRLB
- INSEL1 in CCL.LUTnCTRLB
- INSEL2 in CCL.LUTnCTRLC.

**Internal Feedback Inputs (FEEDBACK)**
When selected (INSELy=FEEDBACK in CCL.LUTnCTRLx), the Sequential (SEQ) output is used as input for the corresponding LUT.

The output from an internal sequential module can be used as input source for the LUT, see figure below for an example for LUT0 and LUT1. The sequential selection for each LUT follows the formula:

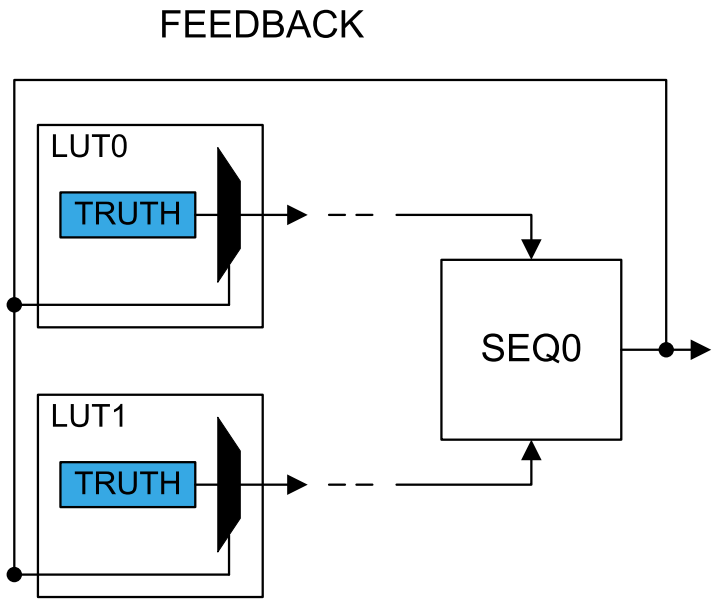
$\mathrm{IN}[2\mathrm{N}][i] = \mathrm{SEQ}[N]$

$\mathrm{IN}[2\mathrm{N}+1][i] = \mathrm{SEQ}[N]$

With *N* representing the sequencer number and *i*=0,1 representing the LUT input index.

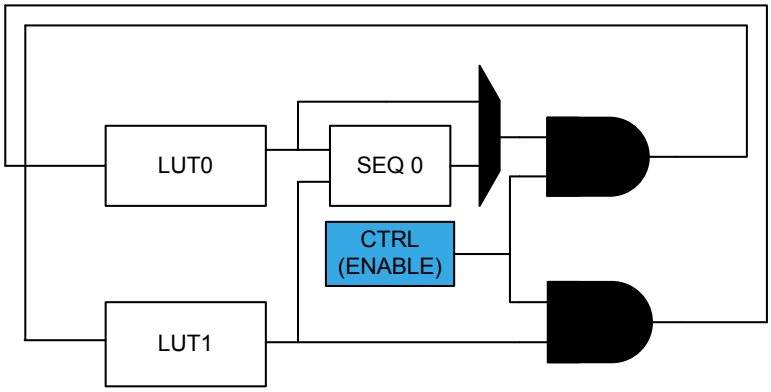
For details, refer to Sequential Logic.

**Figure 28-2. Feedback Input Selection**




**Linked LUT (LINK)**
When selecting the LINK input option, the next LUT's direct output is used as the LUT input. In general, LUT[n+1] is linked to the input of LUT[n]. As example LUT1 is the input for LUT0.

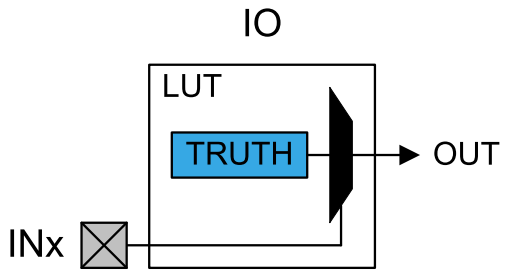
**Figure 28-3. Linked LUT Input Selection**



**Internal Events Inputs Selection (EVENT)**
Asynchronous events from the Event System can be used as input to the LUT. Two event input lines (EVENT0 and EVENT1) are available, and can be selected as LUT input. Before selecting the EVENT input option by writing to the LUT CONTROL A or B register (CCL.LUTnCTRLB or LUTnCTRLC), the Event System must be configured.

**I/O Pin Inputs (IO)**
When selecting the IO option, the LUT input will be connected to its corresponding IO pin. Refer to the I/O Multiplexing section for more details about where the LUTnINy are located.

**Figure 28-4. I/O Pin Input Selection**




**Peripherals**
The different peripherals on the three input lines of each LUT are selected by writing to the respective LUT n Input y bit fields in the LUT n Control B and C registers:

- INSEL0 in CCL.LUTnCTRLB
- INSEL1 in CCL.LUTnCTRLB
- INSEL2 in CCL.LUTnCTRLC.

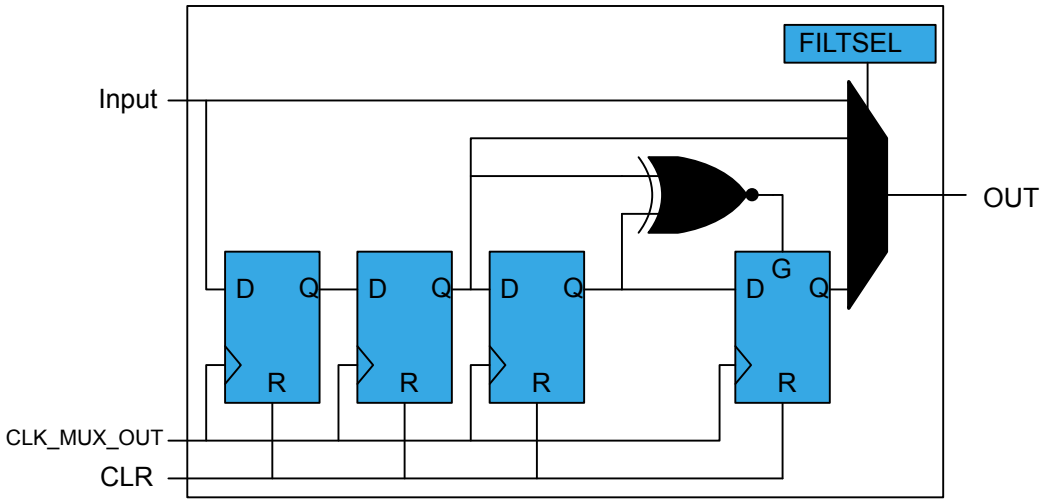
**Related Links**
I/O Multiplexing and Considerations
PORT - I/O Pin Configuration
CLKCTRL - Clock Controller
AC – Analog Comparator
TCA - 16-bit Timer/Counter Type A
TCD - 12-bit Timer/Counter Type D
USART - Universal Synchronous and Asynchronous Receiver and Transmitter
SPI - Serial Peripheral Interface
TWI - Two Wire Interface
I/O Multiplexing and Considerations

### 28.3.2.4 Filter

By default, the LUT output is a combinational function of the LUT inputs. This may cause some short glitches when the inputs change value. These glitches can be removed by clocking through filters, if demanded by application needs.

The Filter Selection bits (FILTSEL) in the LUT Control registers (CCL.LUTnCTRLA) define the digital filter options. When a filter is enabled, the output will be delayed by two to five CLK cycles (peripheral clock or alternative clock). One clock cycle after the corresponding LUT is disabled, all internal filter logic is cleared.

**Figure 28-5. Filter**



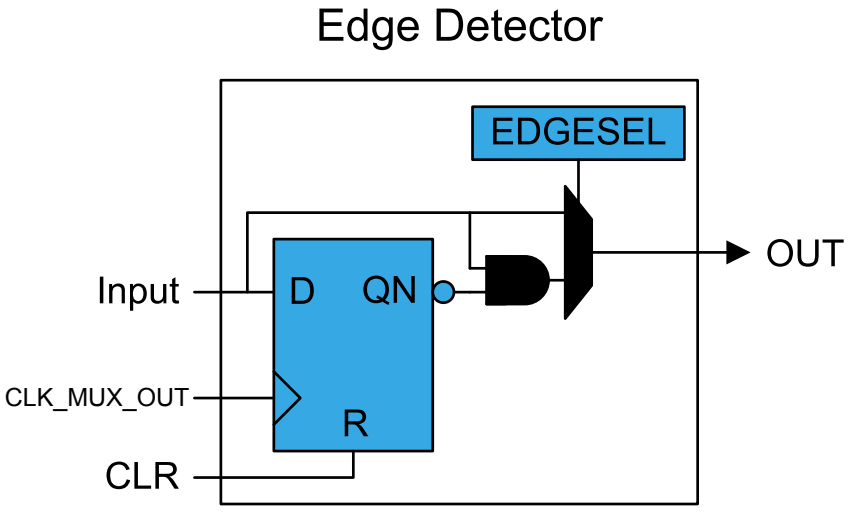

### 28.3.2.5 Edge Detector

The edge detector can be used to generate a pulse when detecting a rising edge on its input. To detect a falling edge, the TRUTH table should be programmed to provide inverted output.

The edge detector is enabled by writing '1' to the Edge Selection bit (EDGEDET) in the LUT n Control A register (CCL.LUTnCTRLA). In order to avoid unpredictable behavior, a valid filter option must be enabled as well.

Edge detection is disabled by writing a '0' to EDGEDET in CCL.LUTnCTRLA. After disabling a LUT, the corresponding internal Edge Detector logic is cleared one clock cycle later.

**Figure 28-6. Edge Detector**




### 28.3.2.6 Sequential Logic

Each LUT pair can be connected to an internal Sequential block. A Sequential block can function as either D flip-flop, JK flip-flop, gated D-latch, or RS-latch. The function is selected by writing the Sequential Selection bits (SEQSEL) in Sequential Control register (CCL.SEQCTRLn).

The Sequential block receives its input from the either LUT, filter or edge detector, depending on the configuration.

The Sequential block is clocked by the same clock as the corresponding LUT, which is either the peripheral clock or input line 2 (IN[2]). This is configured by the Clock Source bit (CLKSRC) in the LUT n Control A register (CCL.LUTnCTRLA).

When the even LUT (LUT0) is disabled, the latch is asynchronously cleared, during which the flip-flop reset signal (R) is kept enabled for one clock cycle. In all other cases, the flip-flop output (OUT) is refreshed on rising edge of the clock, as shown in the respective *Characteristics* tables.

**Gated D Flip-Flop (DFF)**

The D-input is driven by the even LUT output (LUT0), and the G-input is driven by the odd LUT output (LUT1).

**Figure 28-7. D Flip-Flop**

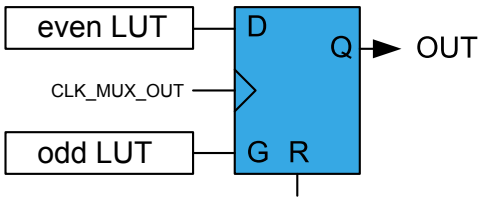



**Table 28-3. DFF Characteristics**

| R | G | D | OUT |
|---|---|---|---|
| 1 | X | X | Clear |
| 0 | 1 | 1 | Set |
| | | 0 | Clear |
| | 0 | X | Hold state (no change) |

**JK Flip-Flop (JK)**

The J-input is driven by the even LUT output (LUT0), and the K-input is driven by the odd LUT output (LUT1).

**Figure 28-8. JK Flip-Flop**

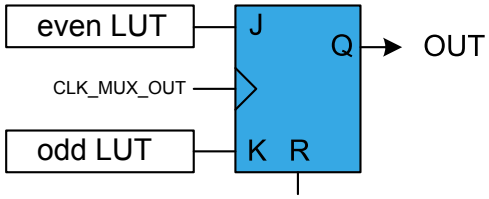



**Table 28-4. JK Characteristics**

| R | J | K | OUT |
|---|---|---|---|
| 1 | X | X | Clear |
| 0 | 0 | 0 | Hold state (no change) |
| 0 | 0 | 1 | Clear |
| 0 | 1 | 0 | Set |
| 0 | 1 | 1 | Toggle |

**Gated D-Latch (DLATCH)**

The D-input is driven by the even LUT output (LUT0), and the G-input is driven by the odd LUT output (LUT1).

**Figure 28-9. D-Latch**

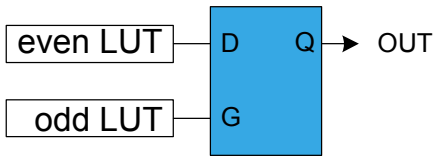



**Table 28-5. D-Latch Characteristics**

| G | D | OUT |
|---|---|---|
| 0 | X | Hold state (no change) |
| 1 | 0 | Clear |
| 1 | 1 | Set |

**RS Latch (RS)**

The S-input is driven by the even LUT output (LUT0), and the R-input is driven by the odd LUT output (LUT1).

**Figure 28-10. RS-Latch**

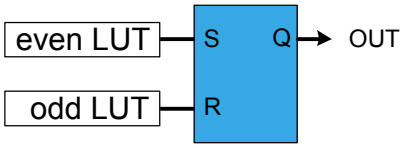


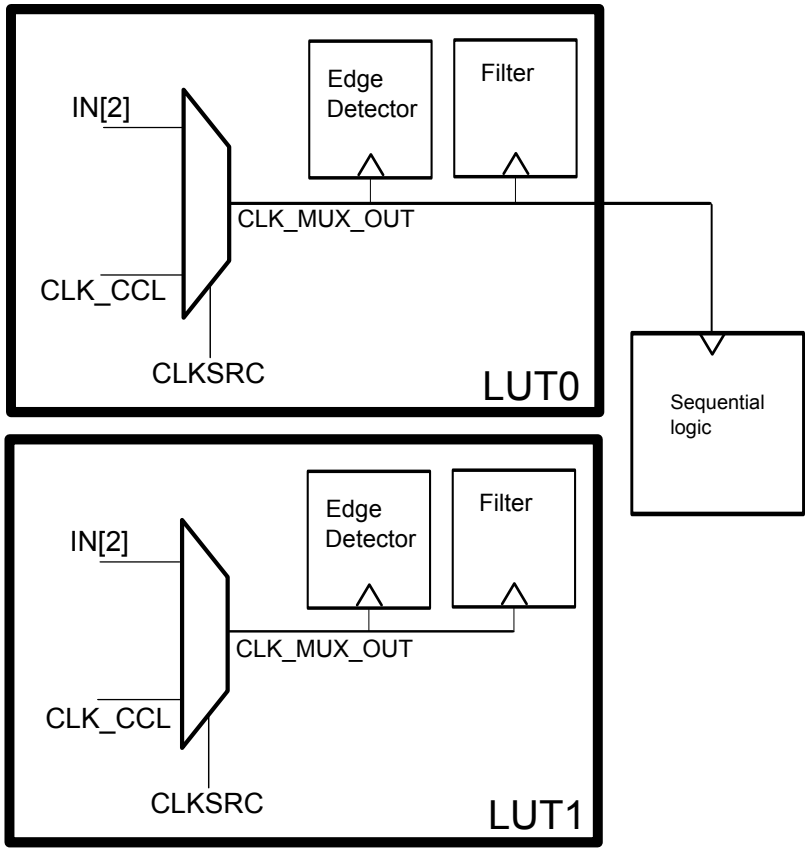

**Table 28-6. RS-latch Characteristics**

| S | R | OUT |
|---|---|---|
| 0 | 0 | Hold state (no change) |
| 0 | 1 | Clear |
| 1 | 0 | Set |
| 1 | 1 | Forbidden state |

#### 28.3.2.7 Clock Source Settings

The Filter, Edge Detector and Sequential logic are by default clocked by the system clock (CLK_PER). It is also possible to use the LUT input 2 (IN[2]) to clock these blocks (CLK_MUX_OUT in figure Figure 28-11). This is configured by writing the Clock Source bit (CLKSRC) in the LUT Control A register (CCL.LUTnCTRLA) to '1'.

**Figure 28-11. Clock Source Settings**




When the Clock Source bit (CLKSRC) is '1', IN[2] is used to clock the corresponding Filter and Edge Detector (CLK_MUX_OUT). The Sequential logic is clocked by CLK_MUX_OUT of the even LUT in the pair. When CLKSRC bit is '1', IN[2] is treated as MASKed (low) in the TRUTH table.

The CCL peripheral must be disabled while changing the clock source to avoid undetermined outputs from the peripheral.

### 28.3.3 Events

The CCL can generate the following output events:

- LUTnOUT: Lookup Table Output Value

The CCL can take the following actions on an input event:

- INx: The event is used as input for the TRUTH table.

**Related Links**

EVSYS - Event System

### 28.3.4 Sleep Mode Operation

Writing the Run In Standby bit (RUNSTDBY) in the Control A register (CCL.CTRLA) to '1' will allow the system clock to be enabled in Standby sleep mode.

If RUNSTDBY is '0' the system clock will be disabled in Standby sleep mode. If the Filter, Edge Detector or Sequential logic are enabled, the LUT output will be forced to zero in Standby sleep mode. In Idle sleep mode, the TRUTH table decoder will continue operation and the LUT output will be refreshed accordingly, regardless of the RUNSTDBY bit.

© 2017 Microchip Technology Inc.

If the Clock Source bit (CLKSRC) in the LUT n Control A register (CCL.LUTnCTRLA) is written to '1', the LUT input 2 (IN[2]) will always clock the Filter, Edge Detector and Sequential block. The availability of the IN[2] clock in sleep modes will depend on the sleep settings of the peripheral employed.

### 28.3.5 Configuration Change Protection

Not applicable.

## 28.4 Register Summary - CCL

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | | RUNSTDBY | | | | | | ENABLE |
| 0x01 | SEQCTRL0 | 7:0 | | | | | SEQSEL[3:0] | | | |
| 0x02 ... 0x04 | Reserved | | | | | | | | | |
| 0x05 | LUT0CTRLA | 7:0 | EDGEDET | CLKSRC | FILTSEL[1:0] | | OUTEN | | | ENABLE |
| 0x06 | LUT0CTRLB | 7:0 | INSEL1[3:0] | | | | INSEL0[3:0] | | | |
| 0x07 | LUT0CTRLC | 7:0 | | | | | INSEL2[3:0] | | | |
| 0x08 | TRUTH0 | 7:0 | TRUTH[7:0] | | | | | | | |
| 0x09 | LUT1CTRLA | 7:0 | EDGEDET | CLKSRC | FILTSEL[1:0] | | OUTEN | | | ENABLE |
| 0x0A | LUT1CTRLB | 7:0 | INSEL1[3:0] | | | | INSEL0[3:0] | | | |
| 0x0B | LUT1CTRLC | 7:0 | | | | | INSEL2[3:0] | | | |
| 0x0C | TRUTH1 | 7:0 | TRUTH[7:0] | | | | | | | |

## 28.5 Register Description

### 28.5.1 Control A

**Name:** CTRLA
**Offset:** 0x00 [ID-00000485]
**Reset:** 0x00
**Property:** -

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

**Bit 6 – RUNSTDBY: Run in Standby**
This bit indicates if the peripheral clock (CLK_PER) is kept running in Standby sleep mode. The setting is ignored for configurations where the CLK_PER is not required. For details refer to Sleep Mode Operation.

| Value | Description |
|---|---|
| 0 | System clock is not required in standby sleep mode. |
| 1 | System clock is required in standby sleep mode. |

**Bit 0 – ENABLE: Enable**

| Value | Description |
|---|---|
| 0 | The peripheral is disabled. |
| 1 | The peripheral is enabled. |

### 28.5.2 Sequential Control 0

**Name:** SEQCTRL0
**Offset:** 0x01 [ID-00000485]
**Reset:** 0x00
**Property:** Enable-Protected

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

**Bits 3:0 – SEQSEL[3:0]: Sequential Selection**
These bits select the sequential configuration.

| Value | Name | Description |
|---|---|---|
| 0x0 | DISABLE | Sequential logic is disabled |
| 0x1 | DFF | D flip flop |
| 0x2 | JK | JK flip flop |
| 0x3 | LATCH | D latch |
| 0x4 | RS | RS latch |
| Other | - | Reserved |

### 28.5.3 LUT n Control A

**Name:** LUT0CTRLA, LUT1CTRLA
**Offset:** 0x05 + n*0x04 [n=0..1]
**Reset:** 0x00
**Property:** Enable-Protected

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | EDGEDET | CLKSRC | FILTSEL[1:0] | | OUTEN | | | ENABLE |
| Access | R/W | R/W | R/W | R/W | R/W | | | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | | | 0 |

**Bit 7 – EDGEDET: Edge Detection**

| Value | Description |
|---|---|
| 0 | Edge detector is disabled. |
| 1 | Edge detector is enabled. |

**Bit 6 – CLKSRC: Clock Source Selection**
This bit selects whether the peripheral clock (CLK_PER) or any input present on input line 2 (IN[2]) is used as clock (CLK_MUX_OUT) for a LUT.

The CLK_MUX_OUT of the even LUT is used for clocking the Sequential block of a LUT pair.

| Value | Description |
|---|---|
| 0 | CLK_PER is clocking the LUT. |
| 1 | IN[2] is clocking the LUT. |

**Bits 5:4 – FILTSEL[1:0]: Filter Selection**
These bits select the LUT output filter options:

Filter Selection

| Value | Name | Description |
|---|---|---|
| 0x0 | DISABLE | Filter disabled |
| 0x1 | SYNCH | Synchronizer enabled |
| 0x2 | FILTER | Filter enabled |
| 0x3 | - | Reserved |

**Bit 3 – OUTEN: Output Enable**
This bit enables the LUT output to the LUTnOUT pin. When written to '1', the pin configuration of the PORT I/O-Controller is overridden.

| Value | Description |
|---|---|
| 0 | Output to pin disabled. |
| 1 | Output to pin enabled. |

**Bit 0 – ENABLE: LUT Enable**

| Value | Description |
|---|---|
| 0 | The LUT is disabled. |
| 1 | The LUT is enabled. |

### 28.5.4 LUT n Control B

SPI connections to the CCL work only in master SPI mode.

USART connections to the CCL work only in asynchronous/synchronous USART master mode.

**Name:** LUT0CTRLB, LUT1CTRLB
**Offset:** 0x06 + n*0x04 [n=0..1]
**Reset:** 0x00
**Property:** Enable-Protected

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | INSEL1[3:0] | | | | INSEL0[3:0] | | | |
| 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 7:4 – INSEL1[3:0]: LUT n Input 1 Source Selection**
These bits select the source for input 1 of LUT n:

| Value | Name | Description |
|---|---|---|
| 0x0 | MASK | Masked input |
| 0x1 | FEEDBACK | Feedback input |
| 0x2 | LINK | Linked other LUT as input source |
| 0x3 | EVENT0 | Event 0 as input source for LUT0 |
| 0x4 | EVENT1 | Event 1 as input source for LUT1 |
| 0x5 | IO | I/O pin LUTn-IN1 input source |
| 0x6 | AC0 | AC0 OUT input source |

| Value | Name | Description |
|---|---|---|
| 0x7 | TCB0 | TCB WO input source |
| 0x8 | TCA0 | TCA WO1 input source |
| 0x9 | TCD0 | TCD WOB input source |
| 0xA | USART0 | USART TXD input source |
| 0xB | SPI0 | SPI MOSI input source |

**Bits 3:0 – INSEL0[3:0]: LUT n Input 0 Source Selection**
These bits select the source for input 0 of LUT n:

| Value | Name | Description |
|---|---|---|
| 0x0 | MASK | Masked input |
| 0x1 | FEEDBACK | Feedback input |
| 0x2 | LINK | Linked other LUT as input source |
| 0x3 | EVENT0 | Event 0 as input source for LUT0 |
| 0x4 | EVENT1 | Event 1 as input source for LUT1 |
| 0x5 | IO | I/O pin LUTn-IN0 input source |
| 0x6 | AC0 | AC0 OUT input source |
| 0x7 | TCB0 | TCB WO input source |
| 0x8 | TCA0 | TCA WO0 input source |
| 0x9 | TCD0 | TCD WOA input source |
| 0xA | USART0 | USART XCK input source |
| 0xB | SPI0 | SPI SCK input source |
| Other | - | Reserved |

### 28.5.5 LUT n Control C

**Name:** LUT0CTRLC, LUT1CTRLC
**Offset:** 0x07 + n*0x04 [n=0..1]
**Reset:** 0x00
**Property:** Enable-Protected

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | INSEL2[3:0] | | | |
| Access | | | | | R/W | R/W | R/W | R/W |
| Reset | | | | | 0 | 0 | 0 | 0 |

**Bits 3:0 – INSEL2[3:0]: LUT n Input 2 Source Selection**
These bits select the source for input 2 of LUT n:

| Value | Name | Description |
|---|---|---|
| 0x0 | MASK | Masked input |
| 0x1 | FEEDBACK | Feedback input |
| 0x2 | LINK | Linked other LUT as input source |
| 0x3 | EVENT0 | Event input source 0 |
| 0x4 | EVENT1 | Event input source 1 |
| 0x5 | IO | I/O pin LUTn-IN2 input source |
| 0x6 | AC0 | AC0 OUT input source |
| 0x7 | TCB0 | TCB WO input source |

| Value | Name | Description |
|---|---|---|
| 0x8 | TCA0 | TCA WO2 input source |
| 0x9 | TCD0 | TCD WOA input source |
| 0xA | - | Reserved |
| 0xB | SPI0 | SPI MISO input source |
| other | - | Reserved |

### 28.5.6 TRUTHn

**Name:** TRUTH0, TRUTH1
**Offset:** 0x08 + n*0x04 [n=0..1]
**Reset:** 0x00
**Property:** Enable-Protected

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TRUTH[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 – TRUTH[7:0]: Truth Table**
These bits define the value of truth logic as a function of inputs IN[2:0].

# 29. AC – Analog Comparator

## 29.1 Features

- One instance of the AC controller, AC0
- 50ns response time for supply voltage above 2.7V
- Zero-cross detection
- Selectable hysteresis:
  - None
  - 10mV
  - 25mV
  - 50mV
- Analog comparator output available on pin
- Comparator output inversion available
- Flexible input selection:
  - 1 Positive pins
  - 1 Negative pins
  - Output from the DAC
  - Internal reference voltage
- Interrupt generation on:
  - Rising edge
  - Falling edge
  - Both egdes
- Event generation:
  - Comparator output

## 29.2 Overview

The Analog Comparator (AC) compares the voltage levels on two inputs and gives a digital output based on this comparison. The AC can be configured to generate interrupt requests and/or Events upon several different combinations of input change.

The dynamic behavior of the AC can be adjusted by a hysteresis feature. The hysteresis can be customized to optimize the operation for each application.

The input selection includes analog port pins, DAC output and internal references. The analog comparator output state can also be output on a pin for use by external devices.

The AC has one positive input and one negative input. The positive input source is one of a selection of 1 analog input pins. The negative input is chosen either from analog input pins or from internal inputs, such as an internal voltage reference.

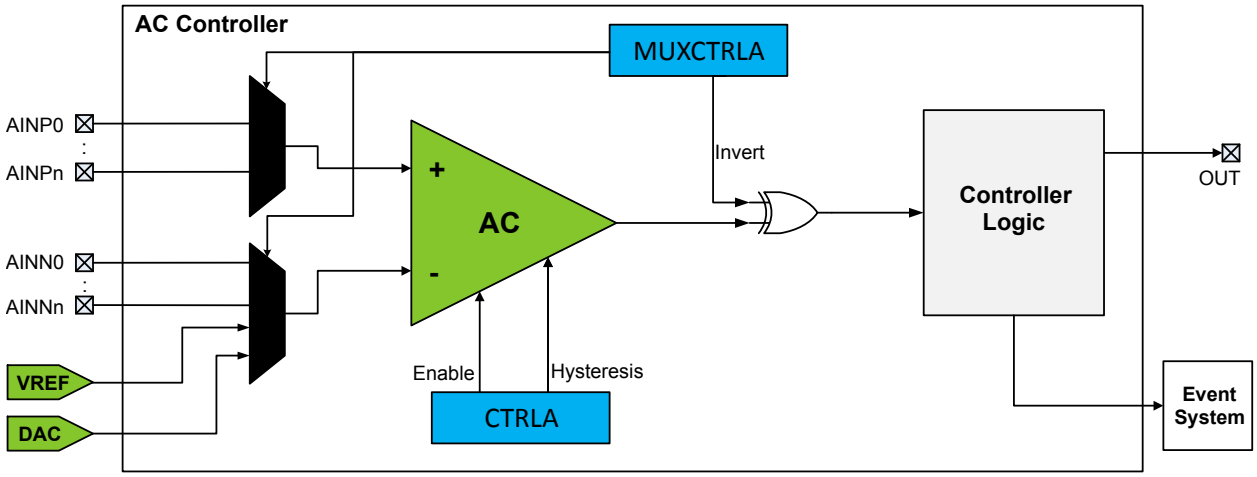
The digital output from the comparator is '1' when the difference between the positive and the negative input voltage is positive, and '0' otherwise.

This device provides one instance of the AC controller, AC0.

© 2017 Microchip Technology Inc.

### 29.2.1 Block Diagram

**Figure 29-1. Analog Comparator**




**Note:** Refer to Signal Description for the number of AINN and AINP.

### 29.2.2 Signal Description

| Signal | Description | Type |
|---|---|---|
| AINN0 | Negative Input 0 | Analog |
| AINP0 | Positive Input 0 | Analog |
| OUT | Comparator Output for AC | Digital |

### 29.2.3 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 29-1. AC System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | Yes | PORT |
| Interrupts | Yes | CPUINT |
| Events | Yes | EVSYS |
| Debug | Yes | UPDI |

#### 29.2.3.1 Clocks

This peripheral depends on the peripheral clock.

#### 29.2.3.2 I/O Lines and Connections

I/O pins AINN0, AINP0 are all analog inputs to the AC.

For correct operation, the pins must be configured in the Port and Port Multiplexing peripherals.

It is recommended to disable the GPIO input when using the AC.

#### 29.2.3.3 Interrupts

Using the interrupts of this peripheral requires the Interrupt Controller to be configured first.

#### 29.2.3.4 Events

The events of this peripheral are connected to the Event System.

#### 29.2.3.5 Debug Operation

This peripheral is unaffected by entering debug mode.

If the peripheral is configured to require periodical service by the CPU through interrupts or similar, improper operation or data loss may result during halted debugging.

## 29.3 Functional Description

### 29.3.1 Initialization

For basic operation, follow these steps:

- Configure the desired input pins in the PORT peripheral.
- Select the positive and negative input sources by writing the Positive and Negative Input MUX Selection bit fields (MUXPOS and MUXNEG) in the MUX Control A register (AC.MUXCTRLA).
- Optional: Enable the output to pin by writing a '1' to the Output Pad Enable bit (OUTEN) in the Control A register (AC.CTRLA).
- Enable the AC by writing a '1' to the ENABLE bit in AC.CTRLA.

During the start-up time after enabling the AC, the output of the AC may be invalid.

The start-up time of the AC by itself is at most 2.5µs. If a internal reference is used, the reference start-up time is normally longer than the AC start-up time. The VREF start-up time is at most 60µs.

### 29.3.2 Operation

#### 29.3.2.1 Input Hysteresis

Applying an input hysteresis helps preventing constant toggling of the output when the noise-afflicted input signals are close to each other.

The input hysteresis can either be disabled or have one of three levels. The hysteresis is configured by writing to the Hysteresis Mode Select bit field (HYSMODE) in the Control A register (AC.CTRLA).

#### 29.3.2.2 Input Sources

The AC has one positive and one negative input. The inputs can be pins and internal sources, such as a voltage reference.

Each input is selected by writing to the Positive and Negative Input MUX Selection bit field (MUXPOS and MUXNEG) in the MUX Control A register (AC.MUXTRLA).

**Pin Inputs**

The following Analog input pins on the port can be selected as input to the analog comparator

- AINN0
- AINP0

**Internal Inputs**

Two internal inputs are available for the analog comparator:

- Output from the DAC
- DAC and AC voltage reference

#### 29.3.2.3 Low Power Mode

For power-conscious applications, the AC provides a Low Power with reduced power consumption and increased propagation delay.

This mode is enabled by writing a '1' to the Low Power Mode bit (LPMODE) in the Control A register (AC.CTRLA).

### 29.3.3 Events

The AC will generate the following Event automatically when the AC is enabled:

- Comparator output Event

### 29.3.4 Interrupts

**Table 29-2. Available Interrupt Vectors and Sources**

| Offset | Name | Vector Description | Conditions |
|---|---|---|---|
| 0x00 | COMP0 | Analog comparator interrupt | AC output is toggling as configured by INTMODE in AC.CTRLA |

When an interrupt condition occurs, the corresponding Interrupt Flag is set in the Status register (AC.STATUS).

An interrupt source is enabled or disabled by writing to the corresponding bit in the peripheral's Interrupt Control register (AC.INTCTRL).

An interrupt request is generated when the corresponding interrupt source is enabled and the Interrupt Flag is set. The interrupt request remains active until the Interrupt Flag is cleared. See the AC.STATUS register description for details on how to clear Interrupt Flags.

### 29.3.5 Sleep Mode Operation

In Idle sleep mode the AC will continue to operate normal.

In Standby sleep mode, the AC is disabled by default. If the Run in Standby Sleep Mode bit (RUNSTDBY) in the Control A register (AC.CTRLA) is written to '1', the AC will continue to operate, but the Status register will not be updated, and no Interrupts are generated if not any other modules request the CLK_PER, but events and the pad output will be updated.

In Power Down sleep mode, the AC and the output to pad are disabled.

### 29.3.6 Configuration Change Protection

Not applicable.

## 29.4 Register Summary - AC

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | RUNSTDBY | OUTEN | INTMODE[1:0] | | LPMODE | HYSMODE[1:0] | | ENABLE |
| 0x01 | Reserved | | | | | | | | | |
| 0x02 | MUXCTRLA | 7:0 | INVERT | | | | | | MUXNEG[1:0] | |
| 0x03<br>...<br>0x05 | Reserved | | | | | | | | | |
| 0x06 | INTCTRL | 7:0 | | | | | | | | CMP |
| 0x07 | STATUS | 7:0 | | | | STATE | | | | CMP |

## 29.5 Register Description

### 29.5.1 Control A

**Name:** CTRLA
**Offset:** 0x00
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RUNSTDBY | OUTEN | INTMODE[1:0] | | LPMODE | HYSMODE[1:0] | | ENABLE |
| 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 – RUNSTDBY: Run in Standby Mode**
Writing a '1' to this bit allows the AC to continue operation in Standby sleep mode. Since the clock is stopped, interrupts and status flags are not updated.

| Value | Description |
|---|---|
| 0 | In Standby sleep mode, the peripheral is halted |
| 1 | In Standby sleep mode, the peripheral continues operation |

**Bit 6 – OUTEN: Analog Comparator Output Pad Enable**
Writing this bit to '1' makes the OUT signal available on the pin.

**Bits 5:4 – INTMODE[1:0]: Interrupt Modes**
Writing to these bits selects what edge(s) of the AC output an interrupt request is triggered.

| Value | Name | Description |
|---|---|---|
| 0x0 | BOTHEDGE | Both negative and positive edge |
| 0x1 | - | Reserved |
| 0x2 | NEGEDGE | Negative edge |
| 0x3 | POSEDGE | Positive edge |

**Bit 3 – LPMODE: Low Power Mode**
Writing a '1' to this bit reduces the current through the comparator. This reduces power consumption, but increases the reaction time of the AC.

| Value | Description |
|---|---|
| 0 | Low Power Mode disabled |
| 1 | Low Power Mode enabled |

**Bits 2:1 – HYSMODE[1:0]: Hysteresis Mode Select**
Writing these bits selects the hysteresis mode for the AC input.

| Value | Name | Description |
|---|---|---|
| 0x0 | OFF | OFF |
| 0x1 | 10 | ±10mV |
| 0x2 | 25 | ±25mV |
| 0x3 | 50 | ±50mV |

**Bit 0 – ENABLE: Enable AC**
Writing this bit to '1' enables the AC.

### 29.5.2 Mux Control A

AC.MUXCTRLA controls Analog Comparator Muxes

**Name:** MUXCTRLA
**Offset:** 0x02
**Reset:** 0x00
**Property:** -

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

**Bit 7 – INVERT: Invert AC Output**
Writing a '1' to this bit enables inversion of the output of the AC. This effectively inverts the input to all the peripherals connected to the signal, and also affects the internal status signals.
Writing to this bit field selects the input signal to the positive input of the AC.

| Value | Name | Description |
|---|---|---|
| 0 | AINP0 | Positive Pin 0 |
| 1 | Reserved | Reserved |

**Bits 1:0 – MUXNEG[1:0]: Negative Input MUX Selection**
Writing to this bit field selects the input signal to the negative input of the AC.

| Value | Name | Description |
|---|---|---|
| 0x0 | AINN0 | Negative Pin 0 |
| 0x1 | Reserved | Reserved |
| 0x2 | VREF | Voltage Reference |
| 0x3 | DAC | DAC output. |

© 2017 Microchip Technology Inc.

### 29.5.3 Interrupt Control

**Name:** INTCTRL
**Offset:** 0x06
**Reset:** 0x00
**Property:** -

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

**Bit 0 – CMP: Analog Comparator Interrupt Enable**
Writing this bit to '1' enables Analog Comparator Interrupt.

### 29.5.4 Status

**Name:** STATUS
**Offset:** 0x07
**Reset:** 0x00
**Property:** -

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

**Bit 4 – STATE: Analog Comparator State**
This shows current status of the OUT signal from the AC. This will have a synchronizer delay to get updated in the I/O register (3 cycles).

**Bit 0 – CMP: Analog Comparator Interrupt Flag**
This is the interrupt flag for AC. Writing a '1' to this bit will clear the Interrupt flag.

# 30. ADC - Analog to Digital Converter

## 30.1 Features

- 10-bit resolution
- ±2 LSB absolute accuracy
- 6.5 - 260µs conversion time
- Up to 115ksps at 10-bit resolution (150ksps at 8-bit)
- Up to six multiplexed single-ended input channels
- Temperature sensor input channel
- 0V to $V_{DD}$ ADC input voltage range
- Multiple internal ADC reference voltage between 0.55V and $V_{DD}$
- Free running or Single Conversion mode
- Interrupt available on ADC conversion complete
- Optional Event triggered conversion
- Optional interrupt on conversion results
- Compare function for accurate monitoring or user defined thresholds (window comparator mode)
- Accumulation up to 64 samples per conversion

## 30.2 Overview

The Analog-to-Digital Converter (ADC) peripheral features a 10-bit successive approximation ADC (SAR), with a sampling rate up to 115ksps at 10-bit resolution (150ksps at 8-bit). The ADC is connected to a 6-channel Analog Multiplexer which allows twelve single-ended voltage inputs. The single-ended voltage inputs refer to 0V (GND). The ADC input channel can either be internal (e.g. a voltage reference) or external through the analog input pins.

An ADC conversion can be started by software, or by using the Event System (EVSYS) to route an event from other peripherals. Making it possible to do periodic sampling of input signals, trigger an ADC conversion on a special condition, or triggering an ADC conversion in standby sleep mode. A window compare feature is available for monitoring the input signal, and can be configured to only trigger an interrupt on user defined thresholds for under, over, inside or outside a window, with minimum software intervention required.

The ADC input signal is fed through a sample-and-hold circuit which ensures that the input voltage to the ADC is held at a constant level during sampling.

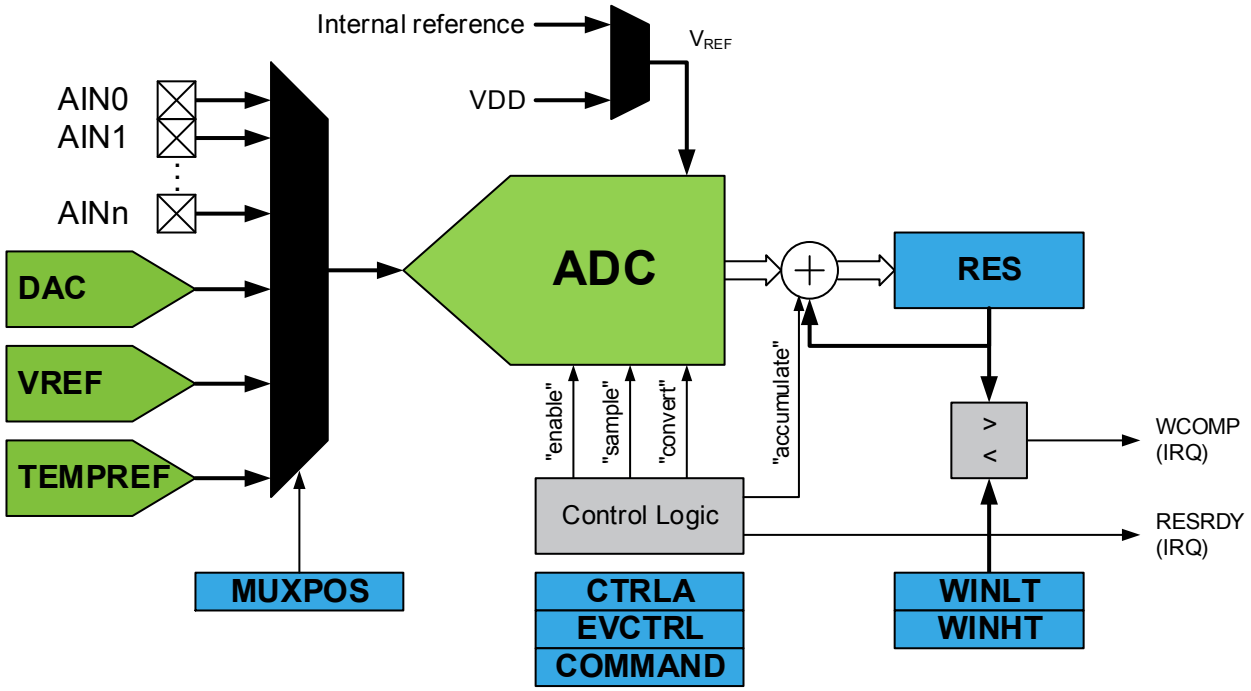
The ADC supports sampling in bursts where a configurable number of conversion results are accumulated into a single ADC result (Sample Accumulation). Further, a sample delay can be configured to tune the ADC sampling frequency associated with a single burst. This is to tune the sampling frequency away from any harmonic noise aliased with the ADC sampling frequency (within the burst) from the sampled signal. An automatic sampling delay variation feature can be used to randomize this delay to slightly change the time between samples.

Internal reference voltages from Voltage Reference (VREF) or VDD are provided on-chip.

This device has one instance of the ADC peripheral, ADC0.

### 30.2.1 Block Diagram

**Figure 30-1. Block Diagram**




The analog input channel is selected by writing to the MUXPOS bits in the MUXPOS register (ADC.MUXPOS). Any of the ADC input pins, GND, internal Voltage Reference (VREF), or temperature sensor, can be selected as single ended input to the ADC. The ADC is enabled by writing a '1' to the ADC ENABLE bit in the Control A register (ADC.CTRLA). Voltage reference and input channel selections will not go into effect before the ADC is enabled. The ADC does not consume power when the ENABLE bit in ADC.CTRLA is zero.

The ADC generates a 10-bit result which can be read from the Result Register (ADC.RES). The result is presented right adjusted.

### 30.2.2 Signal Description

| Pin Name | Type | Description |
|---|---|---|
| AIN[7,6, 3:0] | Analog input | analog input to be converted |

**Related Links**

Configuration Summary

I/O Multiplexing and Considerations

### 30.2.3 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 30-1. ADC System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | Yes | PORT |

| Dependency | Applicable | Peripheral |
|---|---|---|
| Interrupts | Yes | CPUINT |
| Events | Yes | EVSYS |
| Debug | Yes | UPDI |

#### 30.2.3.1 Clocks

The ADC uses the peripheral clock CLK_PER, and has an internal prescaler to generate the ADC clock source CLK_ADC.

**Related Links**

CLKCTRL - Clock Controller

Clock generation

#### 30.2.3.2 I/O Lines and Connections

The I/O-pins (AINx) are configured by the PORT - I/O Pin Controller.

The digital input buffer should be disabled on the pin used as input for the ADC to disconnect the digital domain from the analog domain to obtain the best possible ADC results. This is configured by the PORT - I/O Pin Controller.

**Related Links**

PORT - I/O Pin Configuration

#### 30.2.3.3 Interrupts

Using the interrupts of this peripheral requires the Interrupt Controller to be configured first.

**Related Links**

CPUINT - CPU Interrupt Controller

SREG

Interrupts

#### 30.2.3.4 Events

The events of this peripheral are connected to the Event System.

**Related Links**

EVSYS - Event System

#### 30.2.3.5 Debug Operation

When run-time debugging, this peripheral will continue normal operation. Halting the CPU in debugging mode will halt normal operation of the peripheral.

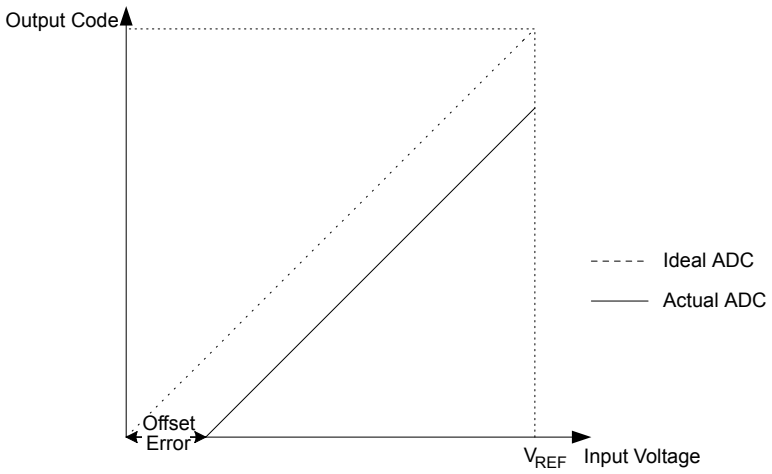
This peripheral can be forced to operate with halted CPU by writing a '1' to the Debug Run bit (DBGRUN) in the Debug Control register of the peripheral (*peripheral*.DBGCTRL).

### 30.2.4 Definitions

An ideal n-bit single-ended ADC converts a voltage linearly between GND and VREF 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 Error** | The deviation of the first transition (0x000 to 0x001) compared to the ideal transition (at 0.5 LSB). Ideal value: 0 LSB. |

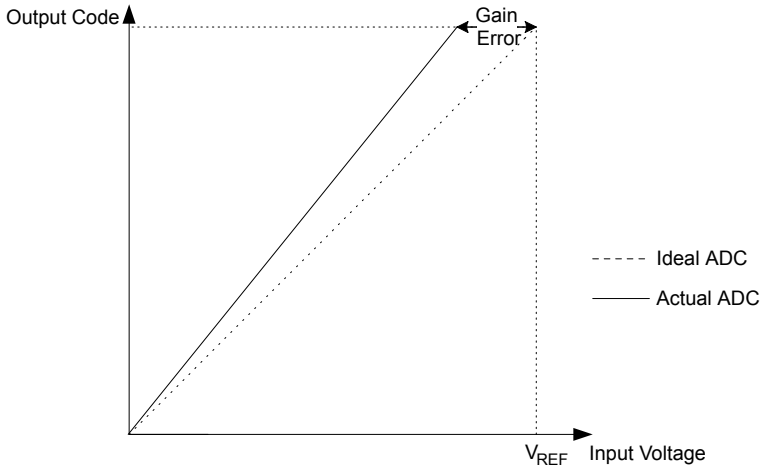
**Figure 30-2. Offset Error**




**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.

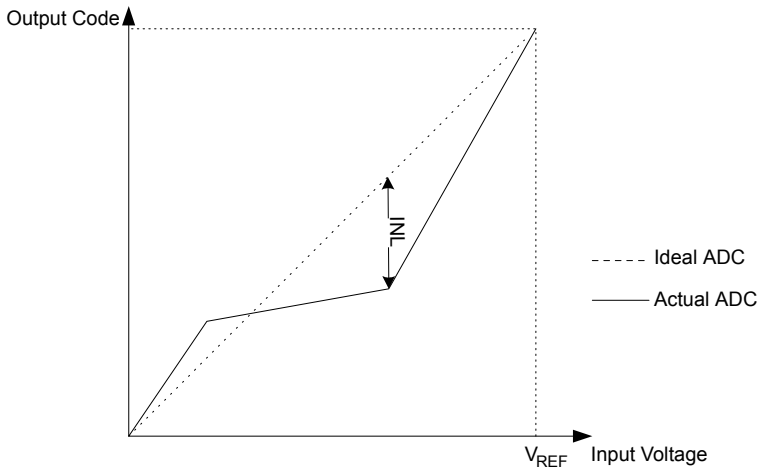
**Figure 30-3. Gain Error**




**Integral Non-Linearity (INL)**

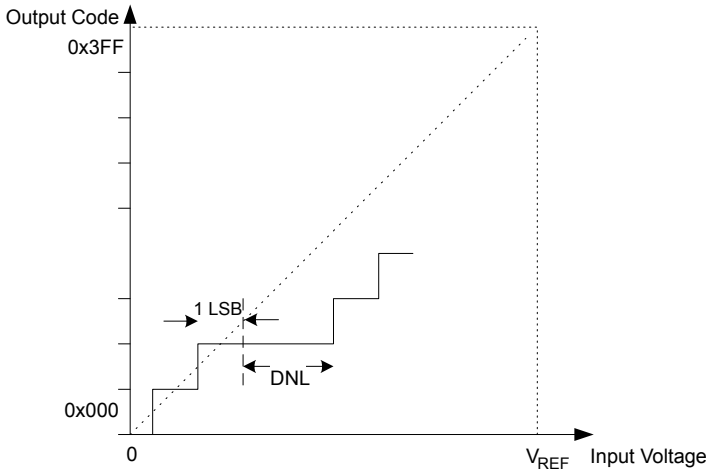
After adjusting for offset and gain error, the INL is the maximum deviation of an actual transition compared to an ideal transition for any code. Ideal value: 0 LSB.

**Figure 30-4. Integral Non-Linearity**




**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 30-5. Differential Non-Linearity**




**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 all aforementioned errors. Ideal value: ±0.5 LSB.

## 30.3 Functional Description

### 30.3.1 Initialization

The following steps are recommended in order to initialize ADC operation:

1. Configure the resolution by writing to the Resolution Selection bit (RESSEL) in the Control A register (ADC.CTRLA).
2. Optional: Enable the Free Running mode by writing a '1' to the Free Running bit (FREERUN) in ADC.CTRLA.

3. Optional: Configure the number of samples to be accumulated per conversion by writing the Sample Accumulation Number Select bits (SAMPNUM) in the Control B register (ADC.CTRLB).
4. Configure a voltage reference by writing to the Reference Selection bit (REFSEL) in the Control C register (ADC.CTRLC). Default is the internal Voltage Reference of the device (VREF, as configured there).
5. Configure the CLK_ADC by writing to the Prescaler bit field (PRESC) in the Control C register (ADC.CTRLC).
6. Configure an input by writing to the MUXPOS bit field in the MUXPOS register (ADC.MUXPOS).
7. Optional: Enable Start Event input by writing a '1' to the Start Event Input bit (STARTEI) in the Event Control register (ADC.EVCTRL). Configure the Event System accordingly.
8. Enable the ADC by writing a '1' to the ENABLE bit in ADC.CTRLA.

Following these steps will initialize the ADC for basic measurements, which can be triggered by an Event (if configured) or by writing a '1' to the Start Conversion bit (STCONV) in the Command register (ADC..COMMAND).

### 30.3.2 Operation

#### 30.3.2.1 Starting a Conversion

Once the input channel is selected by writing to the MUXPOS register (ADC.MUXPOS), a conversion is triggered by writing a '1' to the ADC Start Conversion bit (STCONV) in the Command register (ADC.COMMAND). This bit is one as long as the conversion is in progress. In single conversion mode, STCONV is cleared by hardware when the conversion is completed.

If a different input channel is selected while a conversion is in progress, the ADC will finish the current conversion before changing the channel.

Depending on the Accumulator setting, the conversion result is from a single sensing operation, or from a sequence of accumulated samples. Once the triggered operation is finished, the Result Ready flag (RESRDY) in the Interrupt Flag register (ADC.INTFLAG) is set. And the corresponding interrupt vector is executed if the Result Ready Interrupt Enable bit (RESRDY) in the Interrupt Control register (ADC.INTCTRL) is one and the Global Interrupt Enable bit is one.

A single conversion can be started by writing a '1' to the STCONV bit in ADC.COMMAND. The STCONV bit can be used to determine if a conversion is in progress, the STCONV bit will be set during a conversion, and cleared once the conversion is complete.

The RESRDY interrupt flag in ADC.INTFLAG will be set even if the specific interrupt is disabled, allowing software to check for finished conversion by polling the flag. A conversion can thus be triggered without causing an interrupt.

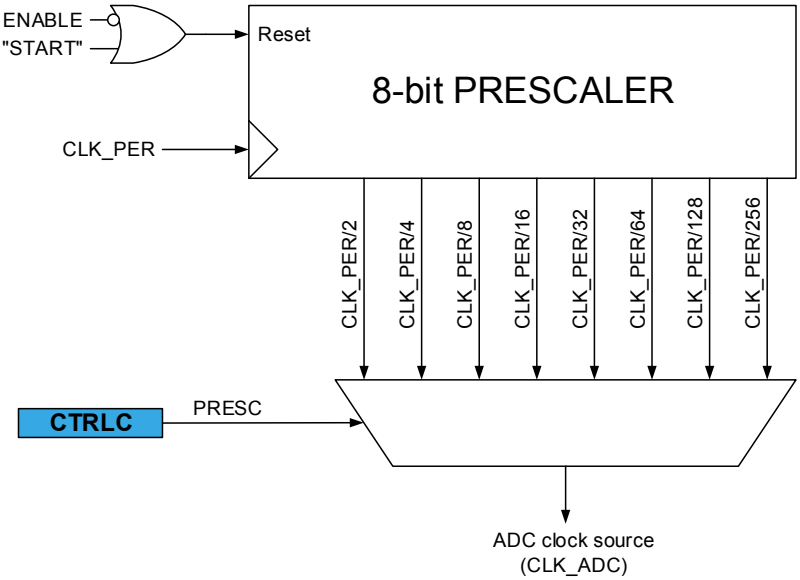
Alternatively, a conversion can be triggered by an event, this is enabled by writing a '1' to the Start Event Input bit (STARTEI) in the Event Control register (ADC.EVCTRL). Any incoming event routed to the ADC thorugh the Event System (EVSYS) will trigger an ADC conversion. This provides a method to start conversions at predictable intervals or at specific conditions.

The event trigger input is edge sensitive, when an event occurs, STCONV in ADC.COMMAND is set. STCONV will be cleared when the conversion is complete.

In Free Running mode, the first conversion is started by writing the STCONV bit to '1' in ADC.COMMAND. A new conversion cycle is started immediately after the previous conversion cycle has completed. A conversion complete will set the RESRDY flag in ADC.INTFLAGS.

### 30.3.2.2 Clock generation

**Figure 30-6. ADC Prescaler**



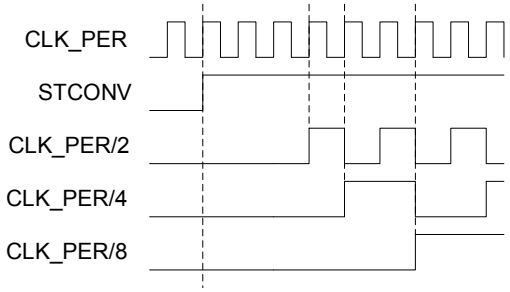

The ADC requires an input clock frequency between 50kHz and 1.5MHz for maximum resolution. If a lower resolution than 10 bits is selected, the input clock frequency to the ADC can be higher than 1.5MHz to get a higher sample rate.

The ADC module contains a prescaler which generates the ADC clock (CLK_ADC) from any CPU clock (CLK_PER) above 100kHz. The prescaling is selected by writing to the Prescaler bits (PRESC) in the Control C register (ADC.CTRLC). The prescaler starts counting from the moment the ADC is switched on by writing a '1' to the ENABLE bit in ADC.CTRLA. The prescaler keeps running as long as the ENABLE bit is one, the prescaler counter is reset to zero when the ENABLE bit is zero.

When initiating a conversion by writing a '1' to the Start Conversion bit (STCONV) in the Command register (ADC.COMMAND) or from event, the conversion starts at the following rising edge of the CLK_ADC clock cycle. The prescaler is kept reset as long as there is no ongoing conversion. This assures a fixed delay from the trigger to the actual start of a conversion in CLK_PER cycles as:

$$\text{StartDelay} = \frac{\text{PRESC}_{\text{factor}}}{2} + 2$$

**Figure 30-7. Start conversion and clock generation**

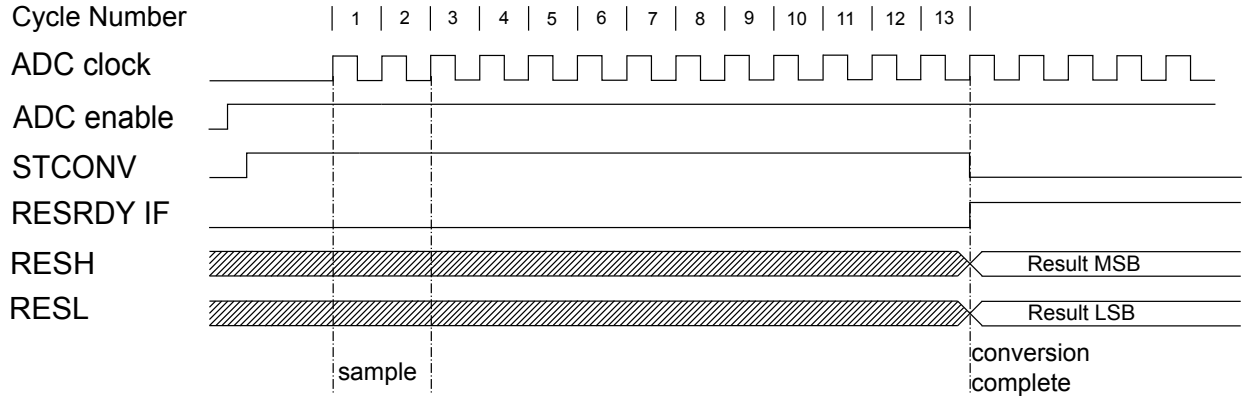



### 30.3.2.3 Conversion timing

A normal conversion takes 13 CLK_ADC cycles. The actual sample-and-hold takes place 2 CLK_ADC cycles after the start of a conversion. Start of conversion is initiated by writing a '1' to the STCONV bit in ADC.COMMAND. When a conversion is complete, the result is available in the Result register (ADC.RES), and the Result Ready interrupt flag is set (RESRDY in ADC.INTFLAG). The interrupt flag will

be cleared when the result is read from the Result registers, or by writing a '1' to the RESRDY bit in ADC.INTFLAG.

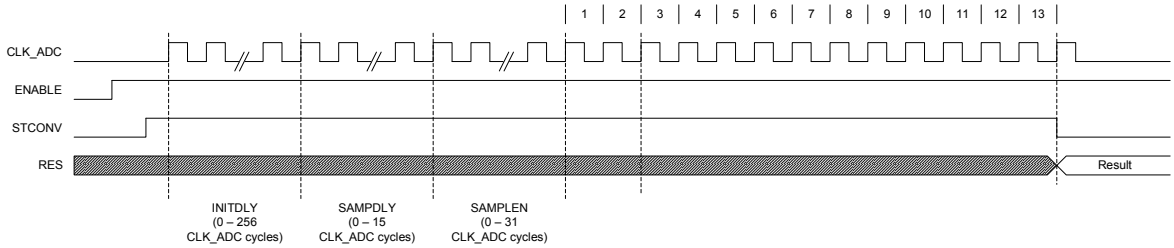
**Figure 30-8. ADC Timing Diagram - Single Conversion**




Both sampling time and sampling length can be adjusted using Sample Delay bit field in Control D (ADC.CTRLD) and sampling Sample Length bit field in the Sample Control register (ADC.SAMPCTRL). Both of these control the ADC sampling time in number of CLK_ADC cycles. This allows sampling high-impedance sources without relaxing conversion speed. See register description for further information. Total sampling time is given by:

$$\text{SampleTime} = \frac{(2 + \text{SAMPDLY} + \text{SAMPLEN})}{f_{\text{CLK_ADC}}}$$

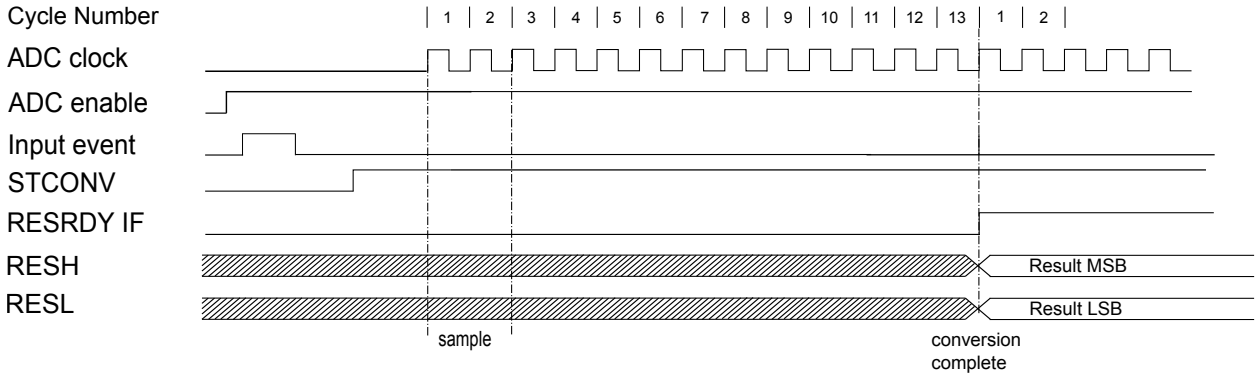
**Figure 30-9. ADC timing Diagram - Single conversion with delays**




In Free Running mode, a new conversion will be started immediately after the conversion completes, while the STCONV bit is one. The sampling rate $R_S$ in free running mode is calculated by

$$R_S = \frac{f_{\text{CLK_ADC}}}{(13 + \text{SAMPDLY} + \text{SAMPLEN})}$$

**Figure 30-10. ADC Timing Diagram - Free Running Conversion**



#### 30.3.2.4 Changing Channel or Reference Selection

The MUXPOS bits the ADC.MUXPOS register and the REFSEL bits in the ADC.CTRLC register are buffered through a temporary register to which the CPU has random access. This ensures that the channel and reference selections only take place at a safe point during the conversion. The channel and reference selections are continuously updated until a conversion is started.

Once the conversion starts, the channel and reference selections are locked to ensure sufficient sampling time for the ADC. Continuous updating resumes in the last CLK_ADC clock cycle before the conversion completes (RESRDY in ADC.INTFLAGS is set). The conversion starts on the following rising CLK_ADC clock edge after the STCONV bit is written to '1'.

**ADC Input Channels**

When changing channel selection, the user should observe the following guidelines to ensure that the correct channel is selected:

In Single Conversion mode: The channel should be selected before starting the conversion. The channel selection may be changed one ADC clock cycle after writing '1' to the STCONV bit.

In Free Running mode: The channel should be select before starting the first conversion. The channel selection may be changed one ADC clock cycle after writing '1' to the STCONV bit. 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.

After switching the input channel the ADC requires a settling time, please refer to the electrical characteristics section for details.

**ADC Voltage Reference**

The reference voltage for the ADC ($V_{REF}$) controls the conversion range of the ADC. Input voltages that exceed the selected $V_{REF}$ will be converted to the maximum result value of the ADC, for an ideal 10-bit ADC this is 0x3FF. $V_{REF}$ can be selected by writing the Reference Selection bits (REFSEL) in the Control C register (ADC.CTRLC) as either $V_{DD}$ or an internal reference. $V_{DD}$ is connected to the ADC through a passive switch.

When using ADCnREFSEL[0:2] in the corresponding VREF.CTRLn register should be configured to the selection closest, but above applied reference voltage. For external references higher than 4.3V, use ADCnREFSEL[0:2] = 0x3

The internal reference is generated from an internal bandgap reference through an internal amplifier, controlled by the Voltage Reference (VREF) peripheral.

**Related Links**

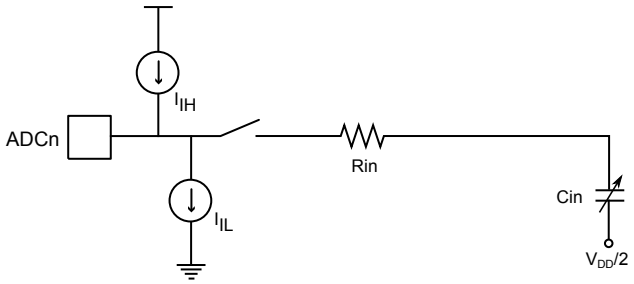
VREF - Voltage Reference

**Analog Input Circuitry**

The analog input circuitry is illustrated in the Figure below. An analog source applied to ADCn is subjected to the pin capacitance and input leakage of that pin (represented by $I_H$ and $I_L$), regardless of whether that channel is selected as input for the ADC. When the channel is selected, the source must drive the S/H capacitor through the series resistance (combined resistance in the input path).

The ADC is optimized for analog signals with an output impedance of approximately 10kΩ or less. If such a source is used, the sampling time will be negligible. If a source with higher impedance is used, the sampling time will depend on how long the source needs to charge the S/H capacitor, which can vary substantially.

**Figure 30-11. Analog Input Schematic**




#### 30.3.2.5 ADC Conversion Result

After the conversion is complete (RESRDY is set), the conversion result RES is available in the ADC Result Register (ADC.RES). The result for a 10-bit conversion is given as:

$$\mathrm{RES} = \frac{1023 \times V_{\mathrm{IN}}}{V_{\mathrm{REF}}}$$

where $V_{IN}$ is the voltage on the selected input pin and $V_{REF}$ the selected voltage reference (see description for REFSEL in ADC.CTRLC and ADC.MUXPOS).

#### 30.3.2.6 Temperature Measurement

The temperature measurement is based on an on-chip temperature sensor. For a temperature measurement, follow these steps:

1. Configure the internal voltage reference to 1.1V: in VREF.CTRLA, write ADC0REFSEL=0x1.
2. Select the internal voltage reference by writing the REFSEL bits in ADC.CTRLC to 0x0.
3. Select the ADC Temperature sensor channel by writing the MUXPOS bits in the MUXPOS register (ADC.MUXPOS) to 0x1E. This enables the temperature sensor.
4. In ADC.CTRLD Select $\mathrm{INITDLY} \geq 32\mu\mathrm{s} \times f_{\mathrm{CLK_ADC}}$
5. In ADC.SAMPCTRL Select $\mathrm{SAMPLEN} \geq 32\mu\mathrm{s} \times f_{\mathrm{CLK_ADC}}$
6. In ADC.CTRLC Select $\mathrm{SAMPCAP} = 5\mathrm{pF}$
7. Acquire the temperature sensor output voltage by starting a conversion.
8. Process the measurement result as described below.

The measured voltage has a linear relationship to the temperature. Due to process variations, the temperature sensor output voltage varies between individual devices at the same temperature. The individual compensation factors are determined during the production test and saved in the Signature Row:

- SIGROW.TEMPSENSE0 is an gain/slope correction
- SIGROW.TEMPSENSE1 is a offset correction

In order to achieve accurate results, the result of the temperature sensor measurement must be processed in the application software using factory calibration values. The temperature (in Kelvin) is calculated by this rule:

```
Temp = (((RESH << 8) | RESL) - TEMPSENSE1) * TEMPSENSE0) >> 8
```

RESH and RESL are the high and low byte of the Result register (ADC.RES), and TEMPSENSEn are the respective values from the Signature row.

It is recommended to follow these steps in user code:

```
int8_t sigrow_offset = SIGROW.TEMPSENSE1;  // Read signed value from signature row
uint8_t sigrow_gain = SIGROW.TEMPSENSE0;    // Read unsigned value from signature row
uint16_t adc_reading = 0;   // ADC conversion result with 1.1 V internal reference
```

```
uint32_t temp = adc_reading - sigrow_offset;
temp *= sigrow_gain;  // Result might overflow 16 bit variable (10bit+8bit)
temp += 0x80;              // Add 1/2 to get correct rounding on division below
temp >>= 8;                // Divide result to get Kelvin
uint16_t temperature_in_K = temp;
```

**Related Links**

TEMPSENSEn

#### 30.3.2.7 Window Comparator Mode

The ADC can raise the WCOMP flag in the Interrupt and Flag register (ADC.INTFLAG) and request an interrupt (WCOMP) when the result of a conversion is above and/or below certain thresholds. The available modes are:

- The result is under a threshold
- The result is over a threshold
- The result is inside a window (above a lower threshold, but below the upper one)
- The result is outside a window (either under the lower or above the upper threshold)

The thresholds are defined by writing to the Window Comparator Threshold registers (ADC.WINLT and ADC.WINHT). Writing to the Window Comparator Mode bit field (WINCM) in the Control E register (ADC.CTRLE) selects the conditions when the flag is raised and/or the interrupt is requested.

Assuming the ADC is already configured to run, follow these steps to use the Window Comparator Mode:

1. Choose which Window Comparator to use (see WINCM description in ADC.CTRLE), and set the required threshold(s) by writing to ADC.WINLT and/or ADC.WINHT.
2. Optional: enable the interrupt request by writing a '1' to the Window Comparator Interrupt Enable bit (WCOMP) in the Interrupt Control register (ADC.INTCTRL)
3. Enable the Window Comparator and select a mode by writing a non-zero value to the WINCM bit field in ADC.CTRLE.

When accumulating multiple samples, the comparison between the result and the threshold will happen after the last sample was acquired. Consequently, the flag is only raised once, after taking the last sample of the accumulation.

### 30.3.3 Events

An ADC conversion can be triggered automatically by an event input if the Start Event Input bit (STARTEI) in the Event Control register (ADC.EVCTRL) is written to '1'.

See also the description of the Asynchronous User Channel n Input Selection in the Event System (EVSYS.ASYNCUSERn).

**Related Links**

ASYNCUSER0, ASYNCUSER1, ASYNCUSER2, ASYNCUSER3, ASYNCUSER4, ASYNCUSER5, ASYNCUSER6, ASYNCUSER7, ASYNCUSER8, ASYNCUSER9, ASYNCUSER10

### 30.3.4 Interrupts

**Table 30-2. Available Interrupt Vectors and Sources**

| Offset | Name | Vector Description | Conditions |
|---|---|---|---|
| 0x00 | RESRDY | Result Ready interrupt | The conversion result is available in the Result register (ADC.RES). |
| 0x02 | WCOMP | Window Comparator interrupt | As defined by WINCM in ADC.CTRLE. |

When an interrupt condition occurs, the corresponding Interrupt Flag is set in the Interrupt Flags register of the peripheral (*peripheral*.INTFLAGS).

An interrupt source is enabled or disabled by writing to the corresponding enable bit in the peripheral's Interrupt Control register (*peripheral*.INTCTRL).

An interrupt request is generated when the corresponding interrupt source is enabled and the Interrupt Flag is set. The interrupt request remains active until the Interrupt Flag is cleared. See the peripheral's INTFLAGS register for details on how to clear Interrupt Flags.

### 30.3.5 Sleep Mode Operation

The ADC is by default disabled In *Standby sleep mode*.

The ADC can stay fully operational in Standby sleep mode if the Run in Standby bit (RUNSTDBY) in the Control A register (ADC.CTRLA) is written to '1'.

When the device is entering Standby sleep mode when RUNSTDBY is one, the ADC will stay active, hence any ongoing conversions will be completed, and interrupts will be executed as configured.

In Standby sleep mode an ADC conversion must be triggered via the Event System (EVSYS), or the ADC must be in free-running mode with the first conversion triggered by software before entering sleep. The peripheral clock is requested if needed and is turned off after the conversion is completed.

When an input Event trigger occurs, the positive edge will be detected, the Start Conversion bit (STCONV) in the Command register (ADC.COMMAND) set, and the conversion will start. When the conversion is completed, the Result Ready Flag (RESRDY) in the Interrupt Flags register (ADC.INTFLAGS) is set and the STCONV bit in ADC.COMMAND is cleared.

The reference source and supply infrastructure need time to stabilize when activated in Standby sleep mode. Configure a delay for the start of the first conversion by writing a non-zero value to the Initial Delay bits (INITDLY) in the Control D register (ADC.CTRLD).

In *Power Down sleep mode*, no conversions are possible. Any ongoing conversions are halted and will be resumed when going out of sleep. At end of conversion, the Result Ready Flag (RESRDY) will be set, but the content of the result registers (ADC.RES) is invalid since the ADC was halted in the middle of a conversion.

**Related Links**

SLPCTRL - Sleep Controller

### 30.3.6 Synchronization

Not applicable.

### 30.3.7 Configuration Change Protection

Not applicable.

## 30.4 Register Summary - ADC

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | RUNSTBY | | | | | RESSEL | FREERUN | ENABLE |
| 0x01 | CTRLB | 7:0 | | | | | | SAMPNUM[2:0] | | |
| 0x02 | CTRLC | 7:0 | | SAMPCAP | REFSEL[1:0] | | | PRESC[2:0] | | |
| 0x03 | CTRLD | 7:0 | INITDLY[2:0] | | | ASDV | SAMPDLY[3:0] | | | |
| 0x04 | CTRLE | 7:0 | | | | | | WINCM[2:0] | | |
| 0x05 | SAMPCTRL | 7:0 | | | | SAMPLEN[4:0] | | | | |
| 0x06 | MUXPOS | 7:0 | | | | MUXPOS[4:0] | | | | |
| 0x07 | Reserved | | | | | | | | | |
| 0x08 | COMMAND | 7:0 | | | | | | | | STCONV |
| 0x09 | EVCTRL | 7:0 | | | | | | | | STARTEI |
| 0x0A | INTCTRL | 7:0 | | | | | | | WCOMP | RESRDY |
| 0x0B | INTFLAGS | 7:0 | | | | | | | WCOMP | RESRDY |
| 0x0C | DBGCTRL | 7:0 | | | | | | | | DBGRUN |
| 0x0D | TEMP | 7:0 | TEMP[7:0] | | | | | | | |
| 0x0E ... 0x0F | Reserved | | | | | | | | | |
| 0x10 | RES | 7:0 | RES[7:0] | | | | | | | |
| 0x11 | | 15:8 | RES[15:8] | | | | | | | |
| 0x12 | WINLT | 7:0 | WINLT[7:0] | | | | | | | |
| 0x13 | | 15:8 | WINLT[15:8] | | | | | | | |
| 0x14 | WINHT | 7:0 | WINHT[7:0] | | | | | | | |
| 0x15 | | 15:8 | WINHT[15:8] | | | | | | | |

## 30.5 Register Description

### 30.5.1 Control A

**Name:** CTRLA
**Offset:** 0x00
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RUNSTBY | | | | | RESSEL | FREERUN | ENABLE |
| Access | R/W | R | R | R | R | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Bit 7 – RUNSTBY: Run in Standby**
This bit determines whether the ADC needs to run when the chip is in standby sleep mode.

**Bit 2 – RESSEL: Resolution Selection**
This bit selects the ADC resolution.

| Value | Description |
|---|---|
| 0 | Full 10-bit resolution. The 10-bit ADC results are accumulated or stored to the ADC Result register (ADC.RES). |
| 1 | 8-bit resolution. The conversion results are truncated to 8 bits (MSBs) before they are accumulated or stored to the ADC Result register (ADC.RES). The two least significant bits are discarded. |

**Bit 1 – FREERUN: Free Running**
Writing a '1' to this bit will enable Free Running mode for the data acquisition. The first conversion is started by writing COMMAND.STCONV bit high. In Free running mode, a new conversion cycle is started immediately after or as soon as the previous conversion cycle has completed. This is signaled by INTFLAGS.RESRDY.

**Bit 0 – ENABLE: ADC Enable**

| Value | Description |
|---|---|
| 0 | ADC is disabled. |
| 1 | ADC is enabled. |

### 30.5.2 Control B

**Name:** CTRLB
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

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

**Bits 2:0 – SAMPNUM[2:0]: Sample Accumulation Number Select**
These bits select how many consecutive ADC sampling results are accumulated automatically. When this bit is written to a value greater than 0x0, the according number of consecutive ADC sampling results are accumulated into the ADC Result register (RES) in one complete conversion.

| Value | Name | Description |
|---|---|---|
| 0x0 | NONE | No accumulation. |
| 0x1 | ACC2 | 2 results accumulated. |
| 0x2 | ACC4 | 4 results accumulated. |
| 0x3 | ACC8 | 8 results accumulated. |
| 0x4 | ACC16 | 16 results accumulated. |
| 0x5 | ACC32 | 32 results accumulated. |
| 0x6 | ACC64 | 64 results accumulated. |
| 0x7 | - | Reserved. |

### 30.5.3 Control C

**Name:** CTRLC
**Offset:** 0x02
**Reset:** 0x00
**Property:** -

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

**Bit 6 – SAMPCAP: Sample Capacitance Selection**
This bit selects the sample capacitance, and hence, the input impedance. The best value is dependent on the reference voltage and the application's electrical properties.

| Value | Description |
|---|---|
| 0 | Recommended for reference voltage values below 1V. |
| 1 | Reduced size of sampling capacitance. Recommended for higher reference voltages. |

**Bits 5:4 – REFSEL[1:0]: Reference Selection**
These bits selects the voltage reference for the ADC.

| Value | Name | Description |
|---|---|---|
| 0x0 | INTERNAL | Internal reference |
| 0x1 | VDD | $V_{DD}$ |
| Other | - | Reserved. |

**Bits 2:0 – PRESC[2:0]: Prescaler**
These bits defines the division factor from peripheral clock (CLK_PER) to the ADC clock (CLK_ADC).

| Value | Name | Description |
|---|---|---|
| 0x0 | DIV2 | CLK_PER divided by 2 |
| 0x1 | DIV4 | CLK_PER divided by 4 |
| 0x2 | DIV8 | CLK_PER divided by 8 |
| 0x3 | DIV16 | CLK_PER divided by 16 |
| 0x4 | DIV32 | CLK_PER divided by 32 |
| 0x5 | DIV64 | CLK_PER divided by 64 |
| 0x6 | DIV128 | CLK_PER divided by 128 |
| 0x7 | DIV256 | CLK_PER divided by 256 |

### 30.5.4 Control D

**Name:** CTRLD
**Offset:** 0x03
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | INITDLY[2:0] | | | ASDV | SAMPDLY[3:0] | | | |
| 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 7:5 – INITDLY[2:0]: Initialization Delay**
These bits defines the initialization/startup delay before the first sample when enabling the ADC or changing to internal reference voltage. Setting this delay will ensure that the reference, muxes, etc are ready before starting the first conversion. The initialization delay will also take place when waking up from deep sleep to do a measurement.

The delay is expressed as a number of CLK_ADC cycles.

| Value | Name | Description |
|---|---|---|
| 0x0 | DLY0 | Delay 0 CLK_ADC cycles. |
| 0x1 | DLY16 | Delay 16 CLK_ADC cycles. |
| 0x2 | DLY32 | Delay 32 CLK_ADC cycles. |
| 0x3 | DLY64 | Delay 64 CLK_ADC cycles. |
| 0x4 | DLY128 | Delay 128 CLK_ADC cycles. |
| 0x5 | DLY256 | Delay 256 CLK_ADC cycles. |
| Other | - | Reserved |

**Bit 4 – ASDV: Automatic Sampling Delay Variation**
Writing this bit to '1' enables automatic sampling delay variation between ADC conversions. The purpose of varying sampling instant is to randomize the sampling instant and thus avoid standing frequency components in frequency spectrum. The value of the SAMPDLY bits is automatically incremented by one after each sample.

When the Automatic Sampling Delay Variation is enabled and the SAMPDLY value reaches 0xF, it wraps around to 0x0.

| Value | Name | Description |
|---|---|---|
| 0 | ASVOFF | The Automatic Sampling Delay Variation is disabled. |
| 1 | ASVON | The Automatic Sampling Delay Variation is enabled. |

**Bits 3:0 – SAMPDLY[3:0]: Sampling Delay Selection**
These bits define the delay between consecutive ADC samples. The programmable Sampling Delay allows modifying the sampling frequency during hardware accumulation, to suppress periodic noise sources that may otherwise disturb the sampling. The SAMPDLY field can be also modified automatically from sampling cycle to another, by setting the ASDV bit. The delay is expressed as CLK_ADC cycles and is given directly by the bitfield setting. The sampling cap is kept open during the delay.

### 30.5.5 Control E

**Name:** CTRLE
**Offset:** 0x4
**Reset:** 0x00
**Property:** -

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

**Bits 2:0 – WINCM[2:0]: Window Comparator Mode**
This field enable and define when the interrupt flag is set in Window Comparator mode. RESULT is the 16-bit accumulator result. WINLT and WINHT are 16-bit lower threshold value and 16-bit higher threshold value, respectively.

| Value | Name | Description |
|---|---|---|
| 0x0 | NONE | No Window Comparison (default) |
| 0x1 | BELOW | *RESULT < WINLT* |
| 0x2 | ABOVE | *RESULT > WINHT* |
| 0x3 | INSIDE | *WINLT < RESULT < WINHT* |
| 0x4 | OUTSIDE | *RESULT < WINLT or RESULT >WINHT)* |
| Other | - | Reserved |

### 30.5.6 Sample Control

**Name:** SAMPCTRL
**Offset:** 0x5
**Reset:** 0x00
**Property:** -

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

**Bits 4:0 – SAMPLEN[4:0]: Sample Length**
These bits extend the ADC sampling length in number of CLK_ADC cycles. By default the sampling time is two CLK_ADC cycles. Increasing the sampling length allows sampling sources with higher impedance. The total conversion time increased with the selected sampling length.

### 30.5.7 MUXPOS

**Name:** MUXPOS
**Offset:** 0x06
**Reset:** 0x00
**Property:** -

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

**Bits 4:0 – MUXPOS[4:0]: MUXPOS**
This bit field selects which single ended analog input is connected to the ADC. If these bits are changed during a conversion, the change will not take effect until this conversion is complete.

| Value | Name | Description |
|---|---|---|
| 0x00 | AIN0 | ADC input pin 0 |
| 0x01 | AIN1 | ADC input pin 1 |
| 0x02 | AIN2 | ADC input pin 2 |
| 0x03 | AIN3 | ADC input pin 3 |
| 0x04 | AIN4 | ADC input pin 4 |
| 0x05 | AIN5 | ADC input pin 5 |
| 0x06 | AIN6 | ADC input pin 6 |
| 0x07 | AIN7 | ADC input pin 7 |
| 0x08 | AIN8 | ADC input pin 8 |
| 0x09 | AIN9 | ADC input pin 9 |
| 0x0A | AIN10 | ADC input pin 10 |
| 0x0B | AIN11 | ADC input pin 11 |
| 0x1C | DAC0 | DAC0 |
| 0x1D | INTREF | Internal reference (from VREF peripheral) |
| 0x1E | TEMPSENSE | Temperature sensor |
| 0x1F | GND | 0V (GND) |
| Other | - | Reserved |

### 30.5.8 Command

**Name:** COMMAND
**Offset:** 0x08
**Reset:** 0x00
**Property:** -

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

**Bit 0 – STCONV: Start Conversion**
Writing a '1' to this bit will start a single measurement. If in free running mode this will start the first conversion. STCONV will read as '1' as long as a conversion is in progress. When the conversion is complete, this bit is automatically cleared.

Writing the bit to '0' during an ongoing conversion will stop the conversion.

### 30.5.9 Event Control

**Name:** EVCTRL
**Offset:** 0x09
**Reset:** 0x00
**Property:** -

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

**Bit 0 – STARTEI: Start Event Input**
This bit enables event input as source for conversion start.

### 30.5.10 Interrupt Control

**Name:** INTCTRL
**Offset:** 0x0A
**Reset:** 0x00
**Property:** -

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

**Bit 1 – WCOMP: Window Comparator Interrupt Enable**
Writing a '1' to this bit enables window comparator interrupt.

**Bit 0 – RESRDY: Result Ready Interrupt Enable**
Writing a '1' to this bit enables result ready interrupt.

### 30.5.11 Interrupt Flags

**Name:** INTFLAGS
**Offset:** 0x0B
**Reset:** 0x00
**Property:** -

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

**Bit 1 – WCOMP: Window Comparator Interrupt Flag**
This window comparator flag is set when the measurement is complete and if the result match the selected window comparator mode defined by WINCM (ADCn.CTRLE). The comparison is done at end of

conversion. The flag is cleared by either writing a '1' to the bit position or by reading the result register (ADCn.RES). Writing a '0' to this bit has no effect.

**Bit 0 – RESRDY: Result Ready Interrupt Flag**
The result ready interrupt flag is set when a measurement is complete and a new result is ready. The flag is cleared by either writing a '1' to the bit location or by reading the result register (ADCn.RES). Writing a '0' to this bit has no effect.

### 30.5.12 Debug Run

**Name:** DBGCTRL
**Offset:** 0x0C
**Reset:** 0x00
**Property:** -

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

**Bit 0 – DBGRUN: Debug Run**

| Value | Description |
|---|---|
| 0 | The peripheral is halted in break debug mode and ignores events. |
| 1 | The peripheral will continue to run in break debug mode when the CPU is halted. |

### 30.5.13 Temporary

The Temporary register is used by the CPU for single-cycle, 16-bit access to the 16-bit registers of this peripheral. It can also be read and written by software. See also Accessing 16-bit Registers. There is one common Temporary register for all the 16-bit registers of this peripheral.

**Name:** TEMP
**Offset:** 0x0D
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TEMP[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 – TEMP[7:0]: Temporary**
Temporary register for read/write operations in 16-bit registers.

### 30.5.14 Result

The RESL and RESH register pair represents the 16-bit value, RES. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

If the analog input is higher than the reference level of the ADC, the 10 bit ADC result will be equal the max value of 0x3FF. Likewise, if the input is below 0V, the ADC result will be 0x000. As the ADC cannot produce a result above 0x3FF values, the accumulated value will never exceed 0xFFC0 even after maximum allowed 64 accumulations.

**Name:** RES
**Offset:** 0x10
**Reset:** 0x00
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | RES[15:8] | | | | | | | |
| Access | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

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

**Bits 15:8 – RES[15:8]: Result high byte**
These bits constitute the MSB of RES register, where the msb is RES[15]. The ADC itself has 10-bit output, ADC[9:0], where the msb is ADC[9]. The data format in ADC and Digital Accumulation is 1's complement, where 0x0000 represents the zero and 0xFFFF represent the largest number (full scale).

**Bits 7:0 – RES[7:0]: Result low byte**
These bits constitute the LSB of ADC/Accumulator Result, (RES) register. The data format in ADC and Digital Accumulation is 1's complement, where 0x0000 represents the zero and 0xFFFF represent the largest number (full scale).

### 30.5.15 Window Comparator Low Threshold

This register is the 16-bit Low Threshold for the digital comparator monitoring the RES register. The ADC itself has 10-bit output, RES[9:0], where the msb is RES[9]. The data format in ADC and Digital Accumulation is one's complement, where 0x0000 represents the zero and 0xFFFF represent the largest number (full scale).

The WINLTH and WINLTL register pair represent the 16-bit value, WINLT. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

When accumulating samples, the window comparator thresholds are applied on the accumulated value and not on each sample.

**Name:** WINLT
**Offset:** 0x12
**Reset:** 0x00
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | WINLT[15:8] | | | | | | | |
| 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 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | WINLT[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 15:8 – WINLT[15:8]: Window Comparator Low Threshold high byte**
These bits hold the MSB of the 16-bit register.

**Bits 7:0 – WINLT[7:0]: Window Comparator Low Threshold low byte**
These bits hold the LSB of the 16-bit register.

### 30.5.16 Window Comparator High Threshold

This register is the 16-bit High Threshold for the digital comparator monitoring the RES register. The ADC itself has 10-bit output, RES[9:0], where the msb is RES[9]. The data format in ADC and Digital Accumulation is one's complement, where 0x0000 represents the zero and 0xFFFF represent the largest number (full scale).

The WINHTH and WINHTL register pair represent the 16-bit value, WINHT. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to Accessing 16-bit Registers.

**Name:** WINHT
**Offset:** 0x14
**Reset:** 0x00
**Property:** -

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | WINHT[15:8] | | | | | | | |
| 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 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | WINHT[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 15:8 – WINHT[15:8]: Window Comparator High Threshold high byte**
These bits hold the MSB of the 16-bit register.

**Bits 7:0 – WINHT[7:0]: Window Comparator High Threshold low byte**
These bits hold the LSB of the 16-bit register.

# 31. DAC - Digital to Analog Converter

## 31.1 Features

- 8-bit resolution
- Up to 350ksps conversion rate
- High drive capabilities
- Functioning as input to Analog Comparator (AC) or ADC
- One instances DAC0

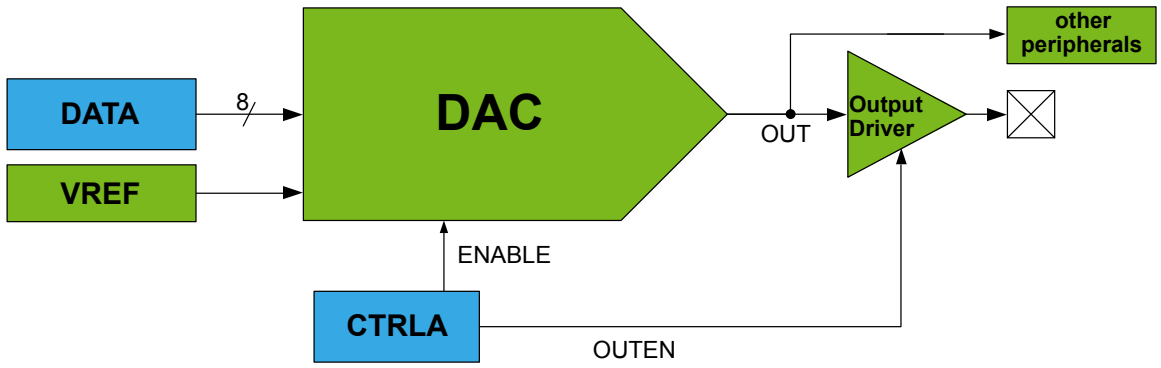
## 31.2 Overview

The Digital-to-Analog Converter (DAC) converts a digital value written to the Data register (DAC.DATA) to an analogue voltage. The conversion range is between GND and the selected reference voltage.

The DAC features an 8-bit Resistor String type DAC, capable of converting 350,000 samples per second (350ksps). The DAC uses the internal Voltage Reference (VREF) as upper limit for conversion. The DAC has one continuous time output with high drive capabilities, which is able to drive 5kΩ and/or 30pF load. The DAC conversion can be started from the application by writing to the data conversion registers.

### 31.2.1 Block Diagram

**Figure 31-1. DAC Block Diagram**




### 31.2.2 Signal Description

| Signal | Description | Type |
|---|---|---|
| OUT | DAC output | Analog |

**Related Links**

I/O Multiplexing and Considerations

### 31.2.3 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 31-1. DAC System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | Yes | PORT |
| Interrupts | No | - |
| Events | No | - |
| Debug | Yes | UPDI |

**Related Links**

Clocks

I/O Lines and Connections

Debug Operation

#### 31.2.3.1 Clocks

This peripheral depends on the peripheral clock.

**Related Links**

CLKCTRL - Clock Controller

#### 31.2.3.2 I/O Lines and Connections

Using the I/O lines of the peripheral requires configuration of the I/O pins.

**Table 31-2. I/O Lines**

| Instance | Signal | I/O Line | Peripheral Function |
|---|---|---|---|
| DAC | OUT | PA6 | A |

The DAC has one analog output pin (OUT) that must be configured before it can be used.

A DAC is also internally connected to the AC and to the ADC. To use this internal OUT as input, both output and input must be configured in their respective registers.

**Related Links**

PORT - I/O Pin Configuration

AC – Analog Comparator

ADC - Analog to Digital Converter

#### 31.2.3.3 Events

Not applicable.

#### 31.2.3.4 Interrupts

Not applicable.

#### 31.2.3.5 Debug Operation

This peripheral is unaffected by entering debug mode.

If the peripheral is configured to require periodical service by the CPU through interrupts or similar, improper operation or data loss may result during halted debugging.

**Related Links**

PORT - I/O Pin Configuration

## 31.3 Functional Description

### 31.3.1 Initialization

To operate the DAC, the following steps are required:

- Select the DAC reference voltage in the Voltage Reference (VREF) peripheral by writing the DAC and AC Reference Selection bits (DAC0REFSEL) in the Control A register of the Voltage Reference (VREF.CTRLA).
- The conversion range is between GND and the selected reference voltage.
- Configure the further usage of the DAC output:
  - Configure an internal peripheral (e.g. AC, ADC) to use the DAC output. See the according peripheral's documentation.
  - Enable the output to a pin by writing a '1' to the Output Enable bit (OUTEN) in the Control A register (DAC.CTRLA). This requires configuration of the Port peripheral.
- Write an initial digital value to the Data register (DAC.DATA).
- Enable the DAC by writing a '1' to the ENABLE bit in the Control A register (DAC.CTRLA).

**Related Links**

VREF - Voltage Reference

AC – Analog Comparator

ADC - Analog to Digital Converter

### 31.3.2 Operation

#### 31.3.2.1 Enabling, Disabling and Resetting

The DAC is enabled by writing a '1' to the ENABLE bit in the Control A register (DAC.CTRLA), and disabled by writing a '0' to this bit.

The OUT output to a pin is enabled by writing the Output Enable bit (OUTEN) in the CTRLA register (DAC.CTRLA).

#### 31.3.2.2 Starting a Conversion

When the DAC is enabled (ENABLE=1 in DAC.CTRLA), a conversion starts as soon as the Data register (DAC.DATA) is written.

When the DAC is disabled (ENABLE=0 in DAC.CTRLA), writing DAC.DATA does not trigger a conversion. Instead, the conversion starts on writing a '1' to ENABLE in DAC.CTRLA.

#### 31.3.2.3 DAC as Source For Internal Peripherals

The analog output of the DAC is internally connected to both the AC and the ADC, and is available to these other peripherals when the DAC is enabled (ENABLE=1 in DAC.CTRLA). When the DAC analog output is only being used internally, it is not necessary to enable the pin output driver (i.e. OUTEN=0 in DAC.CTRLA is acceptable).

**Related Links**

AC – Analog Comparator

ADC - Analog to Digital Converter

### 31.3.3 Sleep Mode Operation

If the Run in Standby bit (RUNSTDBY) in the Control A register (DAC.CTRLA) is written to '1' and CLK_PER is available, the DAC will continue to operate in Standby sleep mode. If RUNSTDBY bit is zero, the DAC will stop the conversion in Standby sleep mode.

If conversion is stopped in Standby sleep mode, the DAC and the output buffer are disabled to reduce power consumption. When the device is exiting Standby sleep mode, the DAC and the output buffer (if configured by OUTEN=1 in DAC.CTRLA) are enabled again. Therefore, a certain start-up time is required before a new conversion is initiated.

In Power Down sleep mode, the DAC and output buffer are disabled to reduce power consumption.

### 31.3.4 Configuration Change Protection

Not applicable.

## 31.4 Register Summary - DAC

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CTRLA | 7:0 | RUNSTDBY | OUTEN | | | | | | ENABLE |
| 0x01 | DATA | 7:0 | DATA[7:0] | | | | | | | |

## 31.5 Register Description

### 31.5.1 Control A

**Name:** CTRLA
**Offset:** 0x00
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RUNSTDBY | OUTEN | | | | | | ENABLE |
| Access | R/W | R/W | | | | | | R/W |
| Reset | 0 | 0 | | | | | | 0 |

**Bit 7 – RUNSTDBY: Run in Standby Mode**
If this bit is written to '1', the DAC or Output Buffer will not automatically be disabled when the device is entering Standby sleep mode.

**Bit 6 – OUTEN: Output Buffer Enable**
Writing a '1' to this bit enables the Output Buffer and sends the OUT signal to a pin.

**Bit 0 – ENABLE: DAC Enable**
Writing a '1' to this bit enables the DAC.

### 31.5.2 DATA

**Name:** DATA
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DATA[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 – DATA[7:0]: Data**
These bits contains the digital data which will be converted to an analog voltage.

# 32. UPDI - Unified Program and Debug Interface

## 32.1 Features

- Programming
  - External programming through UPDI 1-wire (1W) interface
    - Enable programming by 12V or fuse
    - Uses the RESET pin of the device for programming
    - No GPIO pins occupied during operation
    - Asynchronous Half-Duplex UART protocol towards the programmer
- Debugging
  - Memory mapped access to device address space (NVM, RAM, I/O)
  - No limitation on device clock frequency
  - Unlimited number of user program breakpoints
  - 2 Hardware Breakpoints
  - Run-time readout of program counter (PC), Stack Pointer (SP) and CPU Status register (CPU_SREG) for code profiling
  - Program Flow Control
    - Go, Stop, Reset, Step Into
  - Non-intrusive run-time chip monitoring without accessing system registers
    - Monitor CRC status and sleep status
- Unified Programming and Debug Interface (UPDI)
  - Built in error detection with error signature readout
  - Frequency Measurement of internal oscillators using the Event System

## 32.2 Overview

The Unified Program and Debug Interface (UPDI) is an proprietary interface for external programming and on-chip debugging of a device.

The UPDI supports programming of nonvolatile memory (NVM) space; FLASH, EEPROM, fuses, lockbits and the user row. In addition the UPDI can access the entire I/O and data space of the device. See the NVM Controller documentation for programming via the NVM controller and executing NVM controller commands.

Programming and debugging is done through the UPDI Physical interface (UPDI PHY), which is a 1-wire UART based half-duplex interface using the RESET pin for data reception and transmission. Clocking of UPDI PHY is done by an internal oscillator. Enabling of the 1-wire interface, by disabling the reset functionality, is either done by 12V programming or by fusing the RESET pin to UPDI by setting the RESET Pin Configuration (RSTPINCFG) bits in FUSE.SYSCFG0. The UPDI Access layer grants access to the Bus Matrix, with memory mapped access to system blocks such as Memories, NVM, and peripherals.

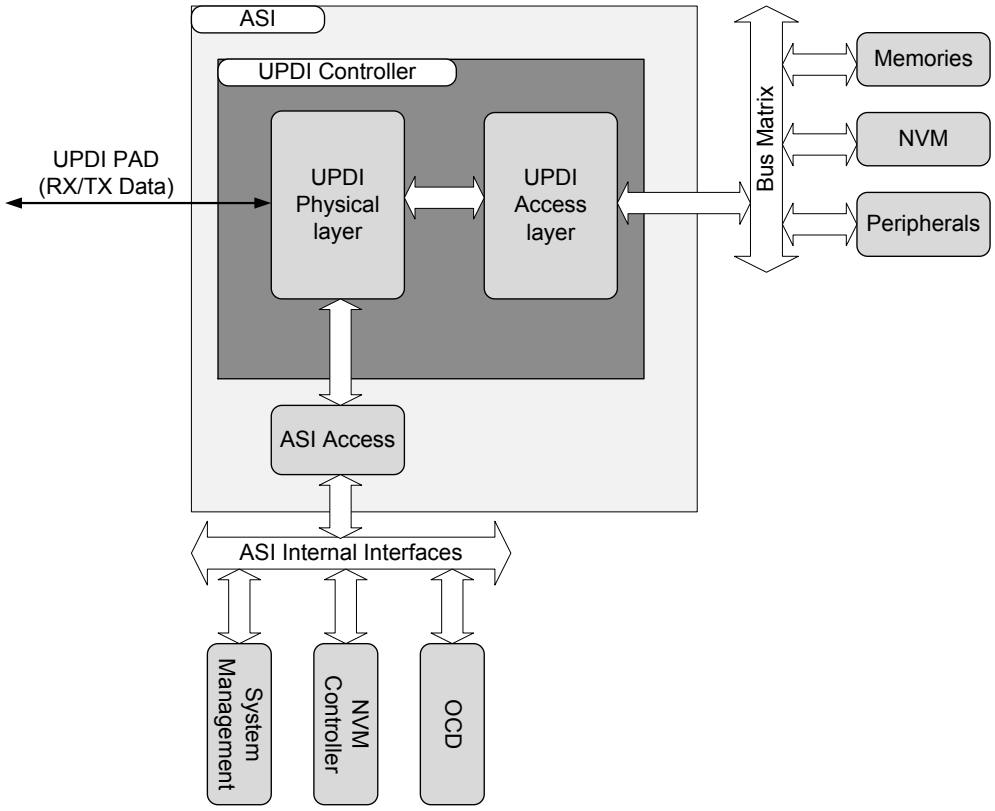
The Asynchronous System Interface (ASI) provides direct interface access to On-Chip Debugging (OCD), NVM and System Management features. This gives the debugger direct access to system information, without requesting bus access.

**Related Links**

NVMCTRL - Non Volatile Memory Controller
Enabling of KEY Protected Interfaces

### 32.2.1 Block Diagram

**Figure 32-1. UPDI Block Diagram**




### 32.2.2 System Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

**Table 32-1. UPDI System Dependencies**

| Dependency | Applicable | Peripheral |
|---|---|---|
| Clocks | Yes | CLKCTRL |
| I/O Lines and Connections | Yes | PORT |
| Interrupts | No | - |
| Events | Yes | EVSYS |
| Debug | Yes | UPDI |

**Related Links**

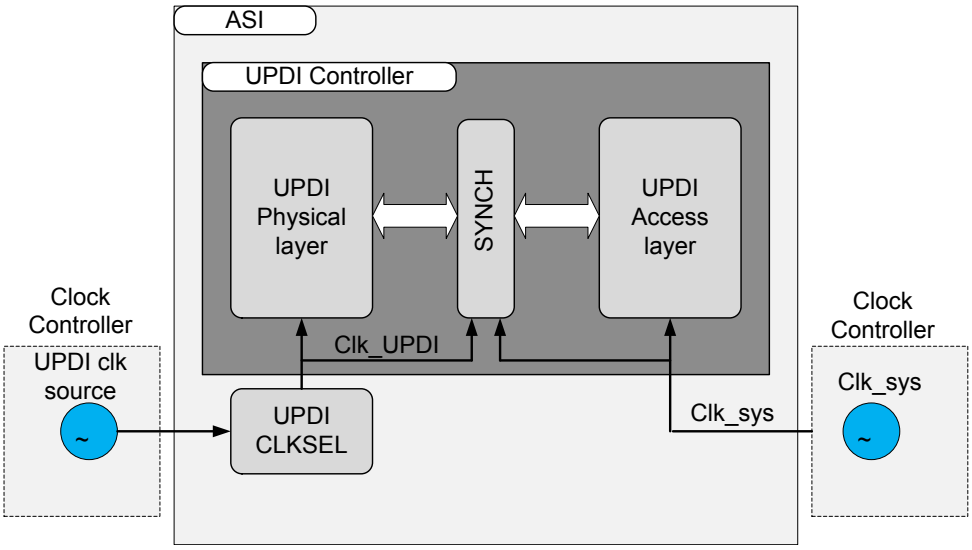
I/O Lines and Connections
Power Management

#### 32.2.2.1 Clocks

The UPDI Physical (UPDI PHY) layer and UPDI Access (UPDI ACC) layer can operate on different clock domains. The UPDI PHY layer clock is derived from an internal oscillator, and the UPDI ACC layer clock

is the same as the system clock. There is a synchronization boundary between the UPDI PHY layer and UPDI ACC layer, which ensures correct operation between the clock domains. The UPDI clock output frequency is selected through the ASI, and the default UPDI clock start-up frequency is 4MHz after enabling the UPDI. The UPDI clock frequency is changed by writing the UPDICLKSEL bits in ASI_CTRLA register.

**Figure 32-2. UPDI clock domains**




**Related Links**

CLKCTRL - Clock Controller

#### 32.2.2.2 I/O Lines and Connections

To operate the UPDI, the $\overline{\text{RESET}}$ pin must be set to UPDI Mode. This is not done through the PORT I/O Pin Configuration as for regular I/O Pins, but through setting the $\overline{\text{RESET}}$ pin Configuration (RSTPINCFG) bits in FUSE.SYSCFG0 as described in UPDI Enable with Fuse Override of RESET pin, or by following the UPDI 12V enable sequence from UPDI Enable with 12V Override of RESET pin. Pull enable, input enable and output enable settings are automatically controlled by the UPDI when active.

#### 32.2.2.3 Events

The events of this peripheral are connected to the Event System.

**Related Links**

EVSYS - Event System

#### 32.2.2.4 Power Management

The UPDI physical layer continues to operate in any sleep mode and is always accessible for a connected debugger, but read/write access to the system bus is restricted in sleep modes where the CPU clock is switched off. The UPDI can be enabled at any time, independent of the system sleep state. See Sleep Mode Operation for details on UPDI operation duing sleep modes.

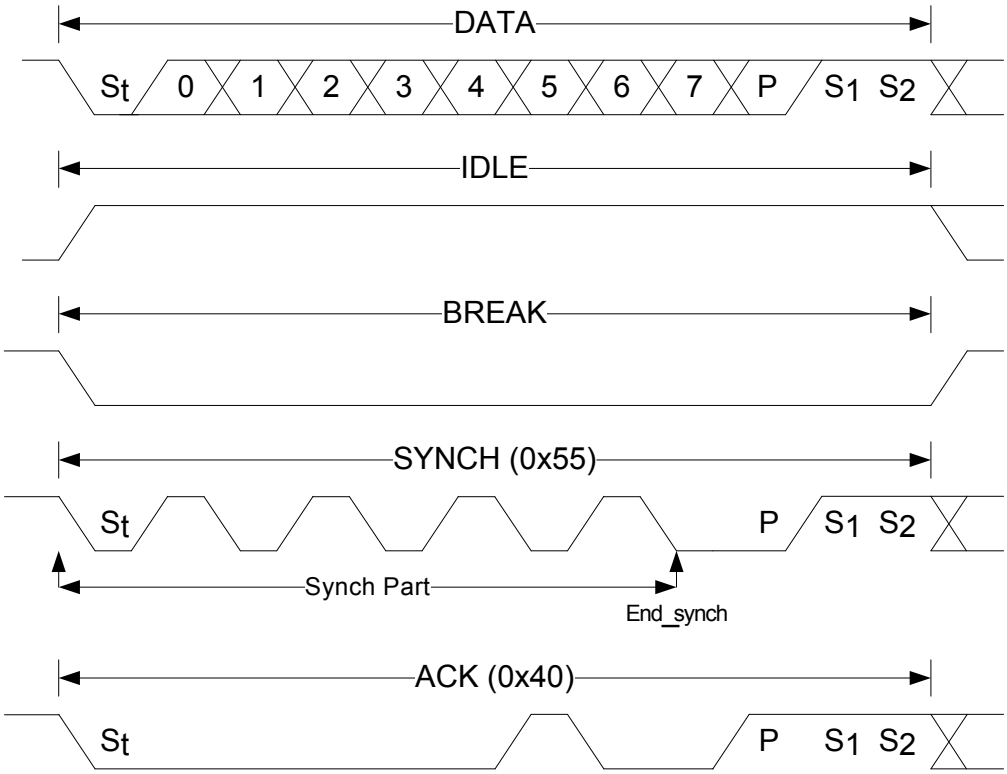
## 32.3 Functional Description

### 32.3.1 Principle of Operation

Communication through the UPDI is based on standard UART communication, using a fixed frame format, and automatic baud rate detection for clock and data recovery. In addition to the data frame, there

are several control frames which are important to the communication. The supported frame formats are presented in Figure 32-3.

**Figure 32-3. Supported UPDI Frame formats**




**Data Frame** Consist of 1 start bit (always low), 8 data bits, 1 parity bit (even parity) and two stop bits (always high). If the start bit, parity bit or stop bits have an incorrect value, an error will be detected and signalized by the UPDI. The parity bit check in the UPDI can be disabled by writing the PARD bit in UPDI.CTRLA, in which case the parity generation from the debugger can be ignored.

**IDLE Frame** Special frame which consist of 12 high bits. This is the same as keeping the transmission line in an IDLE state.

**BREAK** Special frame which consist of 12 low bits. The BREAK frame is used to reset the UPDI back to its default state, and is typically used for error recovery.

**SYNCH** The SYNCH frame (0x55) is used by the baud rate generator to set the baud rate for the coming transmission. A SYNCH character is always expected by the UPDI in front of every new instruction, and after a successful BREAK has been transmitted.

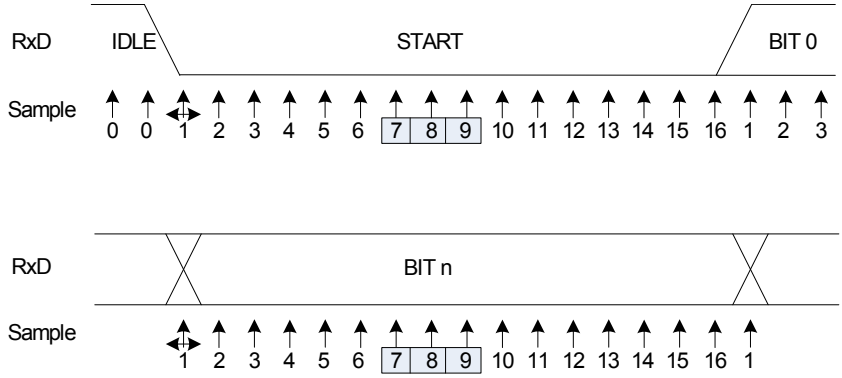
**ACK** The Acknowledge (ACK) character is transmitted from the UPDI whenever a ST or STS instruction has successfully crossed the synchronization boundary, and have gained bus access. When an ACK is received by the debugger, the next transmission can start.

#### 32.3.1.1 UPDI UART

All transmission and reception of serial data on the UPDI is achieved using the UPDI frames presented in Figure 32-3. Communication is initiated from the master (debugger) side, and every transmission must start with a SYNCH character upon which the UPDI can recover the transmission baud rate, and store this setting for the coming data. The baud rate set by the SYNCH character will be used for both reception and transmission for the instruction byte received after the SYNCH. See UPDI Instruction Set for details on when the next SYNCH character is expected in the instruction stream.

There is no writable baud rate register in the UPDI, so the baud rate sampled from the SYNCH character is used for data recovery by sampling the start bit, and do a majority voting on the middle samples. This process is repeated for all bits in the frame, including the parity bit and two stop bits. The baud generator uses 16 samples, and the majority voting is done on sample 7,8 and 9.

**Figure 32-4. UPDI UART Start Bit and Data/Parity/Stop bit sampling**




The transmission Baud Rate must be set up in relation to the selected UPDI clock, which can be adjusted by UPDICLKSEL in UPDI.ASI_CTRLA. See Table 32-2 for recommended maximum and minimum baud rate settings.

**Table 32-2. Recommended UART Baud Rate based on UPDICLKSEL setting**

| UPDICLKSEL[1:0] | MAX Recommended Baud Rate | MIN Recommended Baud Rate |
|---|---|---|
| 0x1 (16MHz) | 0.9Mbps | 0.300kbps |
| 0x2 (8MHz) | 450kbps | 0.150kbps |
| 0x3 (4MHz) - Default | 225kbps | 0.075kbps |

The UPDI baud rate generator utilizes fractional baud counting to minimize the transmission error. With the fixed frame format used by the UPDI, the maximum and recommended receiver transmission error limits can be seen in the following table:

**Table 32-3. Receiver Baud Rate Error**

| Data + parity bits | $R_{slow}$ | $R_{fast}$ | Max Total Error (%) | Recommended max RX error (%) |
|---|---|---|---|---|
| 9 | 96.39 | 104.76 | +4.76 / -3.61 | +1.5 / -1.5 |

#### 32.3.1.2 BREAK Character

The BREAK character is used to reset the internal state of the UPDI to the default setting. This is useful if the UPDI enters an error state due to communication error, or when synchronization between the debugger and the UPDI is lost.

A single BREAK character is enough to reset the UPDI, but in some special cases where the BREAK character is sent when the UPDI has not yet entered the error state, a double BREAK character might be needed. A double BREAK is guaranteed to reset the UPDI from any state. When sending a double BREAK it is required to have at least one stop bit between the BREAK characters.

No SYNCH character is required before the BREAK, because the BREAK is used to reset the UPDI from any state. This means that the UPDI will sample the BREAK based on the last stored baud rate setting, derived from the last received valid SYNCH character. If the communication error was due to incorrect sampling of the SYNCH character, the baud rate is unknown to the connected debugger. For this reason

the BREAK character should be transmitted at the slowest recommended baud rate setting for the selected UPDI clock according to Table 32-4:

**Table 32-4. Recommended BREAK Character Duration**

| UPDICLKSEL[1:0] | Recommended BREAK Character Duration |
|---|---|
| 0x1 (16MHz) | 6.15ms |
| 0x2 (8MHz) | 12.30ms |
| 0x3 (4MHz) - Default | 24.60ms |

### 32.3.2 Operation

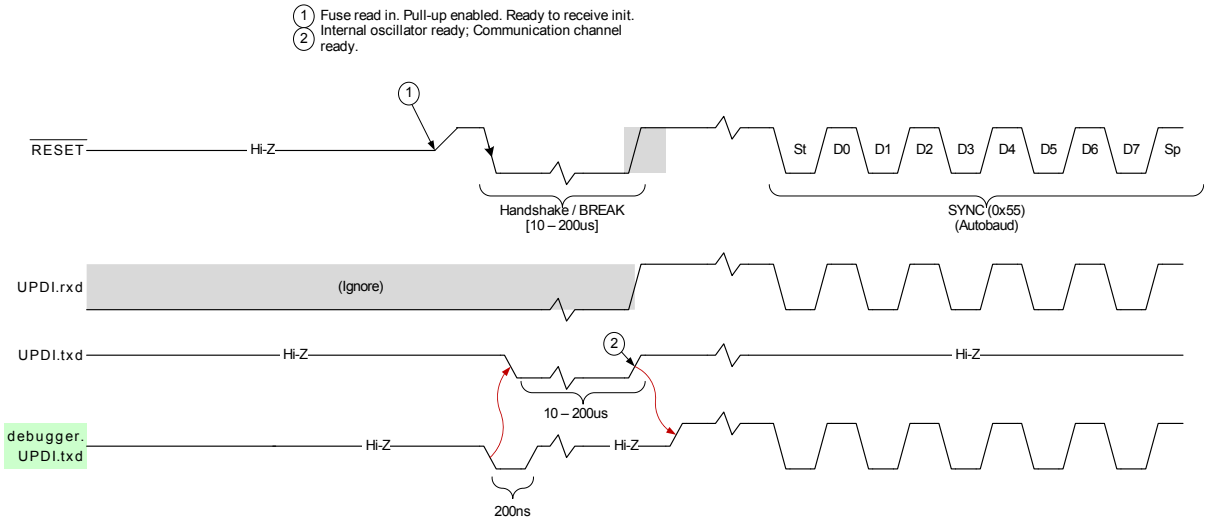
The UPDI must be enabled before the UART communication can start:

- UPDI Enable with 12V Override of RESET pin
- UPDI Enable with Fuse Override of RESET pin

#### 32.3.2.1 UPDI Enable with Fuse Override of $\overline{\text{RESET}}$ pin

When the $\overline{\text{RESET}}$ Pin Configuration (RSTPINCFG) bits in FUSE.SYSCFG0 are 0x1, the $\overline{\text{RESET}}$ pin will be overridden, and the UPDI will take control of the pin and configure it as input with pull-up. When the pull-up is detected by a connected debugger, the UPDI enable sequence, as depicted below, is started.

**Figure 32-5. UPDI Enable sequence with UPDI PAD Enabled by Fuse**




When the pull-up is detected, the debugger initiates the enable sequence by driving the line low for a minimum of 200ns and a maximum of 1us to ensure that the line is released from the debugger before the UPDI enable sequence is done.

The negative edge is detected by the UPDI, which requests the UPDI clock. The UPDI will continue to drive the line low until the clock is stable and ready for the UPDI to use. The duration of this will vary, depending on the status of the oscillator when the UPDI is enabled. A start-up time between 10us and 200us can be expected. After this duration, the data line will be released by the UPDI, and pulled-high.

When the Debugger detects that the line is high, the initial SYNCH character (0x55) must be sent to properly enable the UPDI for communication. If the start bit of the SYNCH character is not sent within 13.5ms, the UPDI will disable itself, and the enable sequence must be repeated. This time is based on counted cycles on the 4MHz UPDI clock, which is default when enabling the UPDI. The disable is performed to avoid the UPDI being enabled unintentionally.

After successful SYNCH character transmission, the first instruction frame can be transmitted.

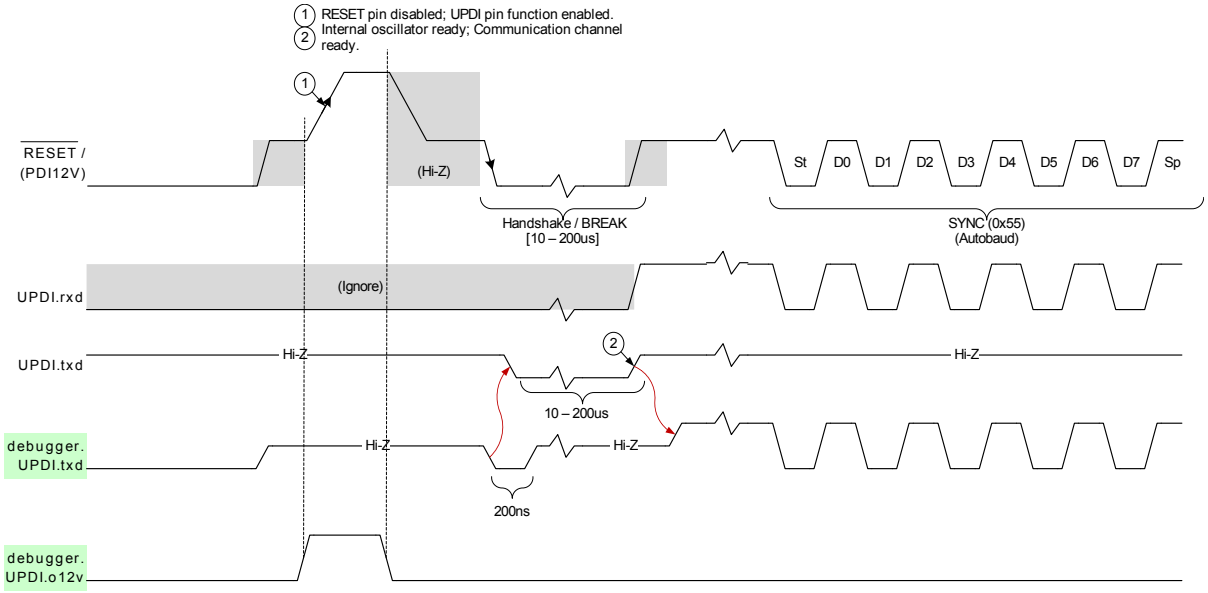
#### 32.3.2.2 UPDI Enable with 12V Override of RESET pin

GPIO or Reset functionality on the RESET pin can be overridden by the UPDI by using 12V programming. By applying a 12V pulse to the RESET pin, the pin functionality is switched to UPDI, independent of RSTPINCFG in FUSE.SYSCFG0. It is recommended to always reset the device before starting the 12V enable sequence.

During power-up, the Power-on Reset (POR) must be released before the 12V pulse can be applied. The duration of the pulse is recommended in the range from 100us to 1ms, before tri-stating. When applying the rising edge of the 12V pulse, the UPDI will be reset. After tri-stating, the UPDI will remain in reset until the RESET pin is driven low by the debugger. This will release the UPDI reset, and initiate the same enable sequence as explained in UPDI Enable with Fuse Override of RESET pin.

The following figure shows the 12V enable sequence.

**Figure 32-6. UPDI Enable Sequence by 12V Programming**




When enabled by 12V, only a POR will disable the UPDI configuration on the RESET pin, and restore the default setting. If issuing a UPDI disable command through the UPDIDIS bit in UPDI.CTRLB, the UPDI will be reset and the clock request will be canceled, but the RESET pin will remain in UPDI configuration.

#### 32.3.2.3 UPDI Disable

Any programming or debug session should be terminated by writing the UPDIDIS bit in UPDI.CTRLB. Writing this bit will reset the UPDI including any decoded KEYs, and disable the oscillator request for the module. If the disable operation is not performed, the UPDI will stay enabled and request its oscillator, causing increased power consumption for the application.

During the enable sequence the UPDI can disable itself in case of a faulty enable sequence. There are two cases which will cause an automatic disable.

- A SYNCH character is not sent within 13.5ms after the initial enable pulse described in UPDI Enable with Fuse Override of RESET pin.
- The first SYNCH character after an initiated enable is too short or too long to register as a valid SYNCH character. See Table 32-2 for recommended baud rate operating ranges.

#### 32.3.2.4 Output Enable Timer Protection for GPIO Configuration

When the RESET Pin Configuration (RSTPINCFG) bits in FUSE.SYSCFG0 are 0x0, the RESET pin configured as GPIO. To avoid the potential conflict between the GPIO actively driving the output and a

12V UPDI enable sequence initiation, a timer protection is disabling the output enable for a minimum time of 8.8ms after each System Reset.

It is always recommended to issue a System Reset before entering the 12V programming sequence.

#### 32.3.2.5 UPDI Communication Error Handling

The UPDI contains a comprehensive error detection system, to be able to provide information to the debugger when recovering from an error scenario. The error detection consist of detecting physical transmission errors like start bit error, parity error, contention error and frame error, to more high level errors like access timeout error. See UPDI_STATUSB.PESIG for an overview of the available error signatures.

Whenever the UPDI detects an error, it will immediately transfer to an internal error state to avoid unwanted system communication. In the error state the UPDI will ignore all incoming data requests, except if a BREAK character is transmitted. The following procedure should always be applied when recovering from an error condition.

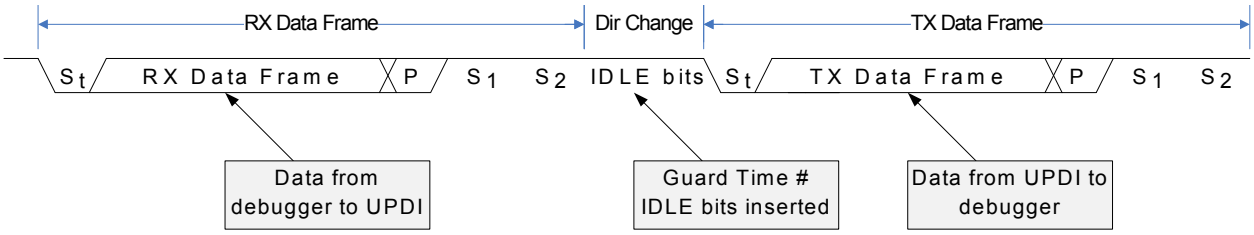
- Send a BREAK character. See BREAK Character for recommended BREAK character handling.
- Send a SYNCH character at the desired baud rate for the next data transfer. Upon receiving a BREAK, the UPDI oscillator setting in UPDI.ASI_CTRLA is reset to the 4MHz default UPDI clock selection. This affects the baud rate range of the UPDI according to Table 32-2 .
- Do a Load Control Status (LDCS) to UPDI.STATUSB register to read the PESIG field. PESIG will give information about the occurred error, and the error signature will be cleared when read.
- The UPDI is now recovered from the error state, and ready to receive the next SYNCH character and instruction.

#### 32.3.2.6 Direction Change

In order to ensure correct timing for half-duplex UART operation, the UPDI has a built in Guard Time mechanism to relax the timing when changing direction from RX mode to TX mode. The Guard Time is a number of IDLE bits inserted before the next start bit is transmitted. The number of IDLE bits can be configured through GTVAL in UPDI.CTRLA. The duration of each IDLE bit is given by the baud rate used by the current transmission.

It is not recommended to use GTVAL setting 0x7, with no additional IDLE bits.

**Figure 32-7. UPDI Direction Change by inserting IDLE bits**



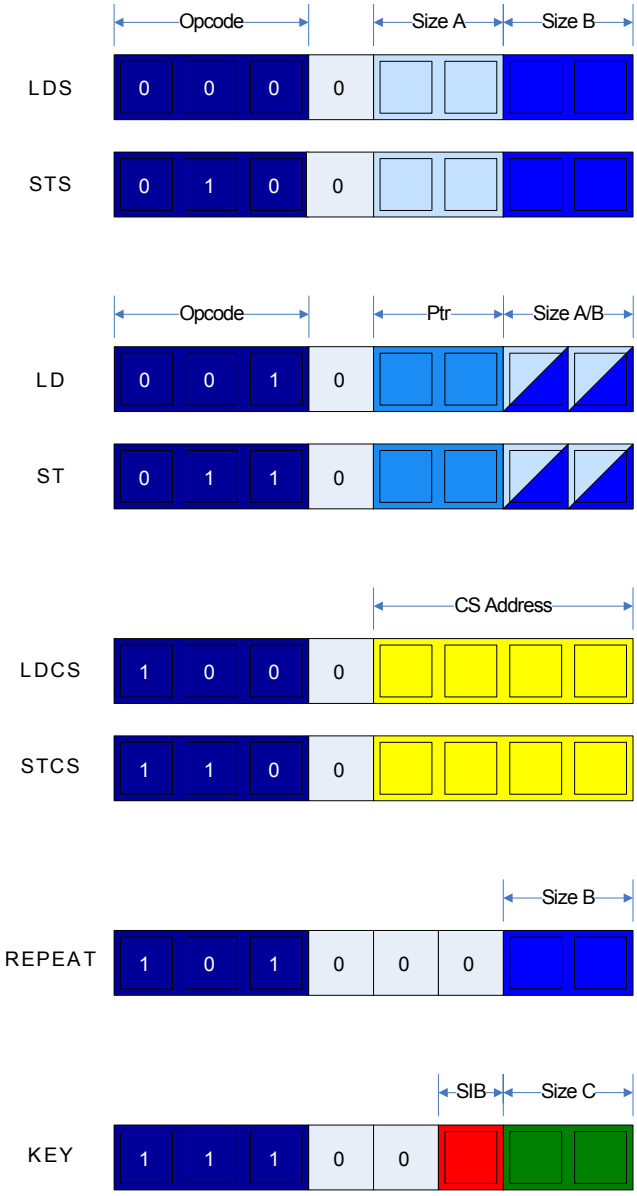

The UPDI Guard Time is the minimum IDLE time that the connected debugger will experience when waiting for data from the UPDI. Because of the asynchronous interface to the system, as presented in Clocks, the ratio between the UPDI clock and the system clock will affect the synchronization time, and how long it takes before the UPDI can transmit data. In the cases where the synchronization delay is shorter than the current Guard Time setting, the Guard Time will be given by GTVAL directly.

### 32.3.3 UPDI Instruction Set

Communication through the UPDI is based on a small instruction set. The instructions are used to access the internal UPDI and ASI Control and Status (CS) space, as well as the memory mapped system space. All instructions are byte instructions, and must be preceded by a SYNCH character to determine the baud

rate for the communication. See UPDI UART for information about setting the baud rate for the transmission. The following Figure gives an overview of the UPDI instruction set.

**Figure 32-8. UPDI Instruction Set Overview**




| OPCODE | |
|---|---|
| 0 0 0 | LDS |
| 0 0 1 | LD |
| 0 1 0 | STS |
| 0 1 1 | ST |
| 1 0 0 | LDCS (LDS Control/Status) |
| 1 0 1 | REPEAT |
| 1 1 0 | STCS (STS Control/Status) |
| 1 1 1 | KEY |

| Size A - Address size | |
|---|---|
| 0 0 | Byte |
| 0 1 | Word (2 Bytes) |
| 1 0 | Reserved |
| 1 1 | Reserved |

| Ptr - Pointer access | |
|---|---|
| 0 0 | *(ptr) |
| 0 1 | *(ptr++) |
| 1 0 | ptr |
| 1 1 | Reserved |

| Size B - Data size | |
|---|---|
| 0 0 | Byte |
| 0 1 | Word (2 Bytes) |
| 1 0 | Reserved |
| 1 1 | Reserved |

| CS Address (CS - Control/Status reg.) | |
|---|---|
| 0 0 0 0 | Reg 0 |
| 0 0 0 1 | Reg 1 |
| 0 0 1 0 | Reg 2 |
| 0 0 1 1 | Reg 3 |
| 0 1 0 0 | Reg 4 (ASI CS space) |
| ..... | |
| 1 1 1 1 | Reserved |

| Size C - Key size | |
|---|---|
| 0 0 | 64 bits (8 Bytes) |
| 0 1 | 128 bits (16 Bytes) |
| 1 0 | Reserved |
| 1 1 | Reserved |

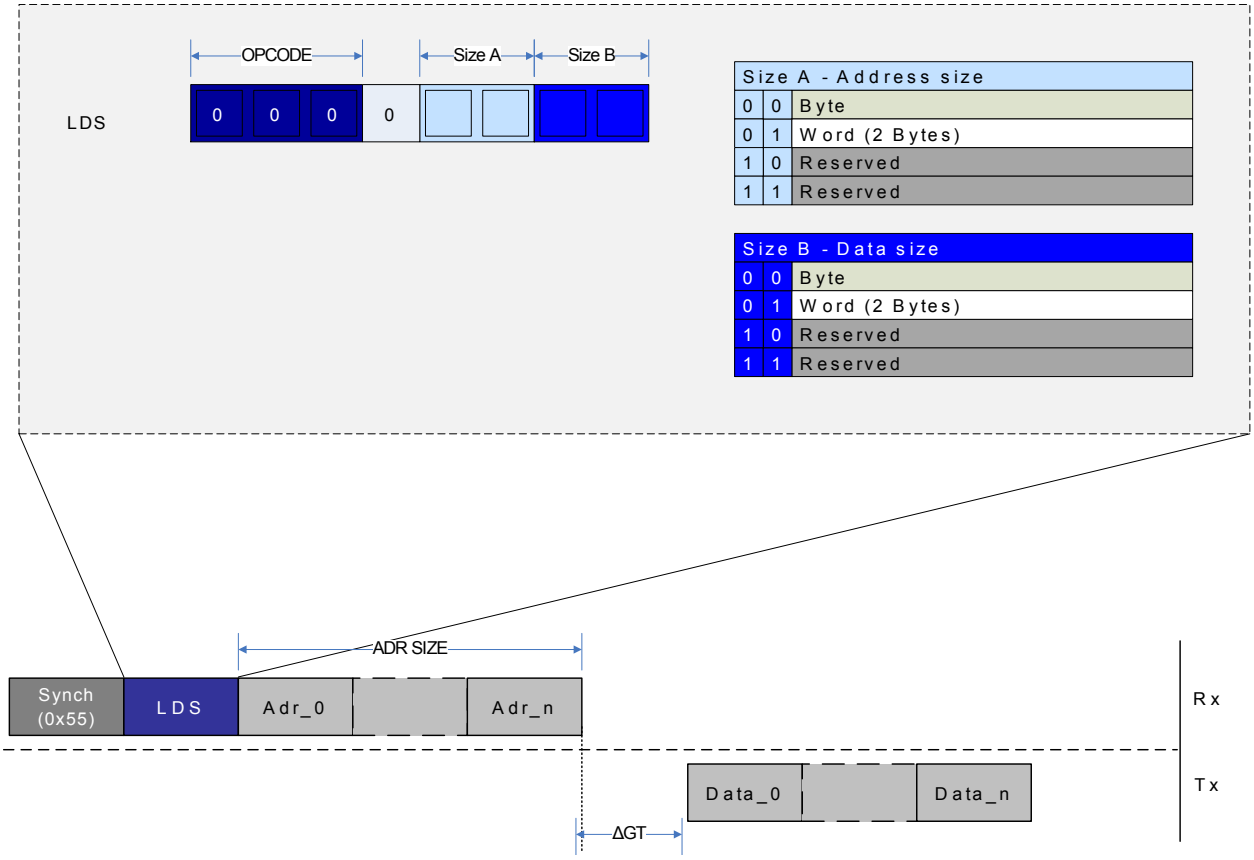
| SIB – System Information Block sel. | |
|---|---|
| 0 | Receive KEY |
| 1 | Send SIB |

#### 32.3.3.1 LDS - Load Data from Data Space Using Direct Addressing

The LDS instruction is used to load data form the bus matrix and into the serial shift register for serial read out. The LDS instruction is based on direct addressing, and the address must be given as an operand to the instruction for the data transfer to start. Maximum supported size for address and data is 16 bit. LDS supports repeated memory access when combined with the REPEAT instruction.

As shown in Figure 32-9, after issuing the SYNCH character followed by the LDS instruction, the number of desired address bytes, as indicated by the SizeA field in the instruction, must be transmitted. The output data size is selected by the SizeB field, and is issued after the specified Guard Time. When combined with the REPEAT instruction, the address must be sent in for each iteration of the repeat, meaning after each time the output data sampling is done. There is no automatic address increment when using REPEAT with LDS, as it uses a direct addressing protocol.

**Figure 32-9. LDS Instruction Operation**

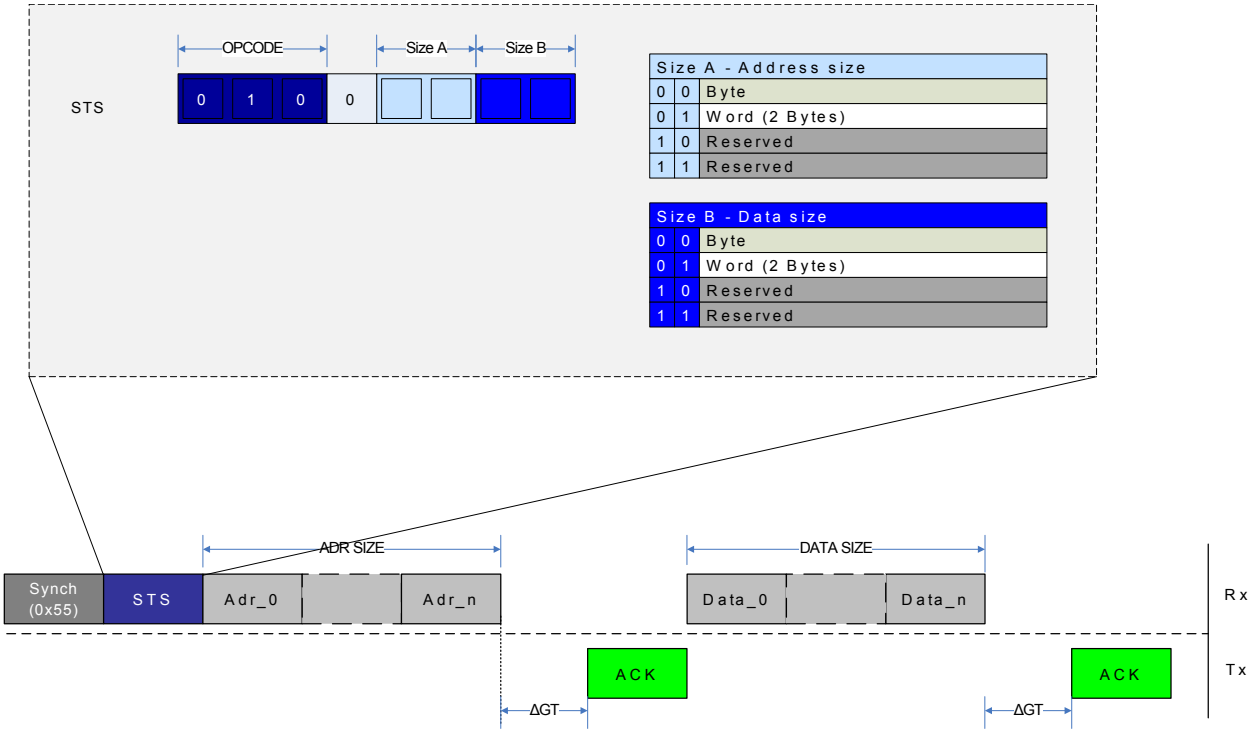



#### 32.3.3.2 STS - Store Data to Data Space Using Direct Addressing

The STS instruction is used to store data that is shifted serially into the PHY layer to the bus matrix address space. The STS instruction is based on direct addressing, where the address is the first set of operands, and data is the second set. The size of the address and data operands are given by the size fields presented in the Figure below. Maximum size for both address and data is 16 bit.

STS supports repeated memory access when combined with the REPEAT instruction.

**Figure 32-10. STS Instruction Operation**



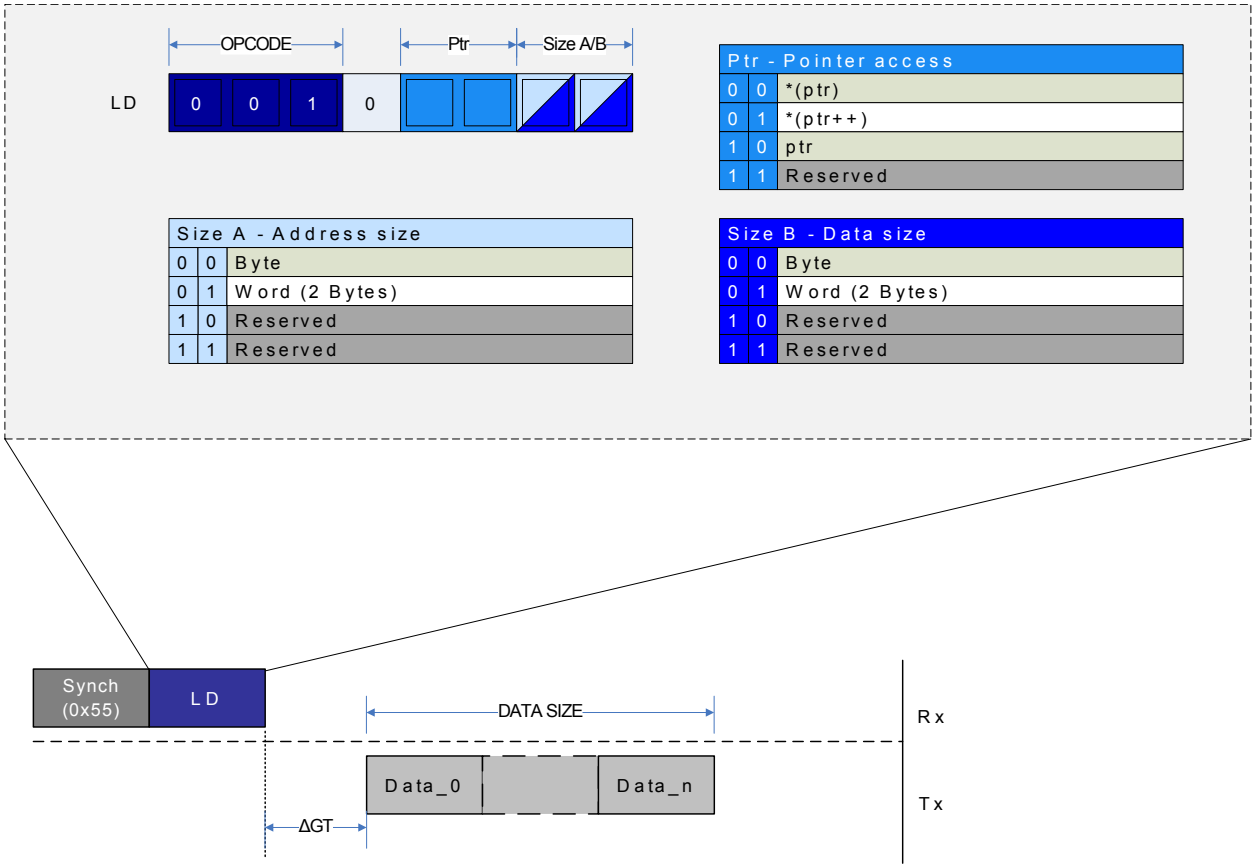

The transfer protocol for an STS instruction is depicted in the Figure as well, following this sequence:

1. The address is sent.
2. An Acknowledge (ACK) is sent back from the UPDI if the transfer was successful.
3. The number of bytes as specified in the STS instruction is sent.
4. A new ACK is received after the data has been successfully transferred.

#### 32.3.3.3 LD - Load Data from Data Space Using Indirect Addressing

The LD instruction is used to load data form the bus matrix and into the serial shift register for serial read out. The LD instruction is based on indirect addressing, which means that the address pointer in the UPDI needs to be written prior to bus matrix access. Automatic pointer post increment operation is supported, and is useful when the LD instruction is used with REPEAT. It is also possible to do a LD of the UPDI pointer register. Maximum supported size for address and data load is 16 bit.

**Figure 32-11. LD Instruction Operation**

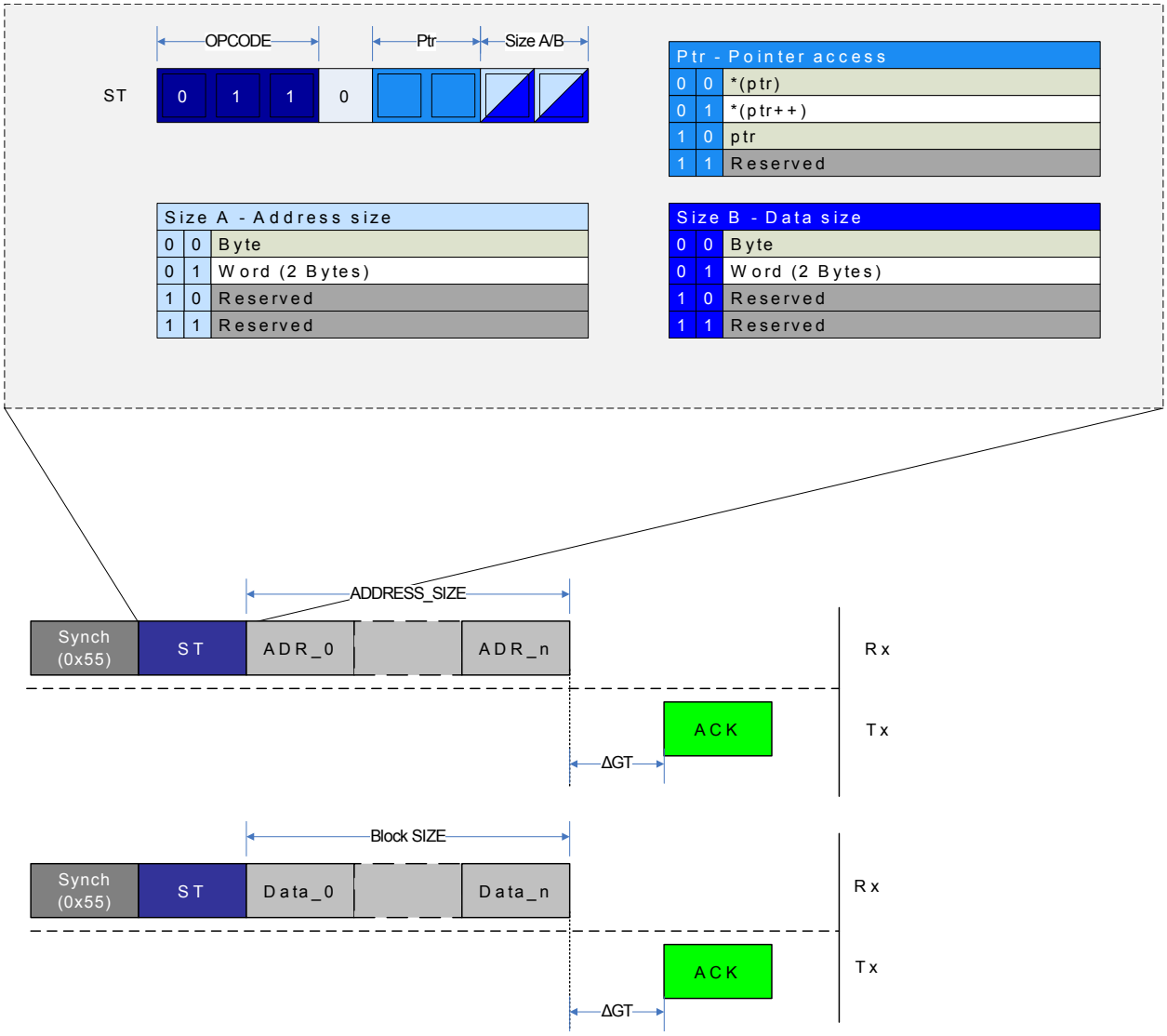



The Figure above shows an example of a typical LD sequence, where data is received after the Guard Time period. Loading data from the UPDI pointer register follows the same transmission protocol.

#### 32.3.3.4 ST - Store Data from Data Space Using Indirect Addressing

The ST instruction is used to store data that is shifted serially into the PHY layer to the bus matrix address space. The ST instruction is based on indirect addressing, which means that the address pointer in the UPDI needs to be written prior to bus matrix access. Automatic pointer post increment operation is supported, and is useful when the ST instruction is used with REPEAT. ST is also used to store the UPDI address pointer into the pointer register. Maximum supported size for storing address and data is 16 bit.

**Figure 32-12. ST Instruction Operation**




The Figure above gives an example of ST to the UPDI pointer register, and store of regular data. In both cases an Acknowledge (ACK) is sent back by the UPDI if the store was successful, and a SYNCH character is sent before each instruction. To write the UPDI pointer register, the following procedure should be followed.

- Set the PTR field in the ST instruction to the signature 0x2
- Set the address size field SizeA to the desired address size
- After issuing the ST instruction, send SizeA bytes of address data
- Wait for the ACK character which signifies a successful write to the address register

After the address register is written, sending data is done in a similar fashion

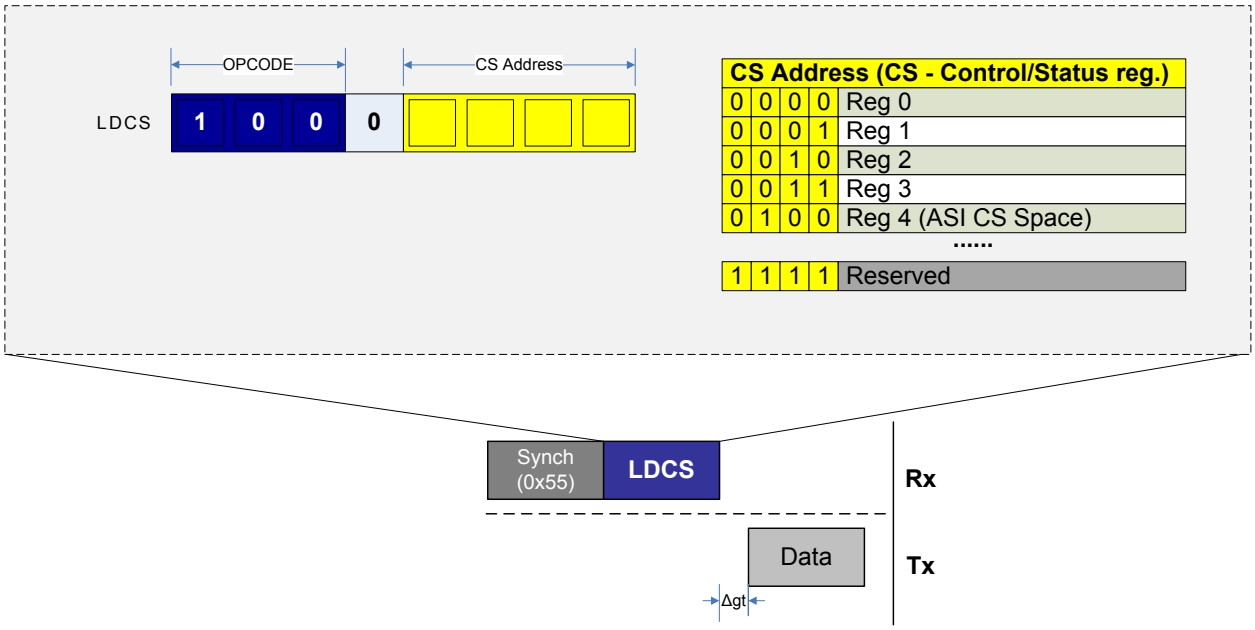
- Set the PTR field in the ST instruction to the signature 0x0 to write to the address specified by the UPDI pointer register. If the PTR field is set to 0x1, the UPDI pointer is automatically updated to the next address according to the data size SizeD field of the instruction after the write is executed.
- Set the SizeD field in the instruction to the desired data size.
- After sending the ST instruction, send SizeD bytes of address data
- Wait for the ACK character which signifies a successful write to the bus matrix

When used with the REPEAT, it is recommended to set up the address register with the start address for the block to be written, and use pointer post increment register to automatically increase the address for each repeat cycle. When using repeat, the data frame of SizeD data bytes can be sent after each received ACK.

#### 32.3.3.5 LCDS - Load Data from Control and Status Register Space

The LCDS instruction is used to load data from the UPDI and ASI CS-space. LCDS is based on direct addressing, where the address is part of the instruction opcode. The total address space for LCDS is 16 byte, and can only access the internal UPDI register space. This instruction only supports byte access, and data size is not configurable.

**Figure 32-13. LCDS Instruction Operation**




The Figure above shows a typical example of LCDS data transmission. A data byte from the LCDS space is transmitted from the UPDI after the Guard-Time is completed.

#### 32.3.3.6 STCS (Store Data to Control and Status register space)

The STCS instruction is used to store data to the UPDI and ASI CS-space. STCS is based on direct addressing, where the address is part of the instruction opcode. The total address space for STCS is 16 byte, and can only access the internal UPDI register space. This instruction only supports byte access, and data size is not configurable.

**Figure 32-14. STCS instruction operation**

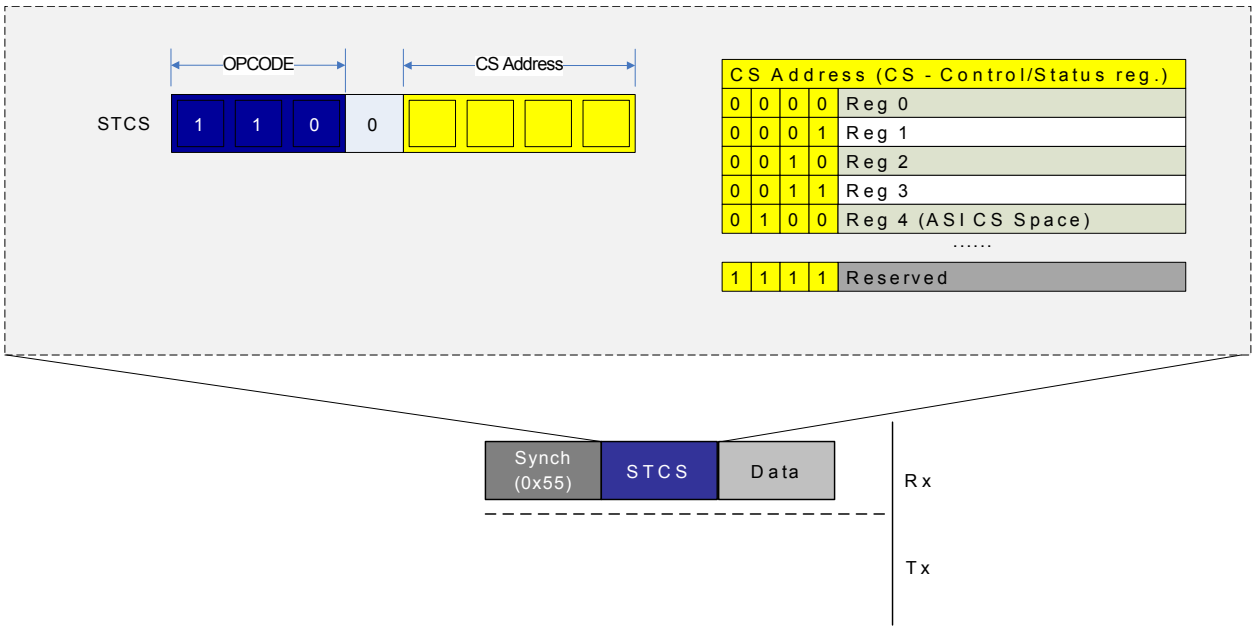


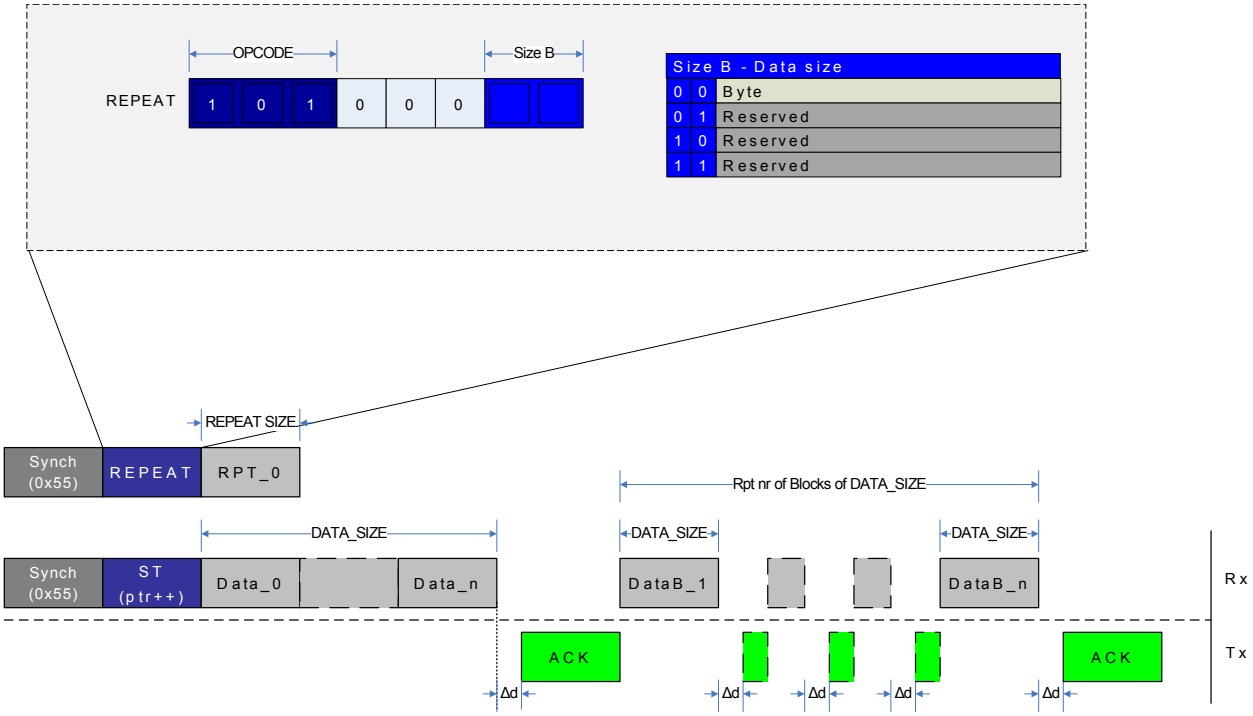

Figure 32-14 shows the data frame transmitted after the SYNCH and instruction frames. There is no response generated from the STCS instruction, as is the case for ST and STS.

#### 32.3.3.7 REPEAT - Set Instruction Repeat Counter

The REPEAT instruction is used to store the repeat count value into the UPDI repeat counter register. When instructions are used with REPEAT, protocol overhead for SYNCH- and Instruction Frame can be omitted on all instructions except the first instruction after the REPEAT is issued. REPEAT is most useful for memory instructions (LD, ST, LDS, STS), but all instructions can be repeated, except the REPEAT instruction itself.

The DATA_SIZE opcode field refers to the size of the repeat value. Only byte size (up to 255 repeats) is supported. The instruction that is loaded directly after the REPEAT instruction will be repeated RPT_0 times. The instruction will be issued a total of RPT_0 + 1 times. An ongoing repeat can only be aborted by sending a BREAK character.

**Figure 32-15. REPEAT Instruction Operation**



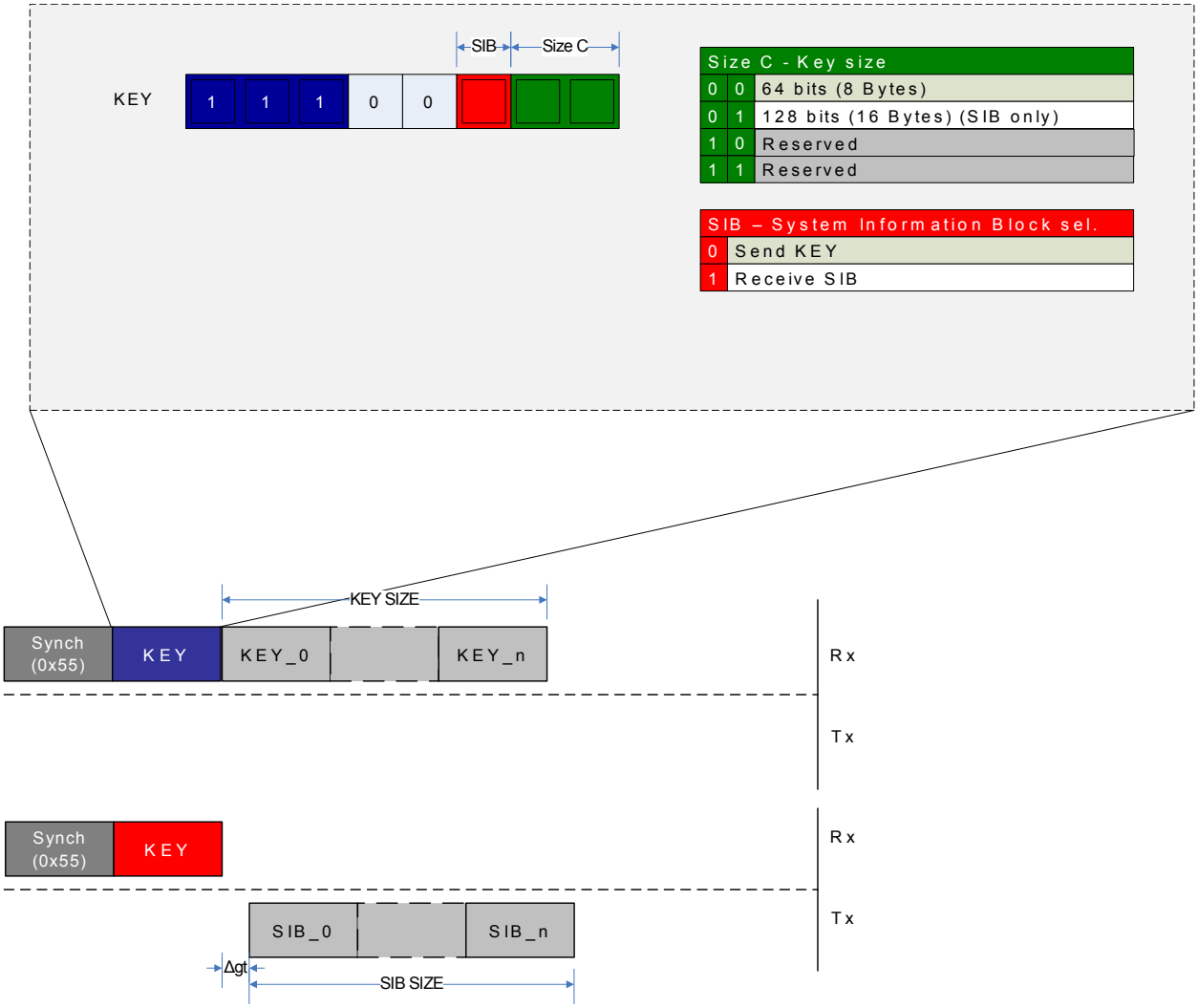

The Figure above gives an example of repeat operation with a ST instruction using pointer post increment operation. After the REPEAT instruction is sent with RPT_0 = ***n***, the first ST instruction is issued with SYNCH- and Instruction frame, while the next ***n*** ST-instructions are executed by only sending in data bytes according to the ST operand DATA_SIZE, and maintaining the Acknowledge (ACK) handshake protocol.

If using indirect addressing instructions (LD/ST) it is recommended to always use the pointer post increment option when combined with REPEAT . Otherwise the same address will be accessed in all repeated access operations. For direct addressing instructions (LDS/STS), the address must always be transmitted as specified in the instruction protocol, before data can be received (LDS) or sent (STS).

### 32.3.3.8 KEY - Set Activation KEY

The KEY instruction is used for communicating KEY bytes to the UPDI, opening up for executing protected features on the device. See Table 32-5for an overview over functions that are activated by KEYs. For the KEY instruction, only 64bit KEY size is supported. If the System Information Block (SIB) field of the KEY instruction is set, the KEY instruction returns the SIB instead of expecting incoming KEY bytes. Maximum supported size for SIB is 128bits.

**Figure 32-16. `KEY` Instruction Operation**




The Figure above shows the transmission of a KEY, and the reception of a SIB. In both cases, the `SIZE_C` field in the opcode determines the number of frames to be sent or received. There is no response after sending a `KEY` to the UPDI. When requesting the SIB, data will be transmitted from the UPDI according to the current Guard Time setting.

### 32.3.4 System Clock Measurement with UPDI

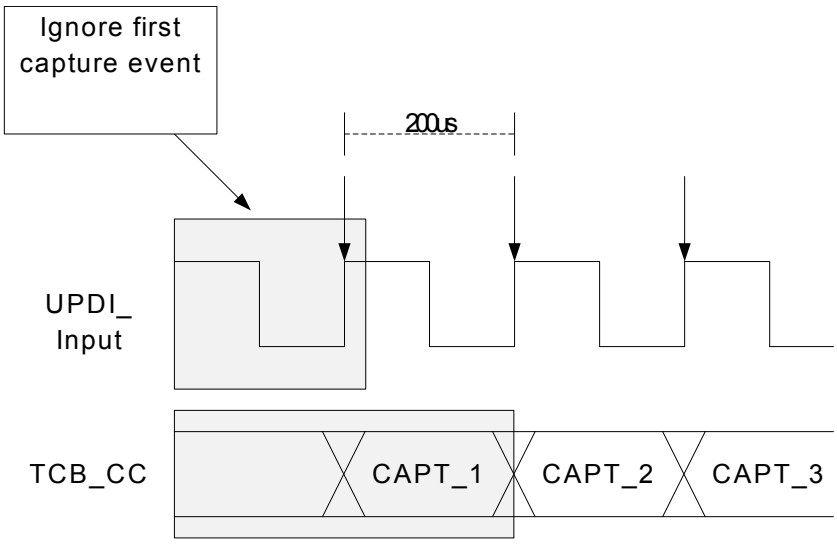
It is possible to use the UPDI to get a accurate measurement of the system clock frequency, by using the UPDI event connected to TCB with Input Capture capabilities. A recommended setup flow for this feature is given by the following steps:

- Set up TCB.CTRLB with setting CNTMODE=0x3, Input Capture frequency measurement mode. See for details.
- Write CAPTEI=1 in TCB.EVCTRL to enable Event Interrupt. Keep EDGE = 0 in TCB.EVCTRL.
- Configure the Event System as described in Events
- For the SYNCH character used to generate the UPDI events, it is recommended to use a slow baud rate in the range of 10kbps - 50kbps to get a more accurate measurement on the value captured by the timer between each UPDI event. One particular thing is that if the capture is set up to trigger an interrupt, the first captured value should be ignored. The second captured value based

on the input event should be used for the measurement. See the figure below for an example using 10kbps UPDI SYNCH character pulses, giving a capture window of 200µs for the Timer.

- It is possible to read out the captured value directly after the SYNCH character, by reading the TCB.CC register, or the value can be written to memory by the CPU once the capture is done.

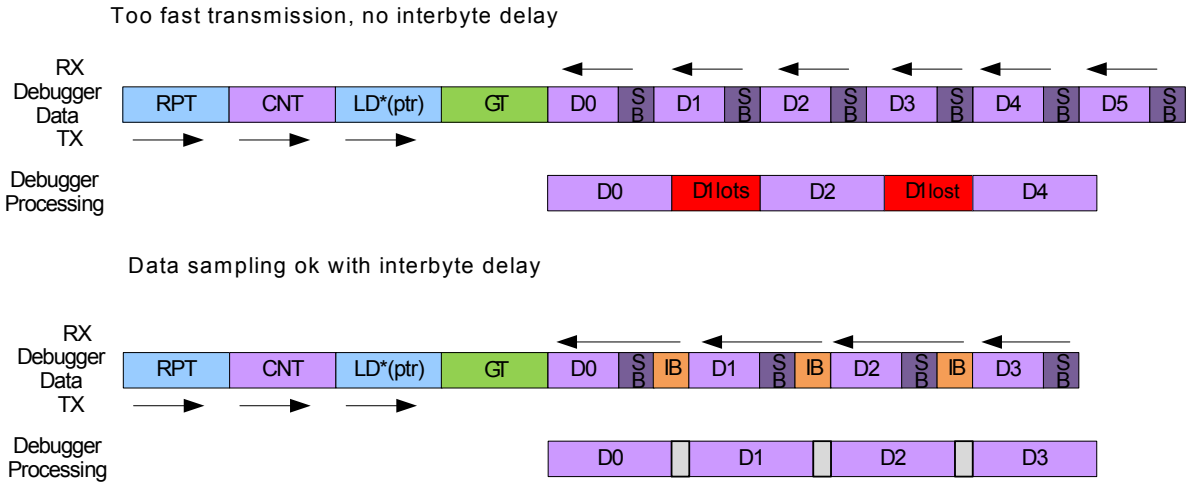
**Figure 32-17. UPDI System Clock Measurement Events**




### 32.3.5 Interbyte Delay

When loading data with the UPDI, or reading out the System Information Block, the output data will normally come out with two IDLE bits between each transmitted byte for a multibyte transfer. Depending on the application on the receiver side, data might be coming out too fast when there is no extra IDLE bits between each byte. By enabling the IBDLY feature in UPDI.CTRLB, two extra stop bits will be inserted between each byte to relax the sampling time for the debugger. Interbyte delay works in the same way as guard time, by inserting extra IDLE bits, but only a fixed number of IDLE bits and only for multibyte transfers. The first transmitted byte after a direction change will be subject to the regular Guard Time before it is transmitted, and the interbyte delay is not added to this time.

**Figure 32-18. Interbyte Delay exemple with LD and RPT**




In Figure 32-18, GT denotes the Guard Time insertion, SB are stop bits and IB is the inserted interbyte delay. The rest of the frames are data and instructions.

### 32.3.6 System Information Block

The System Information Block (SIB) can be read out at any time by setting the SIB bit in the KEY instruction from KEY - Set Activation KEY. The SIB provides a compact form of providing information for the debugger which is vital in identifying and setting up the proper communication channel with the part. The output of the SIB should be interpreted as ASCII symbols. KEY size field should be set to 16byte when reading out the complete SIB, and 8 byte size can be used to read out only the Family_ID. See Figure 32-19 for SIB format description, and which data that is available at different readout sizes.

**Figure 32-19. System Information Block format**

| 16 | 8 | [Byte][Bits] | Field Name |
|---|---|---|---|
| | | [6:0] [55:0] | Family_ID |
| | | [7][7:0] | Reserved |
| | | [10:8][23:0] | NVM_VERSION |
| | | [13:11][23:0] | OCD_VERSION |
| | | [14][7:0] | RESERVED |
| | | [15][7:0] | DBG_OSC_FREQ |

### 32.3.7 Enabling of KEY Protected Interfaces

Access to some internal interfaces and features are protected by the UPDI KEY mechanism. To activate a KEY, the correct KEY data must be transmitted by using the `KEY` instruction as described in KEY instruction. Table 32-5 describes the available KEYs, and the condition required when doing operation with the KEY active. There is no requirement when shifting in the KEY, but you would for instance normally run a Chiperase before enabling the NVMPROG KEY, to unlock the device for debug. But if the NVMPROGKEY is shifted in first, it will not be reset by shifting in the Chiperase KEY afterwards.

**Table 32-5. KEY Activation Overview**

| KEY name | Description | Requirements for operation | Reset |
|---|---|---|---|
| Chiperase | Start NVM Chiperase. Clear Lockbits | None | UPDI Disable / UPDI Reset |
| NVMPROG | Activate NVM Programming | Lockbits Cleared. ASI_SYS_STATUS.NVMPROG set. | Programming Done / UPDI Reset |
| USERROW-Write | Program User Row on Locked part | Lockbits Set. ASI_SYS_STATUS.UROWPROG set | Write to KEY status bit / UPDI Reset |

Table 32-6 gives an overview of the available KEY signatures that must be shifted in to activate the interfaces.

**Table 32-6. KEY Activation Signatures**

| KEY name | KEY signature (LSB written first) | Size |
|---|---|---|
| Chiperase | 0x4E564D4572617365 | 64bit |
| NVMPROG | 0x4E564D50726F6720 | 64bit |
| USERROW-Write | 0x4E564D5573267465 | 64bit |

#### 32.3.7.1 Chip Erase

The following steps should be followed to issue a Chip Erase.

1. Enter the CHIPERASE KEY by using the `KEY` instruction. See Table 32-6 for the CHIPERASE signature.
2. **Optional:** Read the Chip Erase bit in the AS Key Status register (CHIPERASE in UPDI.ASI_KEY_STATUS) to see that the KEY is successfully activated.
3. Write the Reset signature into the UPDI.ASI_RESET_REQ register. This will issue a System Reset.
4. Write 0x00 to the ASI Reset Request register (UPDI.ASI_RESET_REQ) to clear the System Reset.
5. Read the Lock Status bit in the ASI System Status register (LOCKSTATUS in UPDI.ASI_SYS_STATUS).
6. Chip Erase is done when LOCKSTATUS == 0 in UPDI.ASI_SYS_STATUS. If LOCKSTATUS == 1, go to point 5 again.

After a successful Chip Erase, the Lockbits will be cleared, and the UPDI will have full access to the system. Until Lockbits are cleared, the UPDI cannot access the system bus, and only CS-space operations can be performed. See User Row Programming.

**Caution:** During Chip Erase, the BOD is forced ON (ACTIVE=0x1 in BOD.CTRLA), and uses the BOD Level from the BOD Configuration fuse (LVL in BOD.CTRLB = LVL in FUSE.BODCFG). If the supply voltage $V_{DD}$ is below that threshold level, the device is unserviceable until VDD is increased adequately.

#### 32.3.7.2 NVM Programming

If the device is unlocked, it is possible to write directly to the NVM Controller using the UPDI - this will lead to unpredictable code execution if the CPU is active during the NVM programming. To avoid this, the following NVM Programming sequence should be executed.

1. Follow the Chiperase procedure as described in Chip Erase. If the part is already unlocked, this point can be skipped.
2. Enter the NVMPROG KEY by using the `KEY` instruction. See Table 32-6 for the NVMPROG signature.
3. **Optional:** Read the NVMPROG field in the KEY_STATUS register to see that the KEY has been activated.
4. Write the Reset signature into the ASI_RESET_REQ register. This will issue a System Reset.
5. Write 0x00 to the Reset signature in ASI_RESET_REQ register to clear the System Reset.
6. Read NVMPROG in ASI_SYS_STATUS.
7. NVM Programming can start when NVMPROG == 1 in the ASI_SYS_STATUS register. If NVMPROG == 0, go to point 6 again.
8. Write data to NVM through the UPDI.
9. Write the Reset signature into the ASI_RESET_REQ register. This will issue a System Reset.
10. Write 0x00 to the Reset signature in ASI_RESET_REQ register to clear the System Reset.
11. Programming is complete.

#### 32.3.7.3 User Row Programming

The User Row Programming feature allows the user to program new values to the User Row (USERROW) on a locked device. To program with this functionality enabled, the following sequence should be followed.

1. Enter the USERROW-Write KEY located in Table 32-6 by using the `KEY` instruction. See Table 32-6 for the UROWWRITE signature.
2. **Optional:** Read the UROWWRITE bit field in UPDI.ASI_KEY_STATUS to see that the KEY has been activated.
3. Write the Reset signature into the UPDI.ASI_RESET_REQ register. This will issue a System Reset.
4. Write 0x00 to the Reset signature in UPDI.ASI_RESET_REQ register to clear the System Reset.
5. Read UROWPROG bit in UPDI.ASI_SYS_STATUS.
6. User Row Programming can start when UROWPROG == 1. If UROWPROG == 0, go to point 5 again.
7. The writable area has a size of one EEPROM page, 32 byte, and it is only possible to write User Row data to the first 32 byte addresses of the RAM. Addressing outside this memory range will result in a non executed write. The data will map 1:1 with the User Row space, when the data is copied into the User Row upon completion of the Programming sequence.
8. When all User Row data has been written to the RAM, write the UROWWRITEFINAL bit in UPDI.ASI_SYS_CTRLA.
9. Read the UROWPROG bit in UPDI.ASI_SYS_STATUS.
10. The User Row Programming is completed when UROWPROG == 0. If UROWPROG == 1, go to point 9 again.
11. Write the UROWWRITE bit in UPDI.ASI_KEY_STATUS.
12. Write the Reset signature into the UPDI.ASI_RESET_REQ register. This will issue a System Reset.
13. Write 0x00 to the Reset signature in UPDI.ASI_RESET_REQ register to clear the System Reset.
14. User Row Programming is complete.

It is not possible to read back data from the SRAM in this mode. Only writes to the first 32 bytes of the SRAM is allowed.

### 32.3.8 Events

The UPDI is connected to the Event System (EVSYS) as described in ASYNCCH0, ASYNCCH1, ASYNCCH2, ASYNCCH3.

The UPDI can generate the following output events:

- SYNCH Character Positive Edge Event

This Event is set on the UPDI clock for each detected positive edge in the SYNCH character, and it is not possible to disable this event from the UPDI. The recommended application for this Event is system clock frequency measurement through the UPDI. Section System Clock Measurement with UPDI provides the details on how to setup the system for this operation.

**Related Links**

EVSYS - Event System

### 32.3.9 Sleep Mode Operation

The UPDI physical layer runs independently of all sleep modes, and the UPDI is always accessible for a connected debugger independent of the device sleep mode. If the system enters a sleep mode that turns off the CPU clock, the UPDI will not be able to access the system bus and read memories and peripherals. The UPDI physical layer clock is unaffected by the sleep mode settings, as long as the UPDI is enabled. By reading the INSLEEP bit in UPDI.ASI_SYS_STATUS it is possible to monitor if the system domain is in sleep mode. The INSLEEP bit is set if the system is in IDLE sleep mode or deeper.

It is possible to prevent the system clock from stopping when going into sleep mode, by writing the CLKREQ bit in UPDI.ASI_SYS_CTRL to '1'. If this bit is set the system sleep mode state is emulated, and

it is possible for the UPDI to access the system bus and read out peripheral registers even in the deepest sleep modes.

CLKREQ in UPDI.ASI_SYS_CTRL is by default '1', which means that default operation is keeping the system clock on during sleep modes.

## 32.4 Register Summary - UPDI

| Offset | Name | Bit Pos. | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | STATUSA | 7:0 | UPDIREV[3:0] | | | | | | | |
| 0x01 | STATUSB | 7:0 | | | | | | PESIG[2:0] | | |
| 0x02 | CTRLA | 7:0 | IBDLY | | PARD | DTD | RSD | GTVAL[2:0] | | |
| 0x03 | CTRLB | 7:0 | | | | NACKDIS | CCDETDIS | UPDIDIS | | |
| 0x04 ... 0x06 | Reserved | | | | | | | | | |
| 0x07 | ASI_KEY_STATUS | 7:0 | | | UROWWRITE | NVMPROG | CHIPERASE | | | |
| 0x08 | ASI_RESET_REQ | 7:0 | RSTREQ[7:0] | | | | | | | |
| 0x09 | ASI_CTRLA | 7:0 | | | | | | | UPDICLKSEL[1:0] | |
| 0x0A | ASI_SYS_CTRLA | 7:0 | | | | | | | UROWWRITE_FINAL | CLKREQ |
| 0x0B | ASI_SYS_STATUS | 7:0 | | | RSTSYS | INSLEEP | NVMPROG | UROWPROG | | LOCKSTATUS |
| 0x0C | ASI_CRC_STATUS | 7:0 | | | | | | CRC_STATUS[2:0] | | |

## 32.5 Register Description

These registers are only readable through the UPDI with special instructions, and are NOT readable through the CPU.

Registers at offset addresses 0x0-0x3 are the UPDI Physical configuration registers

Registers at offset addresses 0x4-0xC are the ASI level registers.

### 32.5.1 Status A

**Name:** STATUSA
**Offset:** 0x00
**Reset:** 0x10
**Property:** -

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

**Bits 7:4 – UPDIREV[3:0]: UPDI Revision**
These bits are read-only and contain the revision of the current UPDI implementation.

### 32.5.2 Status B

**Name:** STATUSB
**Offset:** 0x01
**Reset:** 0x00
**Property:** -

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

**Bits 2:0 – PESIG[2:0]: UPDI Error Signature**
These bits describe the UPDI Error Signature, and is set when an internal UPDI error condition occurs. The PESIG field is cleared on a read from the debugger.

**Table 32-7. Valid Error Signatures**

| PESIG[2:0] | Error Type | Error Description |
|---|---|---|
| 0x0 | No error | No error detected (Default) |
| 0x1 | Parity error | Wrong sampling of the parity bit |
| 0x2 | Frame error | Wrong Sampling of frame stop bits |
| 0x3 | Access Layer Timeout Error | UPDI can get no data or response from the Access layer. Examples of error cases are system domain in sleep or system domain reset. |
| 0x4 | Clock recovery error | Wrong sampling of frame start bit |
| 0x5 | - | Reserved |
| 0x6 | Reserved | Reserved |
| 0x7 | Contention error | Signalize Driving Contention on the UPDI RXD/TXD line. |

### 32.5.3 Control A

**Name:** CTRLA
**Offset:** 0x02
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | IBDLY | | PARD | DTD | RSD | GTVAL[2:0] | | |
| 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 – IBDLY: Inter-Byte Delay Enable**
Writing a '1' to this bit enables a fixed inter-byte delay between each data byte transmitted from the UPDI when doing multi-byte LD(S). The fixed length is two IDLE characters. Before the first transmitted byte will use the regular GT delay used for direction change.

**Bit 5 – PARD: Parity Disable**
Writing this bit to '1' will disable parity detection in the UPDI by just ignoring the Parity bit. This feature is recommended only during testing.

**Bit 4 – DTD: Disable Timeout Detection**
Setting this bit disables the timeout detection on the PHY layer, which requests a response from the ACC layer within a specified time (65536 UPDI clock cycles).

**Bit 3 – RSD: Response Signature Disable**
Writing a '1' to this bit will disable any response signatures generated by the UPDI. This is to reduce the protocol overhead to a minimum when writing large blocks of data to the NVM space. Disabling the Response Signature should be used with caution, and only when the delay experienced by the UPDI when accessing the system bus is predictable, otherwise loss of data may occur.

**Bits 2:0 – GTVAL[2:0]: Guard Time Value**
This bit field selects the Guard Time Value that will be used by the UPDI when the transmission mode switches from RX to TX. The Guard time is equal to the Baud Rate used in 1-Wire mode.

| Value | Description |
|---|---|
| 0x0 | UPDI Guard Time: 128 cycles (default) |
| 0x1 | UPDI Guard Time: 64 cycles |
| 0x2 | UPDI Guard Time: 32 cycles |
| 0x3 | UPDI Guard Time: 16 cycles |
| 0x4 | UPDI Guard Time: 8 cycles |
| 0x5 | UPDI Guard Time: 4 cycles |
| 0x6 | UPDI Guard Time: 2 cycles |
| 0x7 | GT off (no extra Idle bits inserted) |

### 32.5.4 Control B

**Name:** CTRLB
**Offset:** 0x03
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | NACKDIS | CCDETDIS | UPDIDIS | | |
| Access | R | R | R | R | R | R | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | | |

**Bit 4 – NACKDIS: Disable NACK Response**
Writing this bit to '1' disables the NACK signature sent by the UPDI if a System Reset is issued during an ongoing LD(S) and ST(S) operation.

**Bit 3 – CCDETDIS: Collision and Contention Detection Disable**
If this bit is written to '0', contention detection is enabled for 1W mode. This means that the UPDI can detect a collision in an ongoing 1-Wire transmission.

**Bit 2 – UPDIDIS: UPDI Disable**
Writing a '1' to this bit disables the UPDI PHY interface. The clock request from the UPDI is lowered, and the UPDI is reset. All UPDI PHY configurations and KEYs will be reset when the UPDI is disabled.

### 32.5.5 ASI Key Status

**Name:** ASI_KEY_STATUS
**Offset:** 0x07
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | UROWWRITE | NVMPROG | CHIPERASE | | | |
| Access | | | R | R | R | | | |
| Reset | | | 0 | 0 | 0 | | | |

**Bit 5 – UROWWRITE: User Row Write Key Status**
This bit is set to '1' if the UROWWRITE KEY is active. Otherwise this bit reads as zero.

**Bit 4 – NVMPROG: NVM Programming**
This bit is set to '1' if the NVMPROG KEY is active. This bit is automatically reset after the programming sequence is done. Otherwise this bit reads as zero.

**Bit 3 – CHIPERASE: Chip Erase**
This bit is set to '1' if the CHIPERASE KEY is active. This bit will automatically be reset when the chip erase sequence is completed. Otherwise this bit reads as zero.

### 32.5.6 ASI Reset Request

**Name:** ASI_RESET_REQ
**Offset:** 0x08
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RSTREQ[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 – RSTREQ[7:0]: Reset Request**
A Reset is signalized to the System when writing the Reset signature 0x59h to this address.

Writing any other signature to this register will clear the Reset.

When reading this register, reading bit RSTREQ[0] will tell if the UPDI is holding an active Reset on the system. If this bit is '1', the UPDI has an active Reset request to the system. All other bits will read as '0'.

The UPDI will not be reset when issuing a System Reset from this register.

### 32.5.7 ASI Control A

**Name:** ASI_CTRLA
**Offset:** 0x09
**Reset:** 0x02
**Property:** -

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

**Bits 1:0 – UPDICLKSEL[1:0]: UPDI Clock Select**
Writing these bits selects the UPDI clock output frequency. Default setting after reset and enable is 4MHz. Any other clock output selection is only recommended when the BOD is at the highest level. For all other BOD settings, the default 4MHz selection is recommended.

| Value | Description |
|---|---|
| 0x0 | Reserved |
| 0x1 | 16MHz UPDI clock |
| 0x2 | 8MHz UPDI clock |
| 0x3 | 4MHz UPDI clock (Default Setting) |

### 32.5.8 ASI System Control A

**Name:** ASI_SYS_CTRLA
**Offset:** 0x0A
**Reset:** 0x00
**Property:** -

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

**Bit 1 – UROWWRITE_FINAL : User Row Programming Done**
This bit should be written through the UPDI when the user row data has been written to the RAM. Writing this bit will start the process of programming the user row data to the FLASH.

If this bit is written before the User Row code is written to RAM by the UPDI, the CPU will progress without the written data.

This bit is only writable if the Userrow-write KEY is successfully decoded.

**Bit 0 – CLKREQ: Request System Clock**
If this bit is written to '1', the ASI is requesting the system clock, independent of system sleep modes. This makes it possible for the UPDI to access the ACC layer, also if the system is in sleep mode.

Writing a zero to this bit will lower the clock request.

This bit will be reset when the UPDI is disabled.

This bit is set by default when the UPDI is enabled in any mode (Fuse, 12V).

### 32.5.9 ASI System Status

**Name:** ASI_SYS_STATUS
**Offset:** 0x0B
**Reset:** 0x01
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | RSTSYS | INSLEEP | NVMPROG | UROWPROG | | LOCKSTATUS |
| Access | | | R | R | R | R | | R |
| Reset | | | 0 | 0 | 0 | 0 | | 1 |

**Bit 5 – RSTSYS: System Reset Active**
If this bit is set, there is an active Reset on the system domain. If this bit is cleared, the system is not in Reset.

This bit is cleared on read.

A reset held from the ASI_RESET_REQ register will also affect this bit.

**Bit 4 – INSLEEP: System Domain in Sleep**
If this bit is set, the system domain is in IDLE or deeper sleep mode. If this bit is cleared, the system is not in sleep.

**Bit 3 – NVMPROG: Start NVM Programming**
If this bit is set, NVM Programming can start from the UPDI.

When the UPDI is done, it must reset the system through the UPDI Reset Register.

**Bit 2 – UROWPROG : Start User Row Programming**
If this bit is set, User Row Programming can start from the UPDI.

When the UPDI is done, it must write the UROWWRITE_FINAL bit in ASI_SYS_CTRLA.

**Bit 0 – LOCKSTATUS: NVM Lock Status**
If this bit is set, the device is locked. If a Chiperase is done, and the Lockbits are cleared, this bit will read as zero.

### 32.5.10 ASI CRC Status

**Name:** ASI_CRC_STATUS
**Offset:** 0x0C
**Reset:** 0x00
**Property:** -

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

**Bits 2:0 – CRC_STATUS[2:0]: CRC Execution Status**

These bits signalize the status of the CRC conversion. The bits are one-hot encoded.

| Value | Description |
|---|---|
| 0x0 | Not enabled |
| 0x1 | CRC enabled, busy |
| 0x2 | CRC enabled, done with OK signature |
| 0x4 | CRC enabled, done with FAILED signature |
| Other | Reserved |

# 33. Electrical Characteristics

## 33.1 Disclaimer

All typical values are measured at T = 25°C and $V_{DD}$=3V unless otherwise specified. All minimum and maximum values are valid across operating temperature and voltage unless otherwise specified.

## 33.2 Absolute Maximum Ratings

Stresses beyond those listed in this section 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.

**Table 33-1. Absolute Maximum Ratings**

| Symbol | Description | Conditions | Min. | Max. | Unit |
|---|---|---|---|---|---|
| $V_{DD}$ | Power Supply Voltage | | -0.5 | 6 | V |
| $I_{VDD}$ | Current into a $V_{DD}$ pin | T=[-40, 85]°C | - | 200 | mA |
| | | T=[85, 125]°C | - | 100 | mA |
| $I_{GND}$ | Current out of a GND pin | T=[-40, 85]°C | - | 200 | mA |
| | | T=[85, 125]°C | - | 100 | mA |
| $V_{RST}$ | RESET pin voltage with respect to GND | | -0.5 | 13 | V |
| $V_{PIN}$ | Pin voltage with respect to GND | | -0.5 | $V_{DD}$+0.5 | V |
| $I_{PIN}$ | I/O pin sink/source current | | -40 | 40 | mA |
| $I_{c1}$(1) | I/O pin injection current except RESET pin | $V_{pin}$<GND-0.6V or 5.5V<$V_{pin}$<=6.1V 4.9V<$V_{DD}$<=5.5V | -1 | 1 | mA |
| $I_{c2}$(1) | I/O pin injection current except RESET pin | $V_{pin}$<GND-0.6V or $V_{pin}$<=5.5V $V_{DD}$<=4.9V | -15 | 15 | mA |
| $I_{ctot}$ | Sum of I/O pin injection current except RESET pin | | -45 | 45 | mA |
| $T_{storage}$ | Storage temperature | | -65 | 150 | °C |

**Note:**

1. – If Vpin is lower than GND-0.6V, then a current limiting resistor is required. The negative DC injection current limiting resistor is calculated as R = (GND-0.6V – $V_{pin}$)/$I_{Cn}$.
   – If Vpin is greater than VDD+0.6V, then a current limiting resistor is required. The positive DC injection current limiting resistor is calculated as R = ($V_{pin}$-(VDD+0.6))/$I_{Cn}$.

© 2017 Microchip Technology Inc.

## 33.3 General Operating Ratings

The device must operate within the ratings listed in this section in order for all other electrical characteristics and typical characteristics of the device to be valid.

**Table 33-2. General Operating Conditions**

| Symbol | Description | Condition | Min. | Max. | Unit |
|---|---|---|---|---|---|
| $V_{DD}$ | Operating Supply Voltage | | 1.8(2) | 5.5 | V |
| T | Operating temperature range(1) | Standard temperature range | -40 | 105 | °C |
| | | Extended temperature range(3) | -40 | 125 | |

**Note:**

1. Please refer to the device ordering code for the device temperature range.
2. Operation guaranteed down to 1.8V or VBOD with BODLEVEL=1.8V, whichever is lower.
3. Extended temperature range is only guaranteed down to 2.7V.

**Table 33-3. Operating Voltage and Frequency**

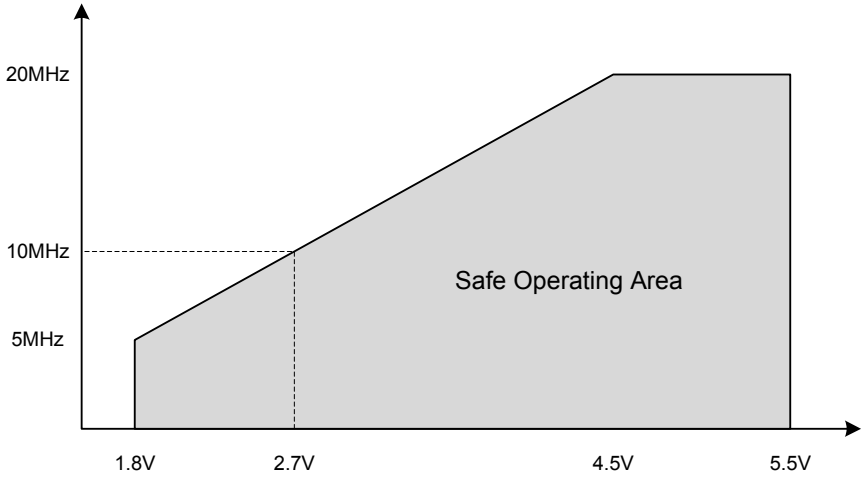
| Symbol | Description | Condition | Min. | Max. | Unit |
|---|---|---|---|---|---|
| CLK_CPU | Operating system clock frequency | $V_{DD}$=[1.8, 5.5]V T=[-40, 105]°C(1) | 0 | 5 | MHz |
| | | $V_{DD}$=[2.7, 5.5]V T=[-40, 105]°C(2) | 0 | 10 | |
| | | $V_{DD}$=[4.5, 5.5]V T=[-40, 105]°C(3) | 0 | 20 | |
| | | $V_{DD}$=[2.7, 5.5]V T=[-40, 125]°C(2) | 0 | 8 | |
| | | $V_{DD}$=[4.5, 5.5]V T=[-40, 125]°C(3) | 0 | 16 | |

**Note:**

1. Operation guaranteed down to BOD triggering level, $V_{BOD}$ with BODLEVEL0.
2. Operation guaranteed down to BOD triggering level, $V_{BOD}$ with BODLEVEL2.
3. Operation guaranteed down to BOD triggering level, $V_{BOD}$ with BODLEVEL7.

The maximum CPU clock frequency depends on $V_{DD}$. As shown in the following figure, the Maximum Frequency vs. $V_{DD}$ is linear between $1.8V < V_{DD} < 2.7V$ and $2.7V < V_{DD} < 4.5V$

**Figure 33-1. Maximum Frequency vs. $V_{DD}$ for [-40, 105]°C**




## 33.4 Power Consumption

The values are measured power consumption under the following conditions, except where noted:

- $V_{DD}$=3V
- T=25°C
- OSC20M used as system clock source, except where otherwise specified
- System power consumption measured with peripherals disabled and without I/O drive.

**Table 33-4. Power consumption in Active and Idle mode**

| Mode | Description | Condition | | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| Active | Active power consumption | CLK_CPU=20MHz (OSC20M) | $V_{DD}$=5V | 9.0 | - | mA |
| | | CLK_CPU=10MHz (OSC20M div2) | $V_{DD}$=5V | 4.8 | - | mA |
| | | | $V_{DD}$=3V | 2.7 | - | mA |
| | | CLK_CPU=5MHz (OSC20M div4) | $V_{DD}$=5V | 2.8 | - | mA |
| | | | $V_{DD}$=3V | 1.6 | - | mA |
| | | | $V_{DD}$=2V | 1.0 | - | mA |
| | | CLK_CPU=32KHz (OSCULP32K) | $V_{DD}$=5V | 18 | - | µA |
| | | | $V_{DD}$=3V | 10 | - | µA |
| | | | $V_{DD}$=2V | 7 | - | µA |
| Idle | Idle power consumption | CLK_CPU=20MHz (OSC20M) | $V_{DD}$=5V | 2.8 | - | mA |
| | | CLK_CPU=10MHz (OSC20M div2) | $V_{DD}$=5V | 1.7 | - | mA |
| | | | $V_{DD}$=3V | 0.9 | - | mA |
| | | CLK_CPU=5MHz (OSC20M div4) | $V_{DD}$=5V | 1.2 | - | mA |
| | | | $V_{DD}$=3V | 0.6 | - | mA |
| | | | $V_{DD}$=2V | 0.4 | - | mA |

| Mode | Description | Condition | | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| | | CLK_CPU=32KHz (OSCULP32K) | $V_{DD}$=5V | 5.4 | - | µA |
| | | | $V_{DD}$=3V | 2.6 | - | µA |
| | | | $V_{DD}$=2V | 1.7 | - | µA |

**Table 33-5. Power consumption in Power-down, Standby and Reset mode**

| Mode | Description | Condition | | Typ. 25°C | Max. 85°C | Max. 125°C | Unit |
|---|---|---|---|---|---|---|---|
| Standby | Standby power consumption | RTC running at 1.024kHz from external XOSC32K (CL=7.5pF) | $V_{DD}$=3V | 0.69 | - | - | µA |
| | | RTC running at 1.024kHz from internal OSCULP32K | $V_{DD}$=3V | 0.71 | 6.0 | 8.0 | µA |
| Power Down/ Standby | Power down/Standby power consumption are the same when all peripherals are stopped | All peripherals stopped | $V_{DD}$=3V | 0.1 | 5.0 | 7.0 | µA |
| Reset | Reset power consumption | Reset line pulled down | $V_{DD}$=3V | 100 | - | - | µA |

## 33.5 Wake-Up Time

Wake-up time from sleep mode is measured from the edge of the wake-up signal to the first instruction executed.

Operating conditions:

- $V_{DD}$=3V
- T=25°C
- OSC20M as system clock source, unless otherwise specified.

**Table 33-6. Start-up, Reset, and Wake-Up Time from OSC20M**

| Symbol | Description | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $t_{wakeup}$ | Start-up time from any Reset release | | - | 200 | - | µs |
| | Wake-up from Idle mode | OSC20M @ 20MHz; $V_{DD}$=5V | - | 1 | - | |
| | | OSC20M @ 10MHz; $V_{DD}$=3V | - | 2 | - | |

| Symbol | Description | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| | | OSC20M @ 5MHz; $V_{DD}$=2V | - | 4 | - | |
| | Wake-up from Standby and Power Down mode | | - | 10 | - | |

## 33.6 Peripherals Power Consumption

The table below can be used to calculate the additional current consumption for the different I/O peripherals in the various operating modes.

Operating conditions:

- $V_{DD}$=3V
- T=25°C
- OSC20M at 1MHz used as system clock source, except where otherwise specified.

**Table 33-7. Peripherals Power Consumption**

| Peripheral | Conditions | Typ.(1) | Unit |
|---|---|---|---|
| BOD | Continuous | 19 | µA |
| | Sampling @ 1kHz | 1.2 | |
| TCA | 16bit count @ 1MHz | 12.6 | µA |
| TCB | 16bit count @ 1MHz | 7.4 | µA |
| RTC | 16bit count @ OSCULP32K | 1.2 | µA |
| WDT (including OSCULP32K) | | 0.7 | µA |
| OSC20M | | 125 | µA |
| AC | Fast Mode(2) | 92 | µA |
| | Low Power Mode(2) | 45 | µA |
| ADC | 50ksps | 325 | µA |
| | 100ksps | 340 | µA |
| XOSC32K | $C_L$=7.5pF | 0.5 | µA |
| OSCULP32K | | 0.4 | µA |
| USART | Enable @ 9600 Baud | 13 | µA |
| SPI (Master) | Enable @ 100kHz | 2.1 | µA |
| TWI (Master) | Enable @ 100kHz | 23.9 | µA |
| TWI (Slave) | Enable @ 100kHz | 17.1 | µA |
| Flash programming | Erase Operation | 1.5 | mA |
| | Write Operation | 3.0 | |

**Note:**

1. Current consumption of the module only. To calculate the total power consumption of the system, add this value to the base power consumption as listed in *Power Consumption*.
2. CPU in Standby mode.

## 33.7 BOD and POR Characteristics

**Table 33-8. Power Supply Characteristics**

| Symbol | Description | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| SRON | Power-on Slope | | - | - | 100 | V/ms |

**Table 33-9. Power On Reset (POR) Characteristics**

| Symbol | Description | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $V_{POR}$ | POR threshold voltage on $V_{DD}$ falling | $V_{DD}$ falls/rises at 0.5V/ms or slower | 0.8 | - | 1.6 | V |
| | POR threshold voltage on $V_{DD}$ rising | | 1.4 | - | 1.8 | |

**Table 33-10. Brownout Detection (BOD) Characteristics**

| Symbol | Description | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $V_{BOD}$ | BOD triggering level (falling/rising) | BODLEVEL7 | 3.9 | 4.2 | 4.5 | V |
| | | BODLEVEL2 | 2.4 | 2.6 | 2.9 | |
| | | BODLEVEL0 | 1.7 | 1.8 | 2.0 | |
| $V_{INT}$ | Interrupt level 0 | Percentage above the selected BOD level | - | 4 | - | % |
| | Interrupt level 1 | | - | 13 | - | |
| | Interrupt level 2 | | - | 25 | - | |
| $V_{HYS}$ | Hysteresis | BODLEVEL7 | - | 80 | - | mV |
| | | BODLEVEL2 | - | 40 | - | |
| | | BODLEVEL0 | - | 25 | - | |
| $T_{BOD}$ | Detection time | Continuous | - | 7 | - | µs |
| | | Sampled, 1kHz | - | 1 | - | ms |
| | | Sampled, 125Hz | - | 8 | - | |
| $T_{start}$ | Start-up time | Time from enable to ready | - | 40 | - | µs |

## 33.8 External Reset Characteristics

**Table 33-11. External Reset Characteristics**

| Mode | Description | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $V_{HVRST}$ | $\overline{RESET}$ pin threshold for High-Voltage Reset | | 11.5 | - | 12.5 | V |
| $V_{RST_VIH}$ | Input High Voltage for $\overline{RESET}$ | | 0.8×$V_{DD}$ | - | $V_{DD}$+0.2 | |
| $V_{RST_VIL}$ | Input Low Voltage for $\overline{RESET}$ | | -0.2 | - | 0.2×$V_{DD}$ | |
| $t_{RST}$ | Minimum pulse width on $\overline{RESET}$ pin | | 2.5 | - | - | µs |
| $R_{RST}$ | $\overline{RESET}$ pull-up resistor | $V_{Reset}$=0V | 20 | - | 60 | kΩ |

## 33.9 Oscillators and Clocks

Operating conditions:

- $V_{DD}$=3V, except where specified otherwise.

**Table 33-12. 20MHz Internal Oscillator (OSC20M) Characteristics**

| Symbol | Description | Condition | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| $f_{OSC20M}$ | Accuracy with 16MHz and 20MHz Frequency Selection relative to the factory-stored frequency value | Factory calibrated $V_{DD}$=3V(1) | T=[0, 70]°C, $V_{DD}$=[1.8, 4.5]V(3) | -2.0 | - | 2.0 | % |
| | | Factory calibrated $V_{DD}$=5V(1) | T=[0, 70]°C, $V_{DD}$=[4.5, 5.5]V(3) | -2.0 | - | 2.0 | |
| | Accuracy with 16MHz and 20MHZ Frequency Selection | Factory calibrated | T=25°C, 3.0V | -3.0 | - | 3.0 | % |
| | | | T=[0, 70]°C, $V_{DD}$=[1.8, 3.6]V(3) | -4.0 | - | 4.0 | |
| | | | Full operation range(3) | -5.0 | - | 5.0 | |
| $f_{CAL}$ | User calibration range | OSC16M(2) | | 14.5 | - | 17.5 | MHz |
| | | OSC20M(2) | | 18.5 | - | 21.5 | MHz |
| %CAL | Calibration step size | | | - | 1.5 | - | % |
| DC | Duty cycle | | | - | 50 | - | % |
| $T_{start}$ | Start-up time | Within 2% accuracy | | - | 8 | - | µs |

**Note:**

1. See also description of OSC20M on calibration.

2. Oscillator Frequencies above speed specification must be divided so that CPU clock always is within specification.
3. These values are based on characterization and not covered by production testlimits.

**Table 33-13. 32.768kHz Internal Oscillator (OSCULP32K) Characteristics**

| Symbol | Description | Condition | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| $f_{OSCULP32K}$ | Accuracy | Factory calibrated | T=25°C, 3.0V | -3 | - | 3 | % |
| | | | T=[0, 70]°C, $V_{DD}$=[1.8, 3.6]V(1) | -10 | - | +10 | |
| | | | Full operation range(1) | -30 | - | +30 | |
| DC | Duty cycle | | | - | 50 | - | % |
| $T_{start}$ | Start-up time | | | - | 250 | - | µs |

**Note:**

1. These values are based on characterization and not covered by production testlimits.

**Table 33-14. 32.768kHz External Crystal Oscillator (XOSC32K) Characteristics**

| Symbol | Description | Condition | Min. | typ | Max. | Unit |
|---|---|---|---|---|---|---|
| $F_{out}$ | Frequency | | - | 32.768 | - | kHz |
| $T_{start}$ | Startup time | $C_L$=7.5pF | - | 300 | - | ms |
| $C_L$ | Crystal load capacitance | | 7.5 | - | 12.5 | pF |
| $C_{TOSC1}$ | Parasitic capacitor load | | - | 5.5 | - | pF |
| $C_{TOSC2}$ | | | - | 5.5 | - | pF |
| ESR | Equivalent Series Resistance - Safety Factor=3 | $C_L$=7.5pF | - | - | 80 | kΩ |
| | | $C_L$=12.5pF | - | - | 40 | |

**Figure 33-2. External Clock Waveform Characteristics**

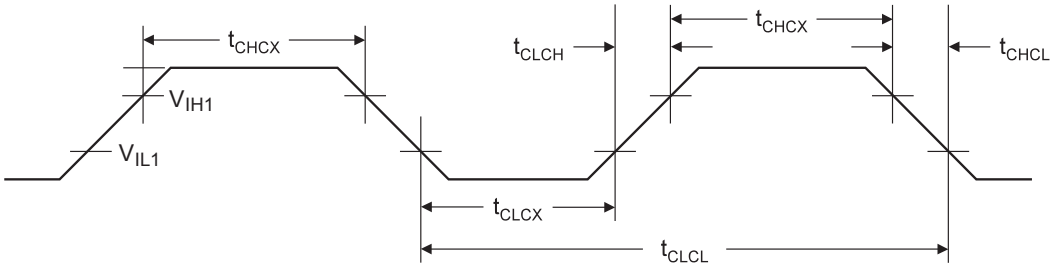


**Table 33-15. External Clock Characteristics**

| Symbol | Description | Condition | $V_{DD}$=[1.8, 5.5]V | | $V_{DD}$=[2.7, 5.5]V | | $V_{DD}$=[4.5, 5.5]V | | Unit |
|---|---|---|---|---|---|---|---|---|---|
| | | | Min. | Max. | Min. | Max. | Min. | Max. | |
| $f_{CLCL}$ | Frequency | | 0 | 5.0 | 0.0 | 10.0 | 0.0 | 20.0 | MHz |
| $t_{CLCL}$ | Clock Period | | 200 | - | 100 | - | 50 | - | ns |
| $t_{CHCX}$ | High Time | | 80 | - | 40 | - | 20 | - | ns |
| $t_{CLCX}$ | Low Time | | 80 | - | 40 | - | 20 | - | ns |

**Related Links**

16/20MHz Oscillator (OSC20M)

## 33.10 I/O Pin Characteristics

**Table 33-16. I/O Pin Characteristics ($T_A$=[-40, 105]°C, $V_{DD}$=[1.8, 5.5]V unless otherwise noted)**

| Symbol | Description | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $V_{IL}$ | Input Low Voltage, except $\overline{\text{RESET}}$ pin as I/O | | -0.2 | - | 0.3×$V_{DD}$ | V |
| $V_{IH}$ | Input High Voltage, except $\overline{\text{RESET}}$ pin as I/O | | 0.7×$V_{DD}$ | - | $V_{DD}$+0.2V | V |
| $I_{IH}$ / $I_{IL}$ | I/O pin Input Leakage Current, except $\overline{\text{RESET}}$ pin as I/O | $V_{DD}$=5.5V, Pin high | - | < 0.05 | - | µA |
| | | $V_{DD}$=5.5V, Pin low | - | < 0.05 | - | |
| $V_{OL}$ | I/O pin drive strength | $V_{DD}$=1.8V, $I_{OL}$=1.5mA | - | - | 0.36 | V |
| | | $V_{DD}$=3.0V, $I_{OL}$=7.5mA | - | - | 0.6 | |
| | | $V_{DD}$=5.0V, $I_{OL}$=15mA | - | - | 1 | |
| $V_{OH}$ | I/O pin drive strength | $V_{DD}$=1.8V, $I_{OH}$=1.5mA | 1.44 | - | - | V |
| | | $V_{DD}$=3.0V, $I_{OH}$=7.5mA | 2.4 | - | - | |
| | | $V_{DD}$=5.0V, $I_{OH}$=15mA | 4 | - | - | |
| $I_{total}$ | Maximum combined I/O sink current per pin group(1) | | - | - | 100 | mA |

| Symbol | Description | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| | Maximum combined I/O source current per pin group(1) | | - | - | 100 | |
| $V_{IL2}$ | Input Low Voltage on $\overline{\text{RESET}}$ pin as I/O | | -0.2 | - | $0.3 \times V_{DD}$ | V |
| $V_{IH2}$ | Input High Voltage on $\overline{\text{RESET}}$ pin as I/O | | $0.7 \times V_{DD}$ | - | $V_{DD}+0.2V$ | V |
| $V_{OL2}$ | I/O pin sink current on $\overline{\text{RESET}}$ pin as I/O | $V_{DD}$=1.8V, $I_{OL}$=0.1mA | - | - | 0.36 | V |
| | | $V_{DD}$=3.0V, $I_{OL}$=0.25mA | - | - | 0.6 | |
| | | $V_{DD}$=5.0V, $I_{OL}$=0.5mA | - | - | 1 | |
| $V_{OH2}$ | I/O pin source current on $\overline{\text{RESET}}$ pin as I/O | $V_{DD}$=1.8V, $I_{OH}$=0.1mA | 1.44 | - | - | V |
| | | $V_{DD}$=3.0V, $I_{OH}$=0.25mA | 2.4 | - | - | |
| | | $V_{DD}$=5.0V, $I_{OH}$=0.5mA | 4 | - | - | |
| $t_{RISE}$ | Rise time | $V_{DD}$=3.0V, load=20pF | - | 2.5 | - | ns |
| | | $V_{DD}$=5.0V, load=20pF | - | 1.5 | - | |
| $t_{FALL}$ | Fall time | $V_{DD}$=3.0V, load=20pF | - | 2.0 | - | ns |
| | | $V_{DD}$=5.0V, load=20pF | - | 1.3 | - | |
| $C_{pin}$ | I/O pin capacitance except TOSC and TWI pins | | - | 3 | - | pF |
| $C_{pin}$ | I/O pin capacitance on TOSC pins | | - | 5.5 | - | pF |
| $C_{pin}$ | I/O pin capacitance on TWI pins | | - | 10 | - | pF |
| $R_p$ | Pull-up resistor | | 20 | 35 | 50 | kΩ |

**Note:**

1. Pin group A (PA[7:0]), pin group B (PB[7:0]). The combined continuous sink/source current for all I/O ports should not exceed the limits.

## 33.11 USART

**Figure 33-3. USART in SPI Mode - Timing Requirements in Master Mode**

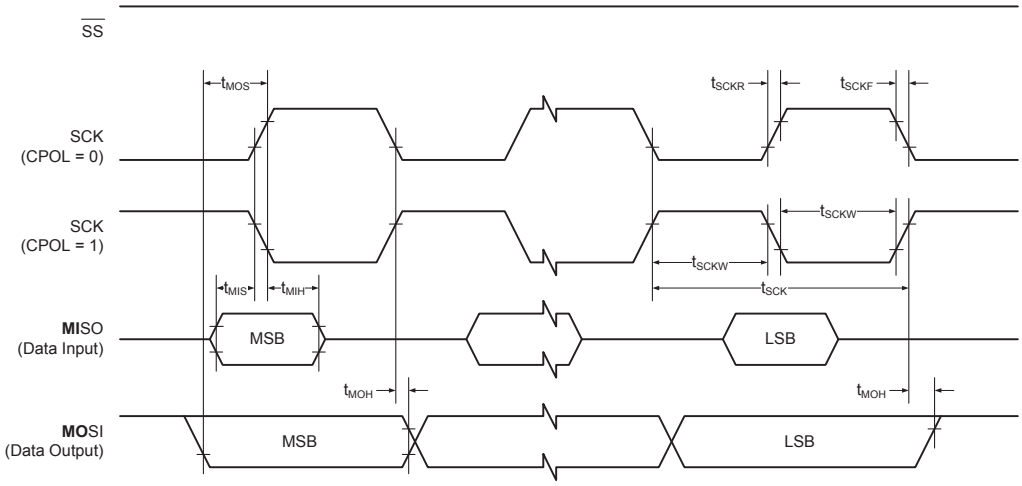



**Figure 33-4. USART in SPI Mode - Timing Requirements in Slave Mode**

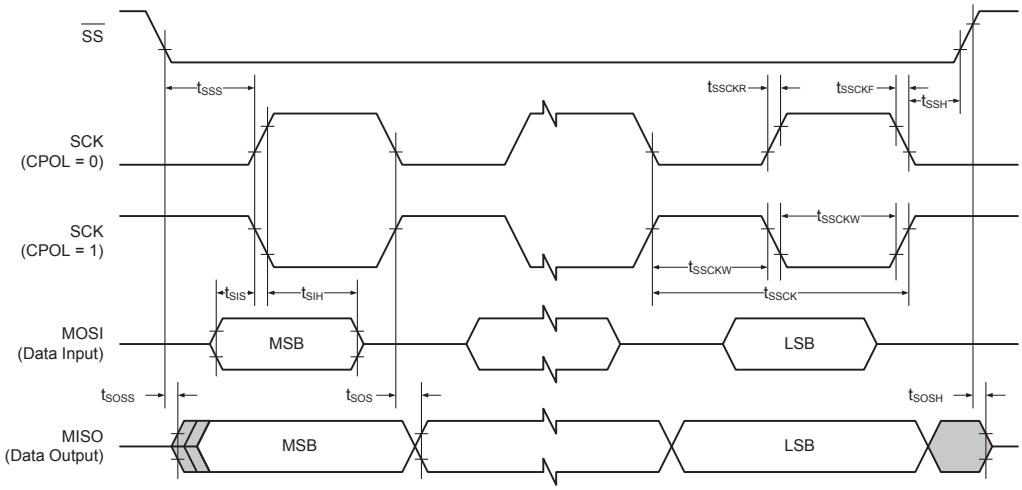



**Table 33-17. USART in SPI Mode - Timing Characteristics**

| Symbol | Description | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $f_{SCK}$ | SCK clock frequency | Master | - | - | 10 | MHz |
| $t_{SCK}$ | SCK period | Master | 100 | - | - | ns |
| $t_{SCKW}$ | SCK high/low width | Master | - | $0.5 \times t_{SCK}$ | - | ns |
| $t_{SCKR}$ | SCK rise time | Master | - | 2.7 | - | ns |
| $t_{SCKF}$ | SCK fall time | Master | - | 2.7 | - | ns |
| $t_{MIS}$ | MISO setup to SCK | Master | - | 10 | - | ns |
| $t_{MIH}$ | MISO hold after SCK | Master | - | 10 | - | ns |
| $t_{MOS}$ | MOSI setup to SCK | Master | - | $0.5 \times t_{SCK}$ | - | ns |
| $t_{MOH}$ | MOSI hold after SCK | Master | - | 1.0 | - | ns |

## 33.12 SPI

**Figure 33-5. SPI - Timing Requirements in Master Mode**

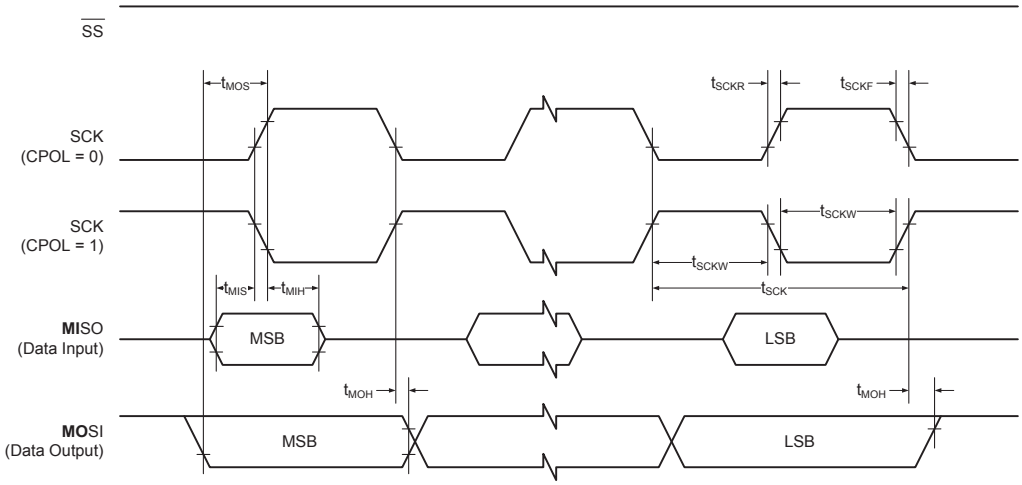



**Figure 33-6. SPI - Timing Requirements in Slave Mode**

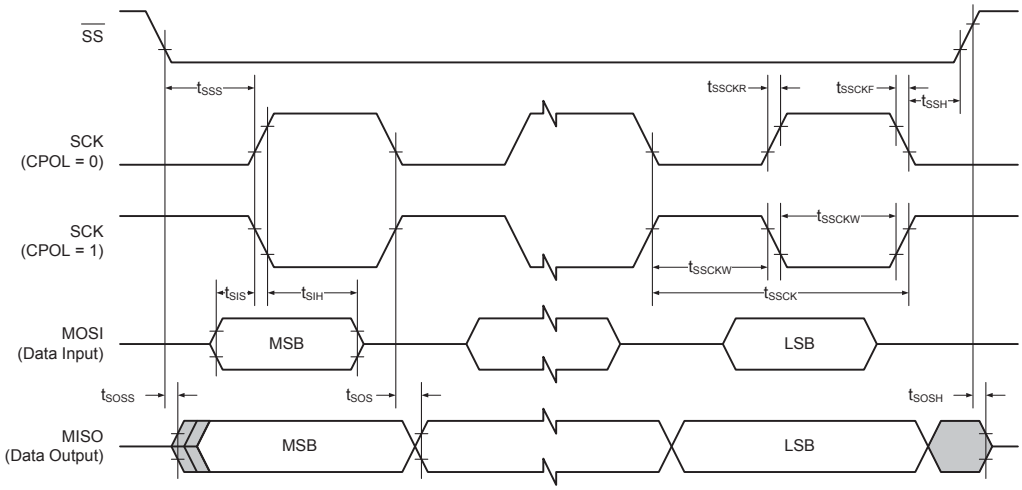



**Table 33-18. SPI - Timing Characteristics**

| Symbol | Description | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $f_{SCK}$ | SCK clock frequency | Master | - | - | 10 | MHz |
| $t_{SCK}$ | SCK period | Master | 100 | - | - | ns |
| $t_{SCKW}$ | SCK high/low width | Master | - | 0.5*SCK | - | ns |
| $t_{SCKR}$ | SCK rise time | Master | - | 2.7 | - | ns |
| $t_{SCKF}$ | SCK fall time | Master | - | 2.7 | - | ns |
| $t_{MIS}$ | MISO setup to SCK | Master | - | 10 | - | ns |
| $t_{MIH}$ | MISO hold after SCK | Master | - | 10 | - | ns |
| $t_{MOS}$ | MOSI setup to SCK | Master | - | 0.5*SCK | - | ns |
| $t_{MOH}$ | MOSI hold after SCK | Master | - | 1.0 | - | ns |

| Symbol | Description | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $f_{SSCK}$ | Slave SCK clock frequency | Slave | - | - | 5 | MHz |
| $t_{SSCK}$ | Slave SCK Period | Slave | 4*t Clkper | - | - | ns |
| $t_{SSCKW}$ | SCK high/low width | Slave | 2*t Clkper | - | - | ns |
| $t_{SSCKR}$ | SCK rise time | Slave | - | - | 1600 | ns |
| $t_{SSCKF}$ | SCK fall time | Slave | - | - | 1600 | ns |
| $t_{SIS}$ | MOSI setup to SCK | Slave | 3.0 | - | - | ns |
| $t_{SIH}$ | MOSI hold after SCK | Slave | $t_{Clkper}$ | - | - | ns |
| $t_{SSS}$ | SS setup to SCK | Slave | 21 | - | - | ns |
| $t_{SSH}$ | SS hold after SCK | Slave | 20 | - | - | ns |
| $t_{SOS}$ | MISO setup to SCK | Slave | - | 8.0 | - | ns |
| $t_{SOH}$ | MISO hold after SCK | Slave | - | 13 | - | ns |
| $t_{SOSS}$ | MISO setup after SS low | Slave | - | 11 | - | ns |
| $t_{SOSH}$ | MISO hold after SS low | Slave | - | 8.0 | - | ns |

## 33.13 TWI

**Figure 33-7. TWI - Timing Requirements**

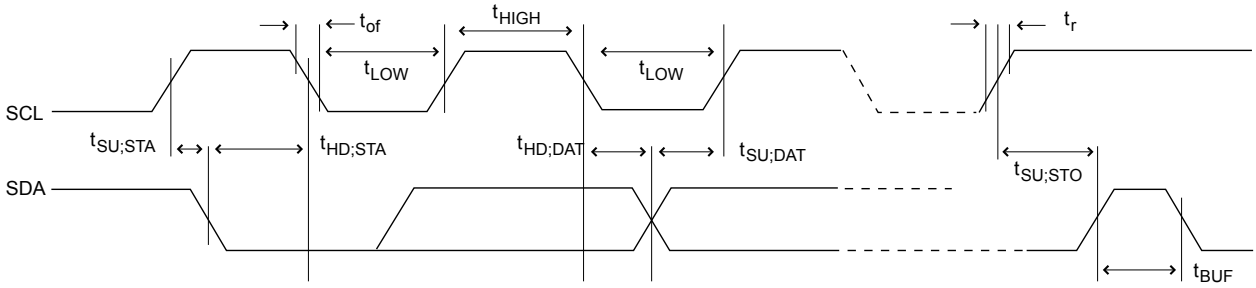



**Table 33-19. TWI - Timing Characteristics**

| Symbol | Description | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $f_{SCL}$ | SCL clock frequency | | 0 | - | 1000 | kHz |
| $V_{IH}$ | Input high voltage | | $0.7 \times V_{DD}$ | - | - | V |
| $V_{IL}$ | Input low voltage | | - | - | $0.3 \times V_{DD}$ | V |
| $V_{HYS}$ | Hysteresis of Schmitt trigger inputs | | $0.1 \times V_{DD}$ | | $0.4 \times V_{DD}$ | V |
| $V_{OL}$ | Output low voltage | $I_{load}$=20mA, Fast mode+ | - | - | $0.2V_{DD}$ | V |
| | | $I_{load}$=3mA, Normal mode, $V_{DD}$>2V | - | - | 0.4V | |

| Symbol | Description | Condition | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $I_{load}$=3mA, Normal mode, $V_{DD}$≤2V | | - | - | 0.2×$V_{DD}$ | |
| $I_{OL}$ | Low level output current | $f_{SCL}$≤400kHz, $V_{OL}$=0.4V | | 3mA | - | - | mA |
| | | $f_{SCL}$≤1MHz, $V_{OL}$=0.4V | | 20mA | - | - | |
| $C_B$ | Capacitive load for each bus line | $f_{SCL}$≤100kHz | | - | - | 400 | pF |
| | | $f_{SCL}$≤400kHz | | - | - | 400 | |
| | | $f_{SCL}$≤1MHz | | - | - | 550 | |
| $t_R$ | Rise time for both SDA and SCL | $f_{SCL}$≤100kHz | | - | - | 1000 | ns |
| | | $f_{SCL}$≤400kHz | | 20 | - | 300 | |
| | | $f_{SCL}$≤1MHz | | - | - | 120 | |
| $t_{OF}$ | Output fall time from $V_{IHmin}$ to $V_{ILmax}$ | 10pF < Capacitance of bus line < 400pF | $f_{SCL}$≤400kHz | 20+0.1×$C_B$ | - | 300 | ns |
| | | | $f_{SCL}$≤1MHz | 20+0.1×$C_B$ | - | 120 | |
| $t_{SP}$ | Spikes suppressed by Input filter | | | 0 | - | 50 | ns |
| $I_L$ | Input current for each I/O pin | 0.1×$V_{DD}$<$V_I$<0.9×$V_{DD}$ | | - | - | 1 | µA |
| $C_I$ | Capacitance for each I/O pin | | | - | - | 10 | pF |
| $R_P$ | Value of pull-up resistor | $f_{SCL}$≤100kHz | | ($V_{DD}$-$V_{OL}$(max)) /$I_{OL}$ | - | 1000ns/ (0.8473×$C_B$) | Ω |
| | | $f_{SCL}$≤400kHz | | - | - | 300ns/ (0.8473×$C_B$) | |
| | | $f_{SCL}$≤1MHz | | - | - | 120ns/ (0.8473×$C_B$) | |
| $t_{HD;STA}$ | Hold time (repeated) START condition | $f_{SCL}$≤100kHz | | 4.0 | - | - | µs |
| | | $f_{SCL}$≤400kHz | | 0.6 | - | - | |
| | | $f_{SCL}$≤1MHz | | 0.26 | - | - | |
| $t_{LOW}$ | Low period of SCL Clock | $f_{SCL}$≤100kHz | | 4.7 | - | - | µs |
| | | $f_{SCL}$≤400kHz | | 1.3 | - | - | |
| | | $f_{SCL}$≤1MHz | | 0.5 | - | - | |
| $t_{HIGH}$ | High period of SCL Clock | $f_{SCL}$≤100kHz | | 4.0 | - | - | µs |
| | | $f_{SCL}$≤400kHz | | 0.6 | - | - | |
| | | $f_{SCL}$≤1MHz | | 0.26 | - | - | |

| Symbol | Description | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $t_{SU;STA}$ | Set-up time for a repeated START condition | $f_{SCL}$≤100kHz | 4.7 | - | - | µs |
| | | $f_{SCL}$≤400kHz | 0.6 | - | - | |
| | | $f_{SCL}$≤1MHz | 0.26 | - | - | |
| $t_{HD;DAT}$ | Data hold time | $f_{SCL}$≤100kHz | 0 | - | 3.45 | µs |
| | | $f_{SCL}$≤400kHz | 0 | - | 0.9 | |
| | | $f_{SCL}$≤1MHz | 0 | - | 0.45 | |
| $t_{SU;DAT}$ | Data setup time | $f_{SCL}$≤100kHz | 250 | - | - | ns |
| | | $f_{SCL}$≤400kHz | 100 | - | - | |
| | | $f_{SCL}$≤1MHz | 50 | - | - | |
| $t_{SU;STO}$ | Setup time for STOP condition | $f_{SCL}$≤100kHz | 4 | - | - | µs |
| | | $f_{SCL}$≤400kHz | 0.6 | - | - | |
| | | $f_{SCL}$≤1MHz | 0.26 | - | - | |
| $t_{BUF}$ | Bus free time between a STOP and START condition | $f_{SCL}$≤100kHz | 4.7 | - | - | µs |
| | | $f_{SCL}$≤400kHz | 1.3 | - | - | |
| | | $f_{SCL}$≤1MHz | 0.5 | - | - | |

## 33.14 Bandgap and VREF

**Table 33-20. Internal Voltage Reference Characteristics**

| Symbol | Description | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|
| $t_{start}$ | Start-up time | - | 25 | - | µs |
| $V_{DDINT055V}$ | Power supply voltage range for INT055V | 1.8 | - | 5.5 | V |
| $V_{DDINT11V}$ | Power supply voltage range for INT11V | 1.8 | - | 5.5 | |
| $V_{DDINT15V}$ | Power supply voltage range for INT15V | 2.0 | - | 5.5 | |
| $V_{DDINT25V}$ | Power supply voltage range for INT25V | 3.0 | - | 5.5 | |
| $V_{DDINT43V}$ | Power supply voltage range for INT43V | 4.8 | - | 5.5 | |

**Table 33-21. ADC Internal Voltage Reference Characteristics(1)**

| Symbol(2) | Description | Condition | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| INT11V | Internal reference voltage | $V_{DD}$=[1.8V,3.6V]<br>T=[0-105]°C | -2.0 | | 2.0 | % |
| INT055V<br>INT15V<br>INT25V | Internal reference voltage | $V_{DD}$=[1.8V,3.6V]<br>T=[0-105]°C | -3.0 | | 3.0 | |
| INT055V<br>INT11V<br>INT15V<br>INT25V<br>INT43V | Internal reference voltage | $V_{DD}$=[1.8V,5.5V]<br>T=[-40-125]°C | -5.0 | | 5.0 | |

**Note:**

1. These values are based on characterization and not covered by production testlimits.
2. The symbols INTxxV refer to the respective values of the ADC0REFSEL and DAC0REFSEL bit fields in the VREF.CTRLA register.

**Table 33-22. DAC and AC Internal Voltage Reference Characteristics(1)**

| Symbol(2) | Description | Condition | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| INT055V<br>INT11V<br>INT15V<br>INT25V | Internal reference voltage | $V_{DD}$=[1.8V,3.6V]<br>T=[0-105]°C | -3.0 | | 3.0 | % |
| INT055V<br>INT11V<br>INT15V<br>INT25V<br>INT43V | Internal reference voltage | $V_{DD}$=[1.8V,5.5V]<br>T=[-40-125]°C | -5.0 | | 5.0 | |

**Note:**

1. These values are based on characterization and not covered by production test limits.
2. The symbols INTxxV refer to the respective values of the ADC0REFSEL and DAC0REFSEL bit fields in the VREF.CTRLA register.

## 33.15 ADC

Operating conditions:

- $V_{DD}$=5V
- Temperature 25°C
- SAMPCAP is 10pF for 0.55V reference, while it is set to 5pF for $V_{REF}$≥1.1V

**Table 33-23. Power Supply, Reference, and Input Range**

| Symbol | Description | Conditions | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $V_{DD}$ | Supply Voltage | | 1.8 | - | 5.5 | V |
| $V_{REF}$ | Reference Voltage | REFSEL = Internal reference | 0.55 | - | $V_{DD}$-0.5 | V |
| | | REFSEL = $V_{DD}$ | 1.8 | - | 5.5 | |
| $C_{IN}$ | Input capacitance | SAMPCAP=5pF | - | 5 | - | pF |
| | | SAMPCAP=10pF | - | 10 | - | |
| $V_{IN}$ | Input Voltage Range | | 0 | - | $V_{REF}$ | V |
| $I_{BAND}$ | Input Bandwidth | 1.1V≤$V_{REF}$ | - | - | 57.5 | kHz |

**Table 33-24. Clock and Timing Characteristics**

| Symbol | Description | Conditions | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $f_{ADC}$ | Sample Rate | 1.1V≤$V_{REF}$ | 15 | - | 115 | ksps |
| | | 1.1V≤$V_{REF}$ (8 bit resolution) | 15 | - | 150 | |
| | | $V_{REF}$=0.55V (10 bit) | 7.5 | - | 20 | |
| $CLK_{ADC}$ | Clock frequency | $V_{REF}$=0.55V (10 bit) | 100 | - | 260 | kHz |
| | | 1.1V≤$V_{REF}$ (10 bit) | 200 | - | 1500 | |
| | | 1.1V≤$V_{REF}$ (8 bit resolution) | 200 | - | 2000 | |
| Ts | Sampling time | | 2 | 2 | 33 | $CLK_{ADC}$ cycles |
| $T_{CONV}$ | Conversion time (latency) | Sampling time = 2$CLK_{ADC}$ | 8.7 | - | 50 | µs |
| $T_{START}$ | Start-up time | Internal $V_{REF}$ | - | 22 | - | µs |

**Table 33-25. Accuracy Characteristics**

| Symbol | Description | Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| Res | Resolution | | | - | 10 | - | bit |
| INL | Integral Non Linearity | REFSEL = Internal reference or $V_{DD}$ | 0.55V≤$V_{REF}$≤$V_{DD}$ $f_{ADC}$=15ksps | - | 1.0 | - | LSB |
| | | | 1.1V≤$V_{REF}$≤2.5 $f_{ADC}$=115ksps | - | 1.0 | - | |
| | | | 2.5V<$V_{REF}$≤$V_{DD}$ $f_{ADC}$=115ksps | - | 1.3 | - | |
| DNL[1] | Dynamic Non Linearity | REFSEL = Internal reference or $V_{DD}$ | $V_{REF}$=0.55V $f_{ADC}$=15ksps | - | 0.4 | - | LSB |
| | | | 1.1V≤$V_{REF}$≤$V_{DD}$ $f_{ADC}$=15ksps | - | 0.3 | - | |

| Symbol | Description | Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | | 1.1V≤$V_{REF}$≤2.5V<br>$f_{ADC}$=115ksps | - | 0.4 | - | |
| | | | 2.5V≤$V_{REF}$≤$V_{DD}$<br>$f_{ADC}$=115ksps | - | 0.7 | - | |
| EABS | Absolute Accuracy | REFSEL = Internal reference or $V_{DD}$ | 0.55V≤$V_{REF}$ ≤1.1V<br>$f_{ADC}$=15ksps | - | 3 | - | LSB |
| | | | $V_{REF}$ = 1.1V<br>$f_{ADC}$≤115ksps | - | 3 | - | |
| | | | 1.5V≤$V_{REF}$ ≤4.3V<br>$f_{ADC}$≤115ksps | - | 2.5 | - | |
| | | | $V_{REF}$=$V_{DD}$<br>$f_{ADC}$≤115ksps | - | 2 | - | |
| EGAIN | Gain Error | REFSEL = Internal reference or $V_{DD}$ | $V_{REF}$=1.1V<br>$f_{ADC}$ = 115ksps | - | 2±2.5 | - | LSB |
| | | | $V_{REF}$=$V_{DD}$<br>$f_{ADC}$ = 115ksps | - | 2 | - | |
| EOFF | Offset Error | REFSEL = Internal reference or $V_{DD}$ | | - | -0.5 | - | LSB |

**Note:**

1. A DNL error of less than or equal to 1LSB guarantees a monotonic transfer function with no missing codes.

## 33.16 DAC

$V_{DD}$=3V, unless stated otherwise.

**Table 33-26. Power Supply, Reference and Input Range**

| Symbol | Description | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|
| $V_{DD}$ | Supply Voltage(1) | 1.8 | 3 | 5.5 | V |
| $R_{Load}$ | Resistive External Load | 5 | - | - | kΩ |
| $C_{Load}$ | Capacitive External Load | - | - | 30 | pF |
| $V_{OUT}$ | Output Voltage Range | 0.2 | - | $V_{DD}$-0.2V | V |
| $I_{OUT}$ | Output sink/source | - | 1 | - | mA |

**Note:** 1. Supply voltage must meet the $V_{DD}$ specification for the $V_{REF}$ level used as DAC reference.

**Table 33-27. Clock and Timing Characteristics**

| Symbol | Description | Conditions | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $f_{DAC}$ | Maximum Conversion Rate | 0.55V≤ $V_{REF}$≤2.5V | - | 350 | - | ksps |
| | | $V_{REF}$=4.3 V | - | 270 | - | ksps |

**Table 33-28. Accuracy Characteristics**

| Symbol | Description | Conditions | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| Res | Resolution | | - | | 8 | bits |
| INL | Integral Non-Linearity | 0.55V≤$V_{REF}$≤4.3 | - | 0.3 | - | LSB |
| DNL | Differential Non-Linearity | 0.55V≤$V_{REF}$≤4.3 | - | 0.25 | - | LSB |
| EOFF | Offset Error | $V_{REF}$ =1.1V $V_{CC}$ = 3.0 T=25C | - | ±0.25 | - | LSB |
| EGAIN | Gain Error | $V_{REF}$=1.1V $V_{CC}$ = 3.0 T=25C | - | ±1 | - | LSB |

## 33.17 AC

**Table 33-29. Analog Comparator Characteristics**

| Symbol | Description | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $V_{IN}$ | Input Voltage | Low Power Mode Enabled | 0 | - | $V_{DD}$ | V |
| | | Low Power Mode Disabled | -0.2 | - | $V_{DD}$ | |
| $C_{IN}$ | Input Pin Capacitance | PA6 | - | 9 | - | pF |
| | | PA7, PB5, PB4 | - | 5 | - | |
| $V_{OFF}$ | Input Offset Voltage, Low Power Mode enabled | $V_{IN}$ = $V_{DD}$/2 | -25 | <10 | 25 | mV |
| | | $V_{IN}$=[0V, $V_{DD}$] | - | <30 | - | |
| | Input Offset Voltage, Low Power Mode disabled | $V_{IN}$=$V_{DD}$/2 | -20 | <5 | 20 | |
| | | $V_{IN}$=[-0.2V, $V_{DD}$] | - | <20 | - | |
| $I_L$ | Input Leakage Current | | - | 5 | - | nA |
| $T_{START}$ | Start-up Time | | - | 1.3 | - | µs |
| $V_{HYS}$ | Hysteresis, Low Power Mode disabled | HYSMODE=0x0 | - | 0 | - | mV |
| | | HYSMODE=0x1 | - | 10 | - | |
| | | HYSMODE=0x2 | - | 25 | - | |
| | | HYSMODE=0x3 | - | 50 | - | |
| $t_{PD}$ | Propagation Delay | 25mV Overdrive, $V_{DD}$≥2.7V, Low Power Mode disabled | - | 50 | - | ns |
| | | 25mV Overdrive, $V_{DD}$≥2.7V, Low Power Mode enabled | - | 150 | - | |

© 2017 Microchip Technology Inc.

## 33.18 Programming Time

See the following table for typical programming times for Flash and EEPROM.

**Table 33-30. Programming Times**

| Symbol | Typ. Programming Time |
|---|---|
| Page Buffer Clear | 7 CLK_CPU cycles |
| Page Write | 2ms |
| Page Erase | 2ms |
| Page Erase-Write | 4ms |
| Chip Erase | 4ms |
| EEPROM Erase | 4ms |

© 2017 Microchip Technology Inc.

# 34. Typical Characteristics

## 34.1 Power Consumption

### 34.1.1 Supply Currents in Active Mode

**Figure 34-1. Active Supply Current vs. Frequency (1-20MHz) at T=25°C**

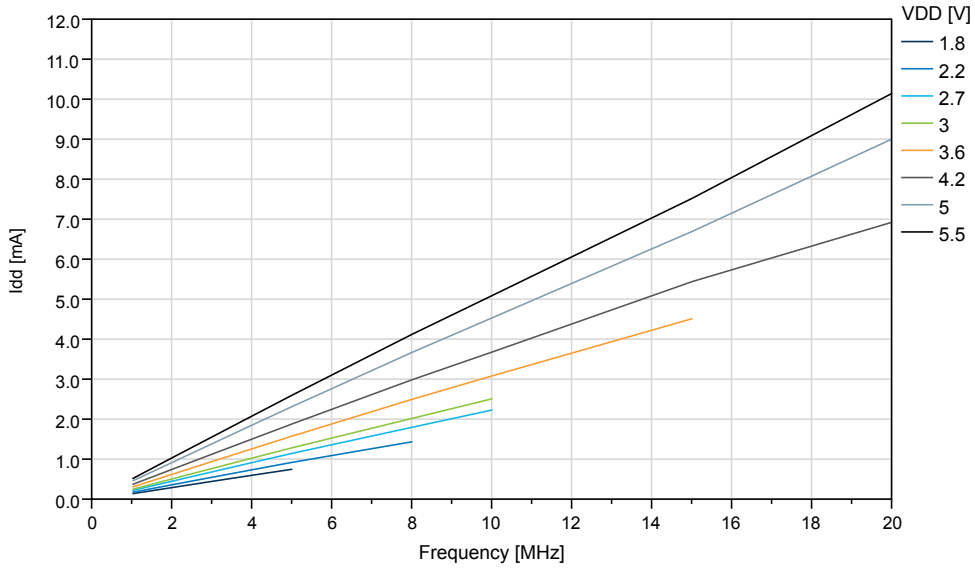



**Figure 34-2. Active Supply Current vs. Frequency [0.1, 1.0]MHz at T=25°C**

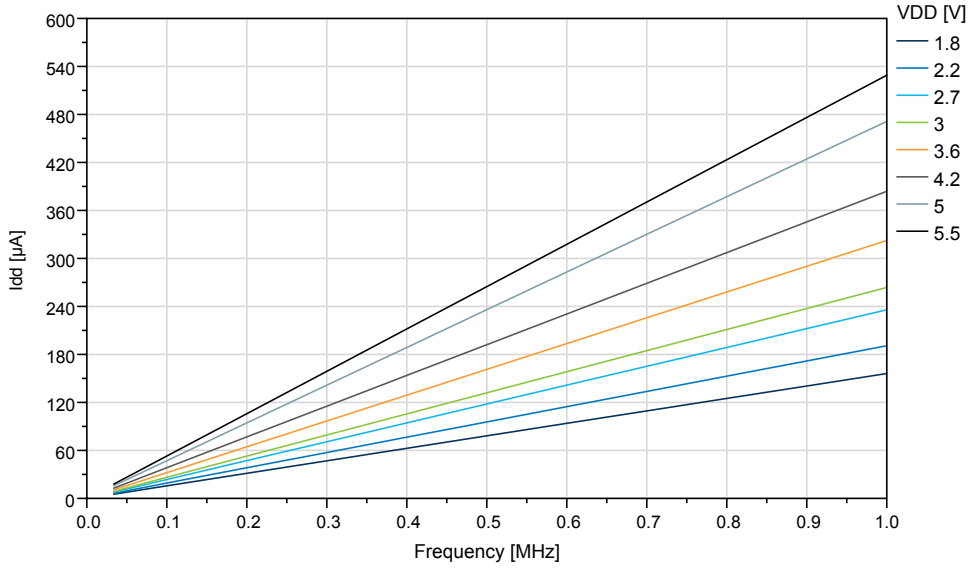


**Figure 34-3. Active Supply Current vs. Temperature (f=20MHz OSC20M)**

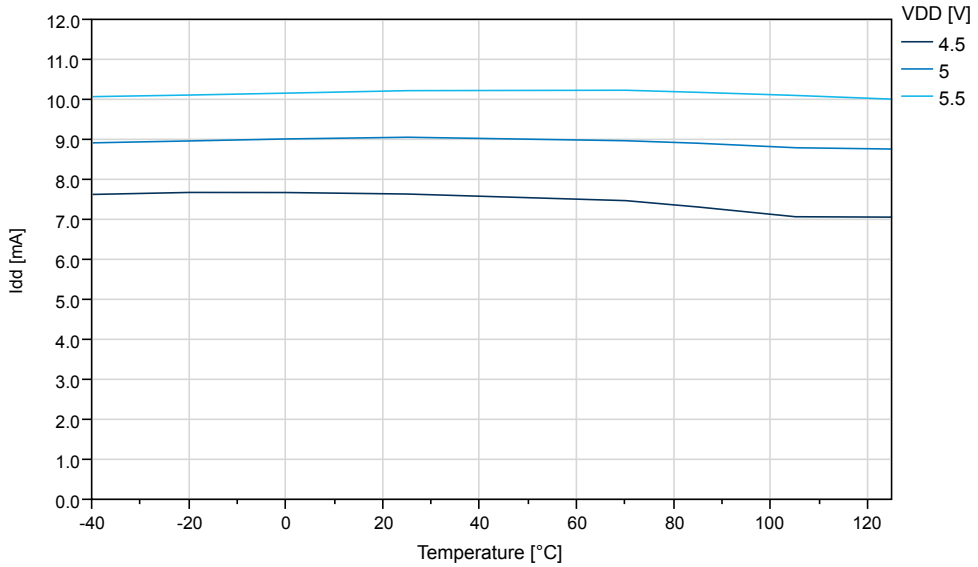



**Figure 34-4. Active Supply Current vs. $V_{DD}$ (f=[1.25, 20]MHz OSC20M) at T=25°C**

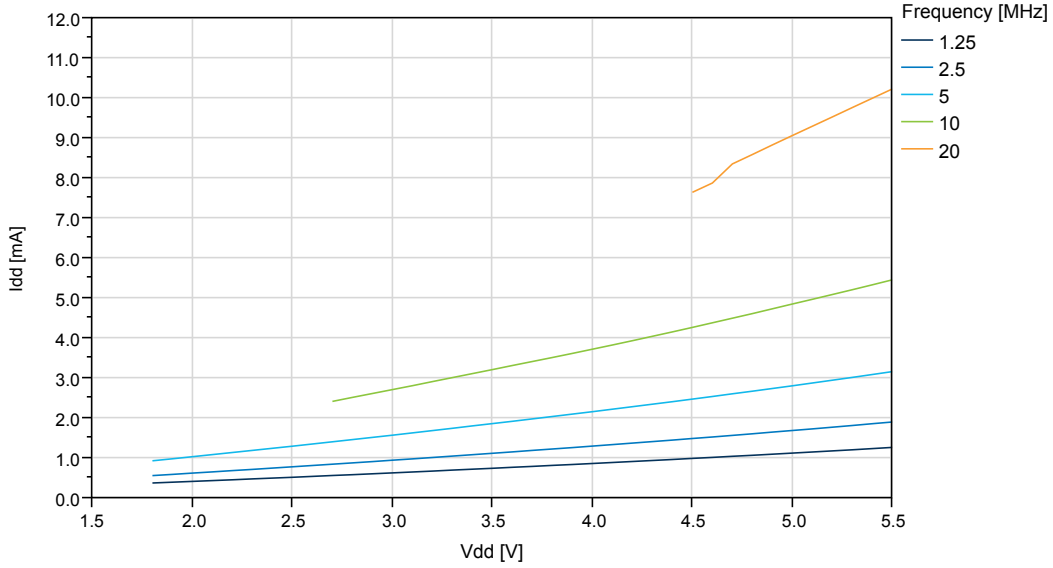
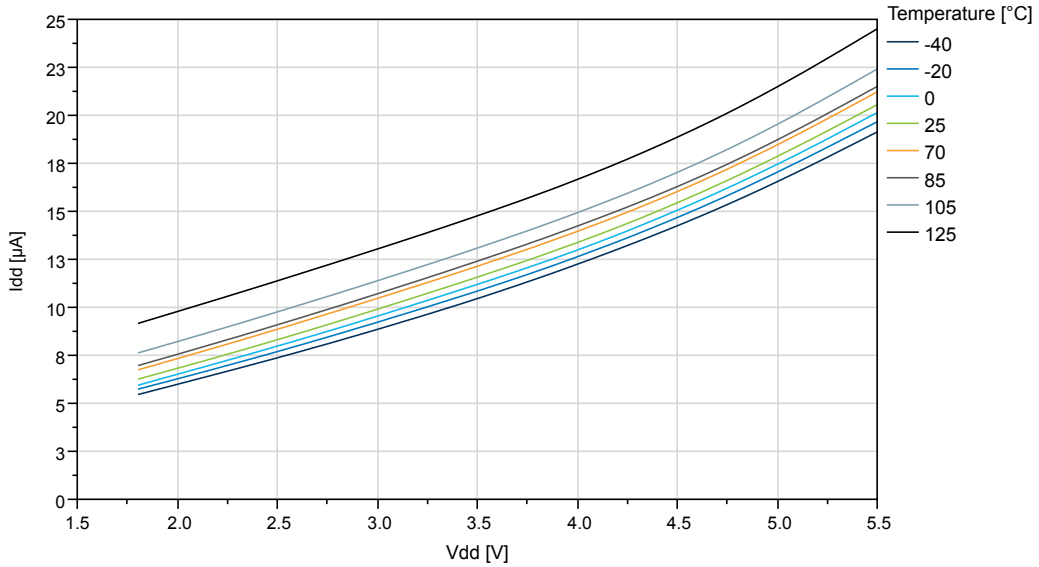


**Figure 34-5. Active Supply Current vs. $V_{DD}$ (f=32KHz OSCULP32K)**




### 34.1.2 Supply Currents in Idle Mode

**Figure 34-6. Idle Supply Current vs. Frequency (1-20MHz) at T=25°C**

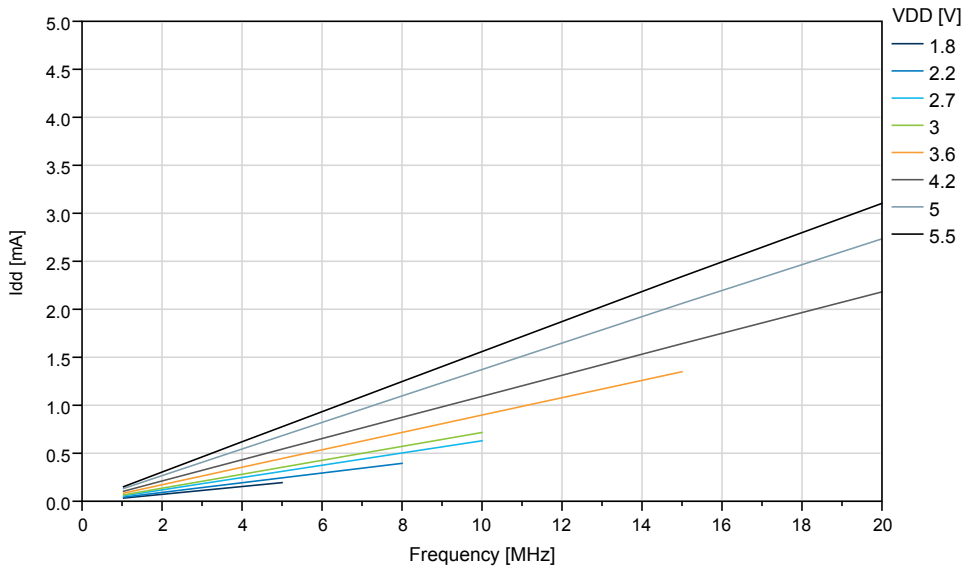


**Figure 34-7. Idle Supply Current vs. Low Frequency (0.1-1.0MHz) at T=25°C**

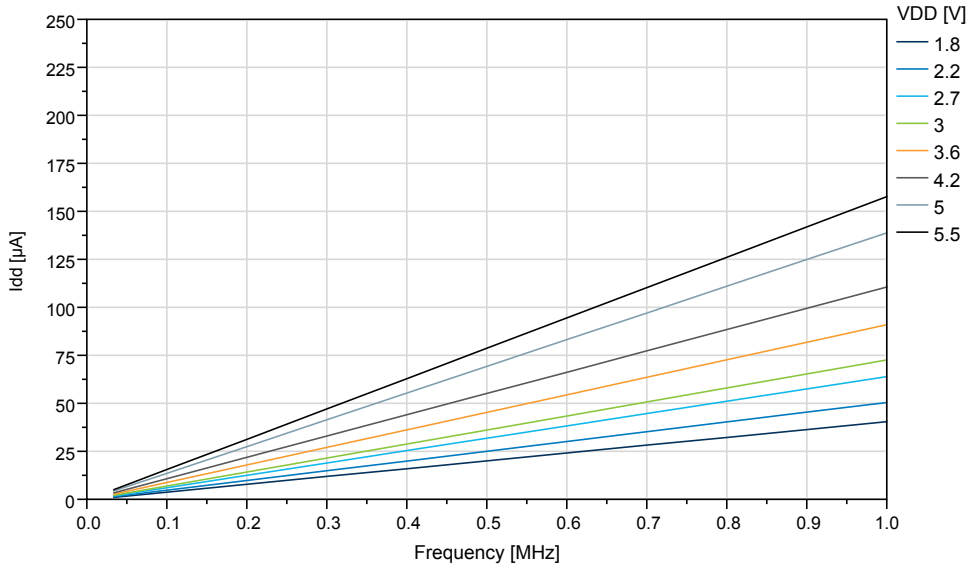



**Figure 34-8. Idle Supply Current vs. Temperature (f=20MHz OSC20M)**

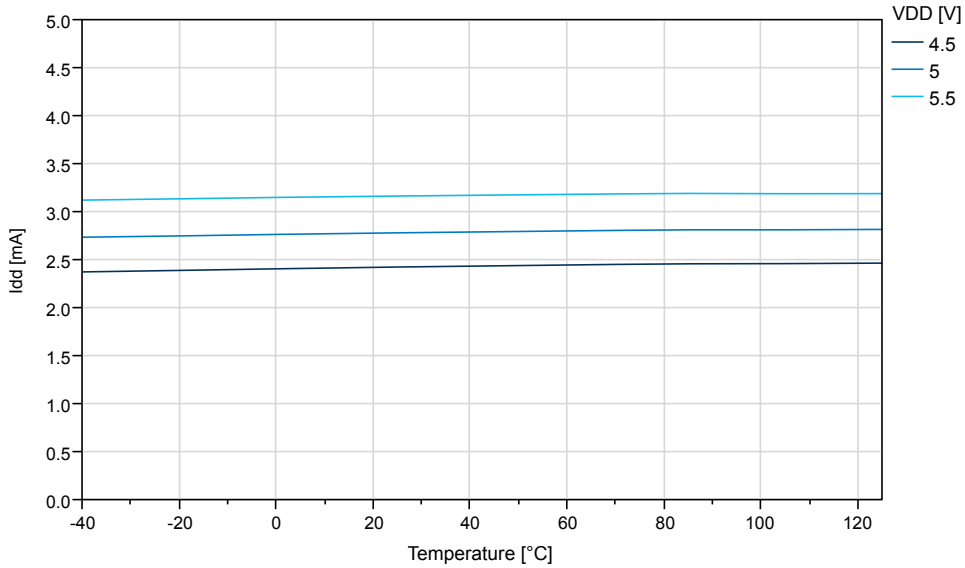
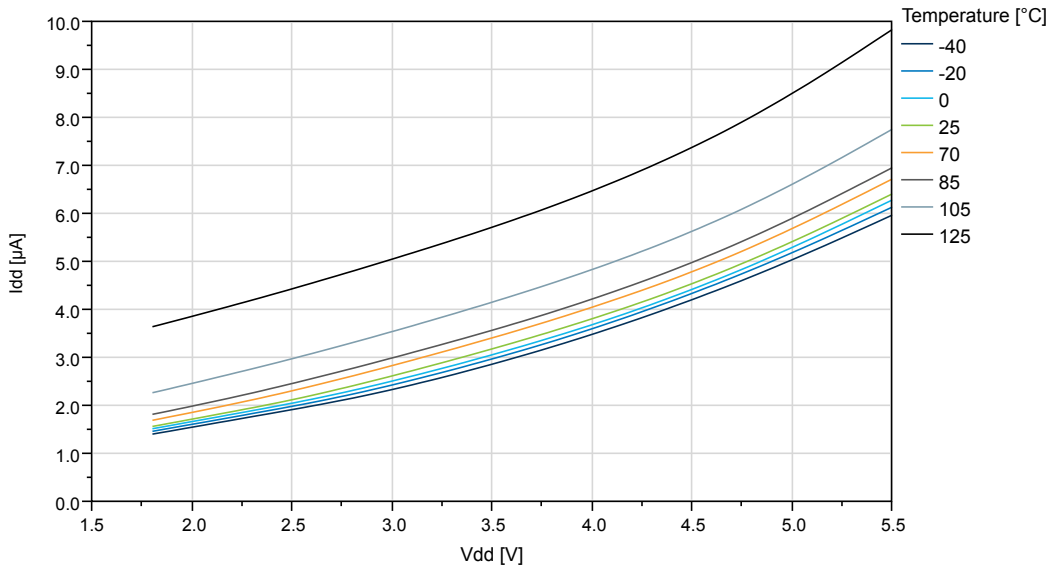


**Figure 34-9. Idle Supply Current vs. $V_{DD}$ (f=32KHz OSCULP32K)**




### 34.1.3 Supply Currents in Power Down Mode

**Figure 34-10. Power Down Mode Supply Current vs. Temperature (all functions disabled)**

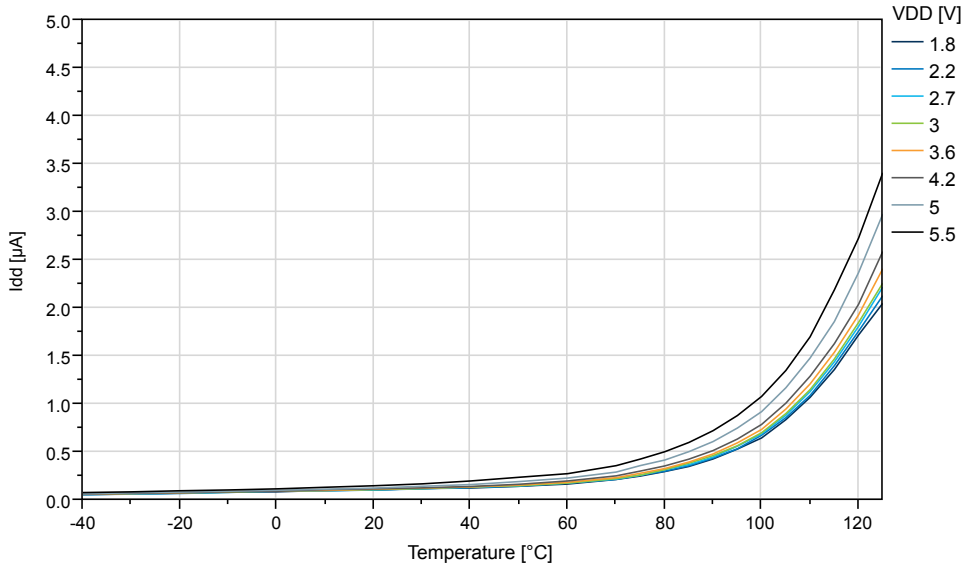
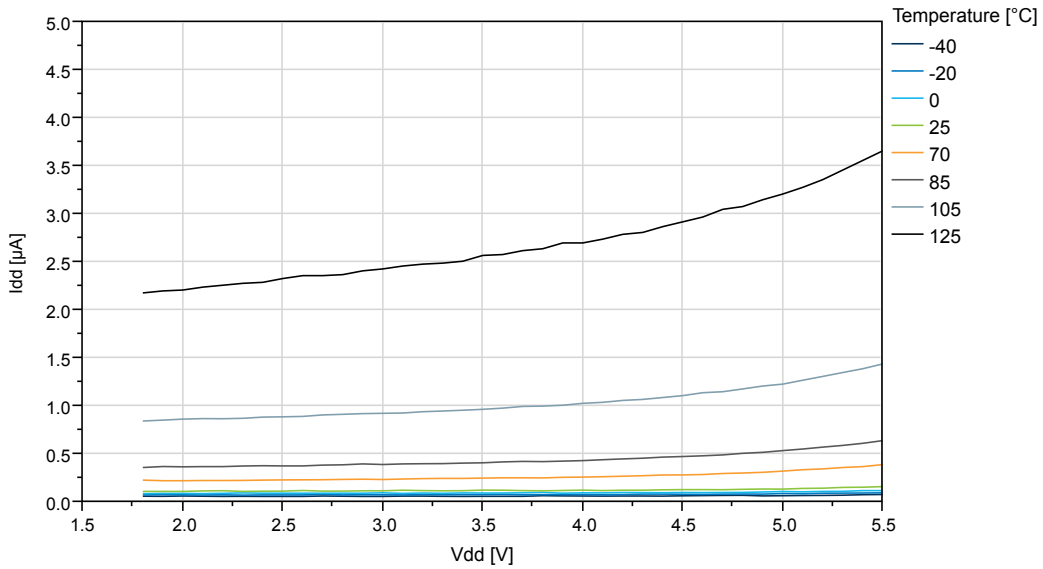


**Figure 34-11. Power Down Mode Supply Current vs. $V_{DD}$ (all functions disabled)**




### 34.1.4 Supply Currents in Standby Mode

**Figure 34-12. Standby Mode Supply Current vs. $V_{DD}$ (RTC running with external 32KHz osc.)**

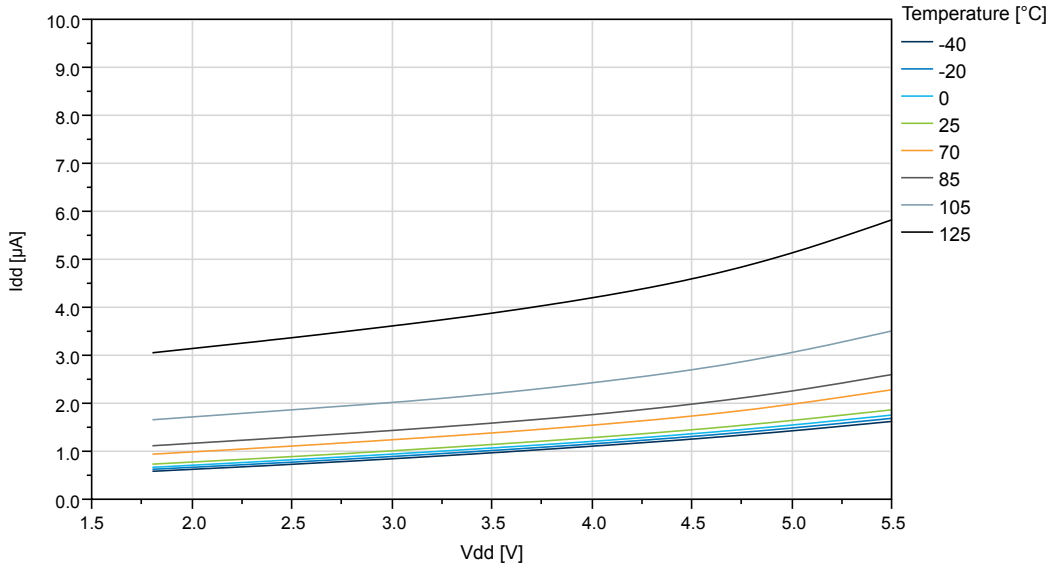


**Figure 34-13. Standby Mode Supply Current vs. $V_{DD}$ (RTC running with internal OSCULP32K)**

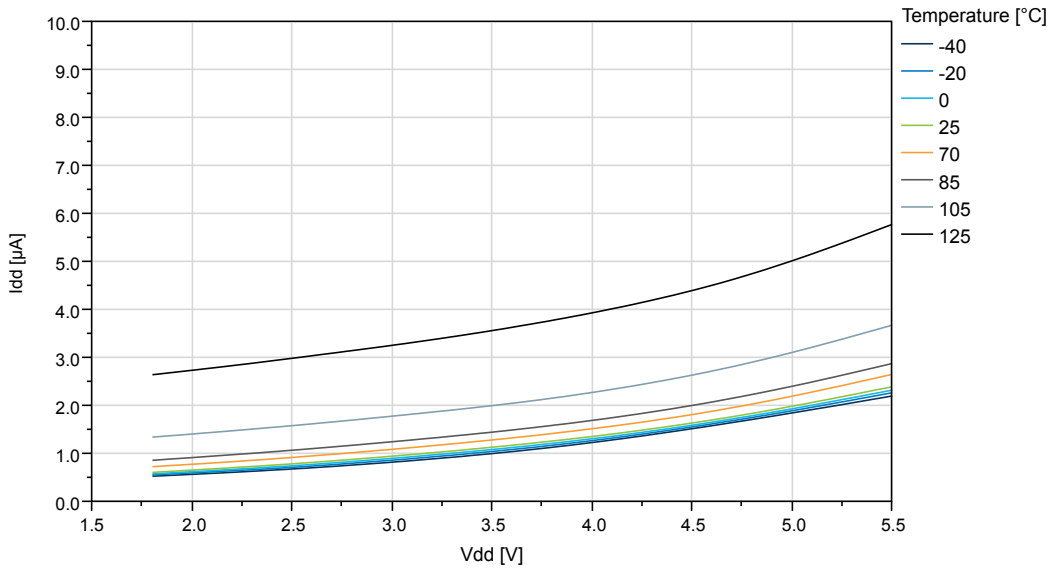



**Figure 34-14. Standby Mode Supply Current vs. $V_{DD}$ (Sampled BOD running at 125Hz)**

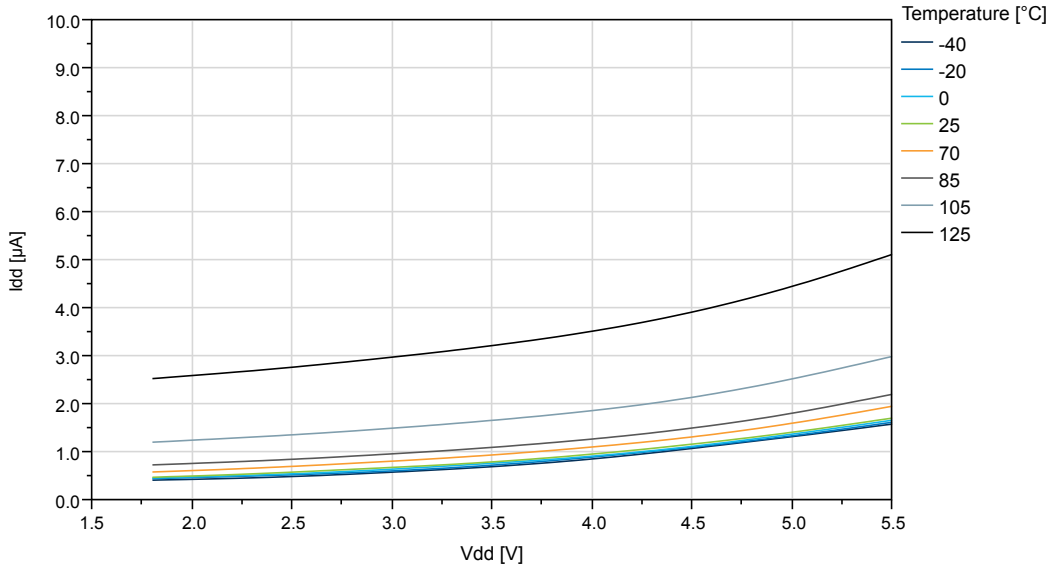
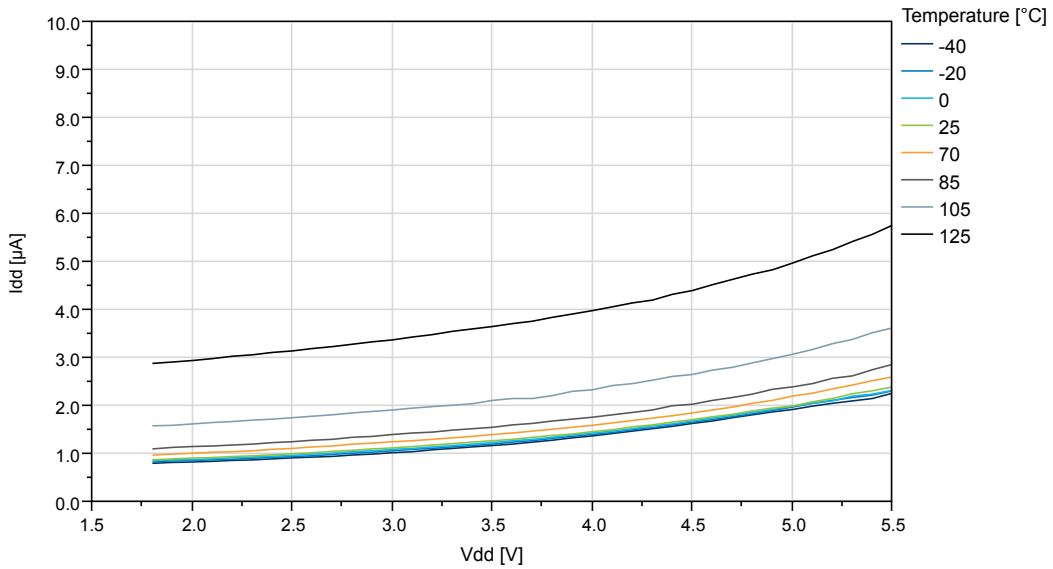


**Figure 34-15. Standby Mode Supply Current vs. $V_{DD}$ (Sampled BOD running at 1kHz)**




## 34.2 GPIO

### GPIO Input Characteristics

**Figure 34-16. IO Pin Input Hysteresis vs. $V_{DD}$**

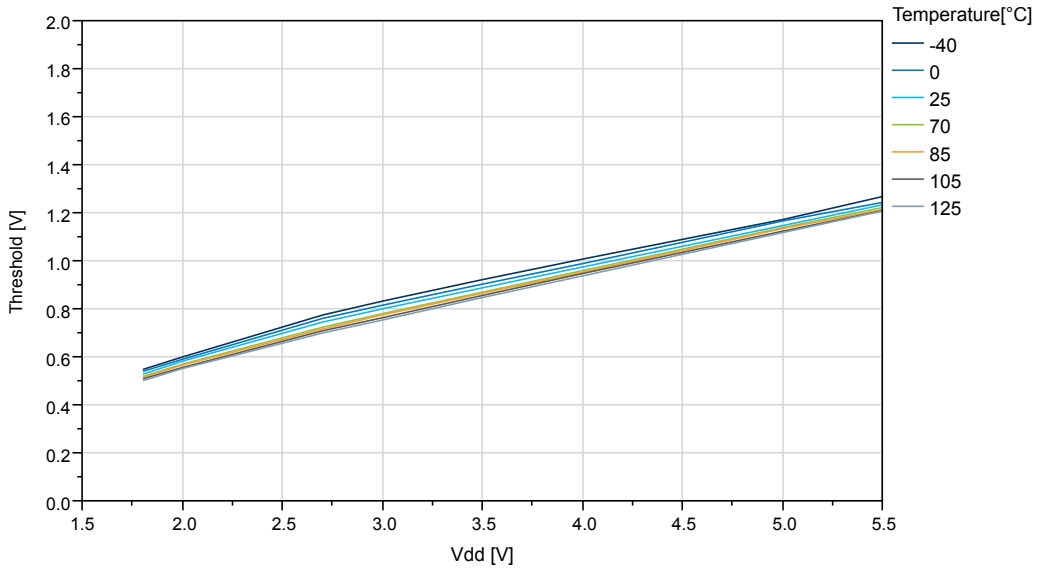


**Figure 34-17. IO Pin Input Threshold Voltage vs. $V_{DD}$ (T=25°C)**

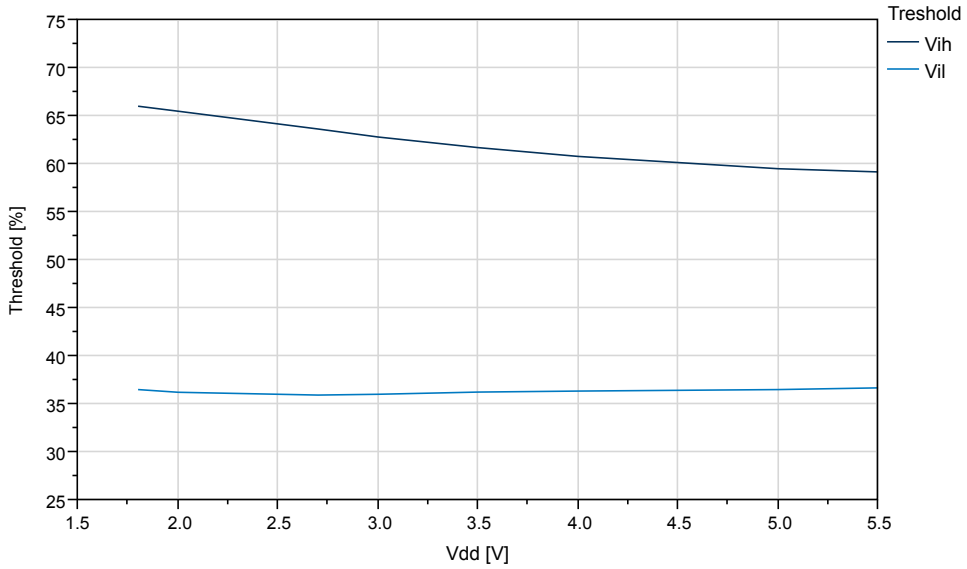



**Figure 34-18. IO Pin Input Threshold Voltage vs. $V_{DD}$ ($V_{IH}$)**

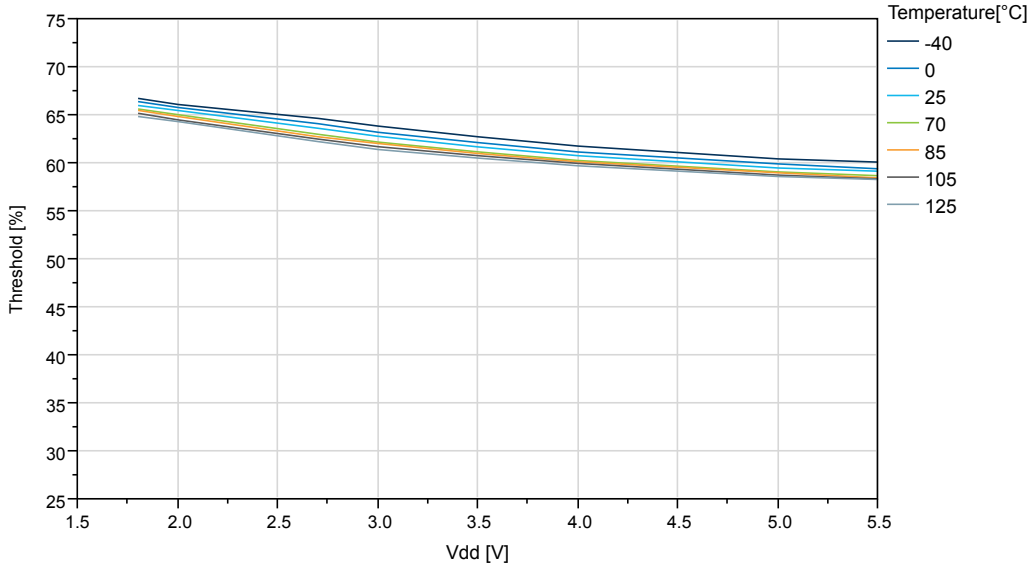
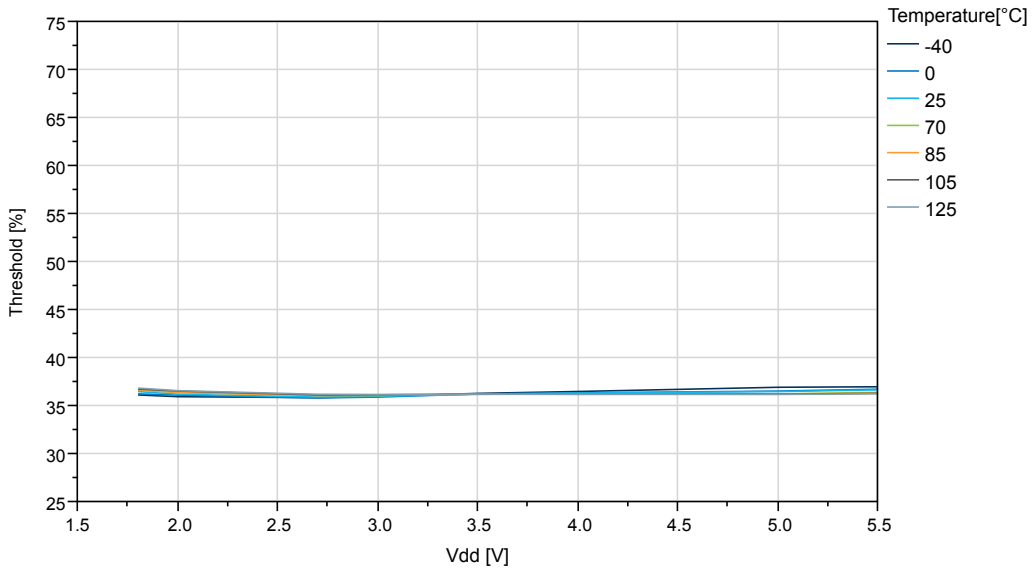


**Figure 34-19. IO Pin Input Threshold Voltage vs. $V_{DD}$ ($V_{IL}$)**




## GPIO Output Characteristics

**Figure 34-20. IO Pin Output Voltage vs. Sink Current ($V_{DD}$=1.8V)**

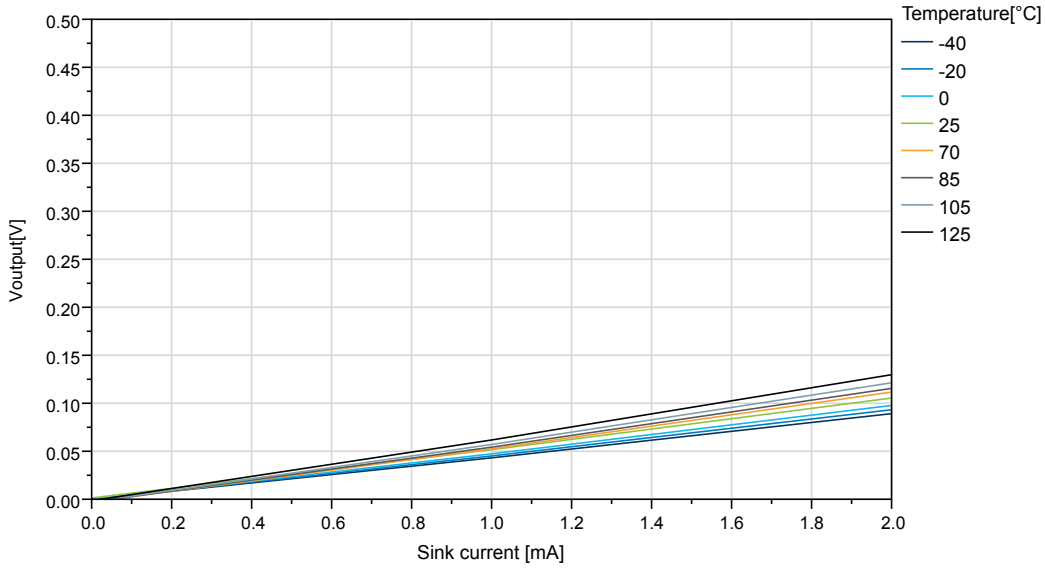


**Figure 34-21. IO Pin Output Voltage vs. Sink Current ($V_{DD}$=3.0V)**

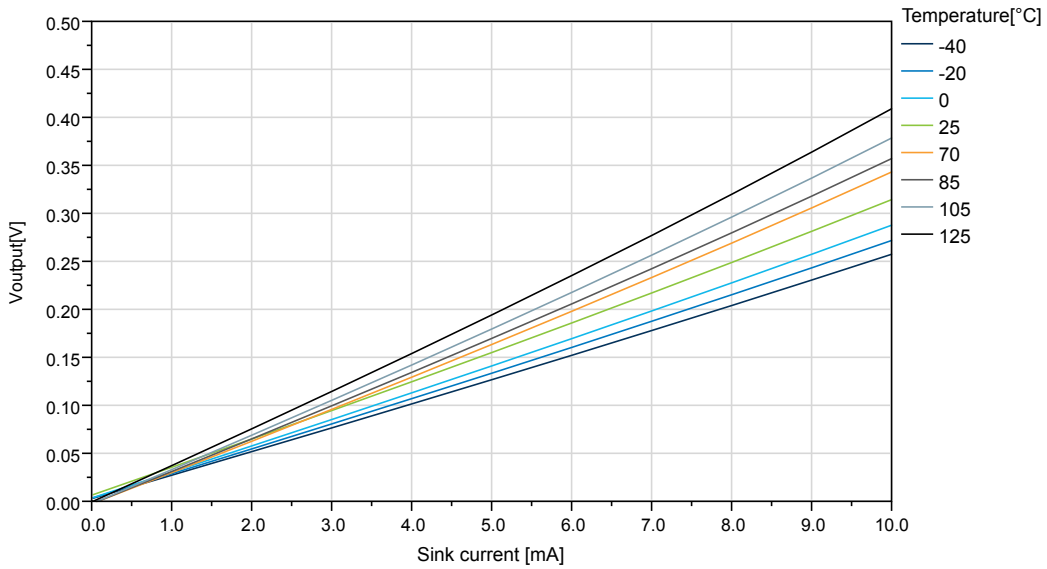



**Figure 34-22. IO Pin Output Voltage vs. Sink Current ($V_{DD}$=5.0V)**

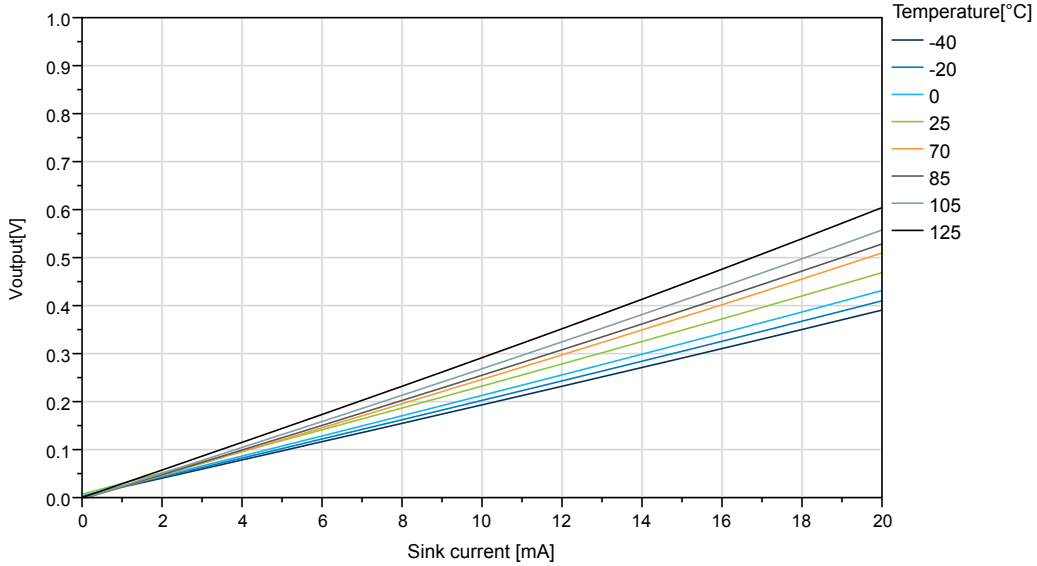


**Figure 34-23. : IO Pin Output Voltage vs. Sink Current (T=25°C)**

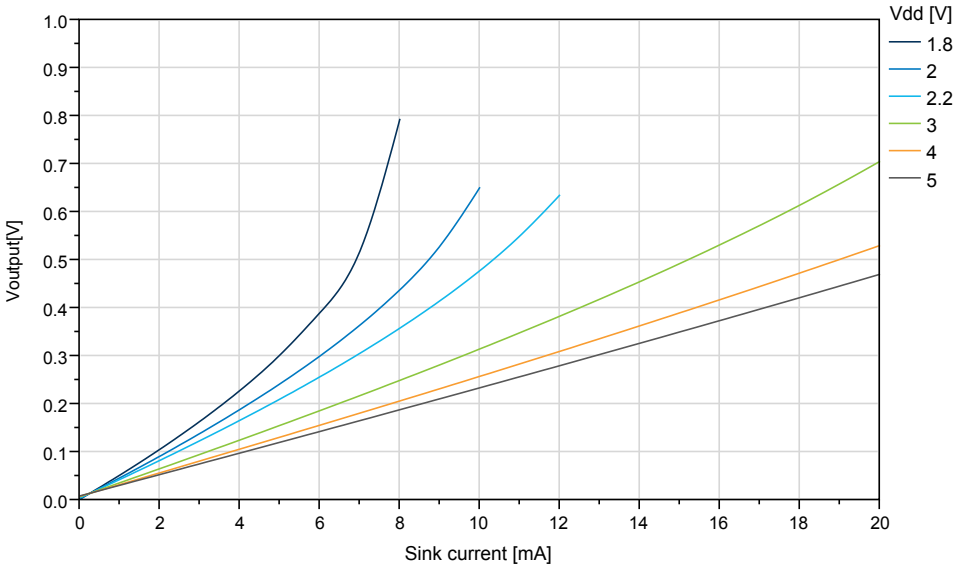



**Figure 34-24. IO Pin Output Voltage vs. Source Current ($V_{DD}$=1.8V)**

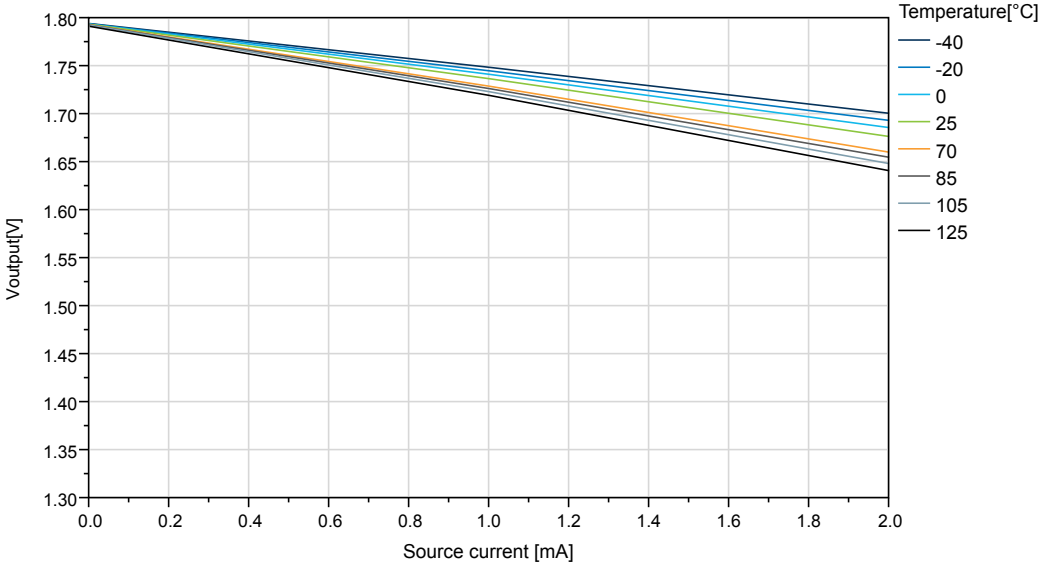


**Figure 34-25. IO Pin Output Voltage vs. Source Current ($V_{DD}$=3.0V)**

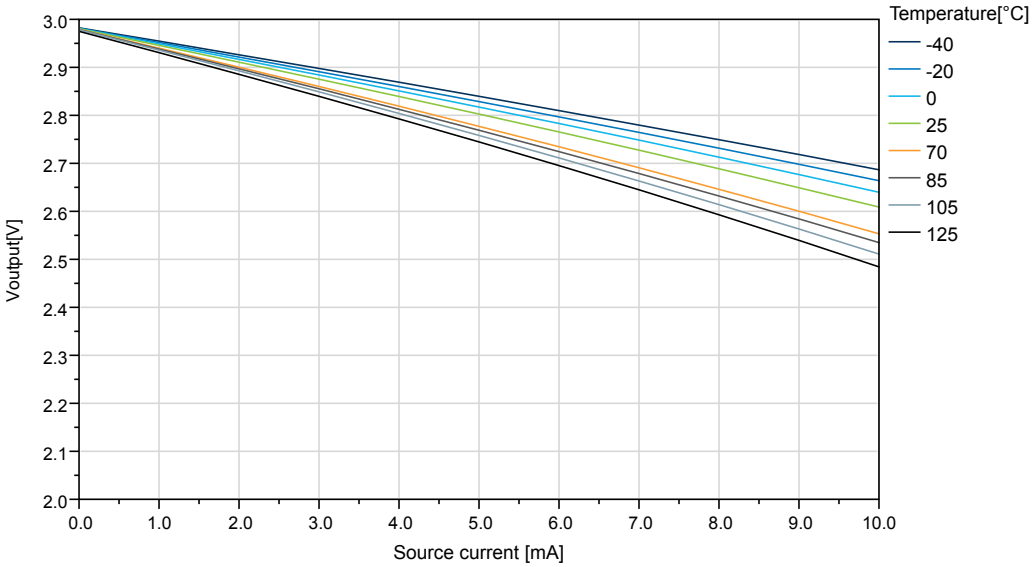



**Figure 34-26. IO Pin Output Voltage vs. Source Current ($V_{DD}$=5.0V)**

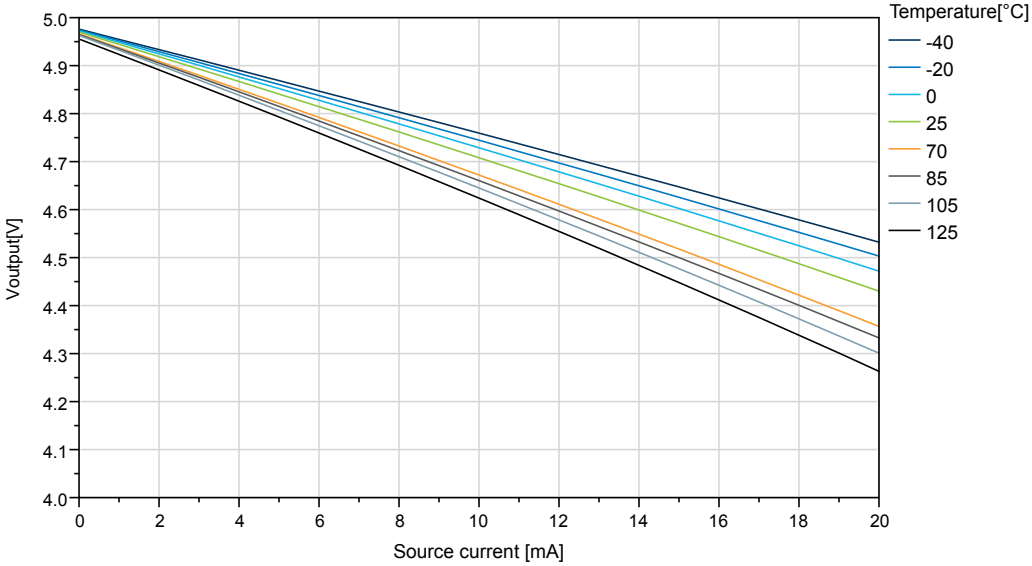
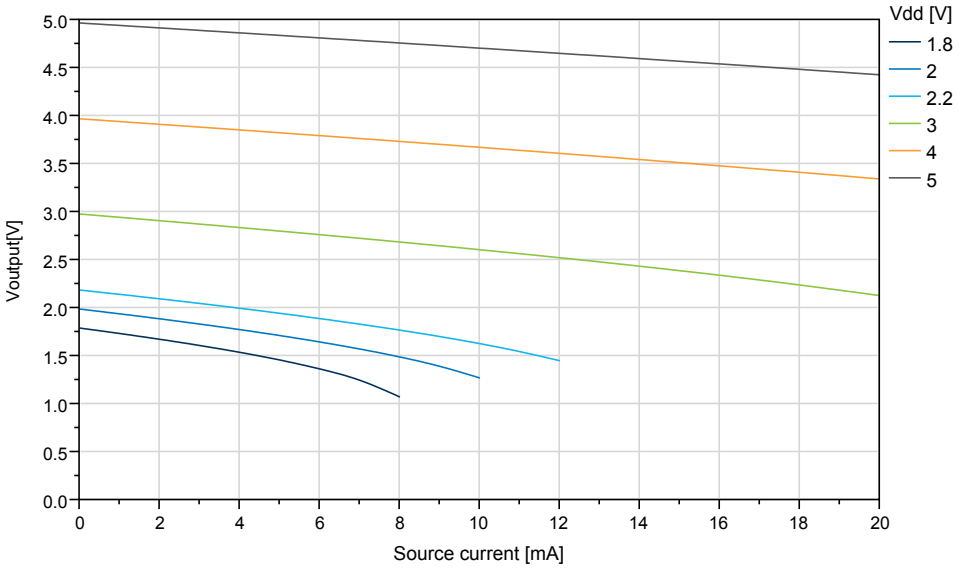


**Figure 34-27. IO Pin Output Voltage vs. Source Current (T=25°C)**




## GPIO Pull-Up Characteristics

**Figure 34-28. IO Pin Pull-Up Resistor Current vs. Input Voltage ($V_{DD}$=1.8V)**

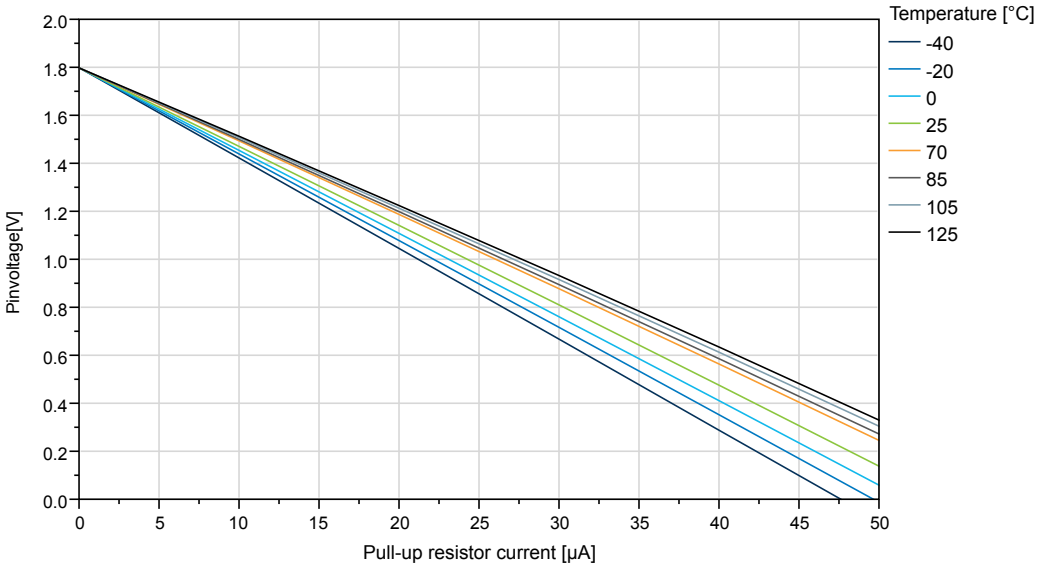


**Figure 34-29. IO Pin Pull-Up Resistor Current vs. Input Voltage ($V_{DD}$=3.0V)**

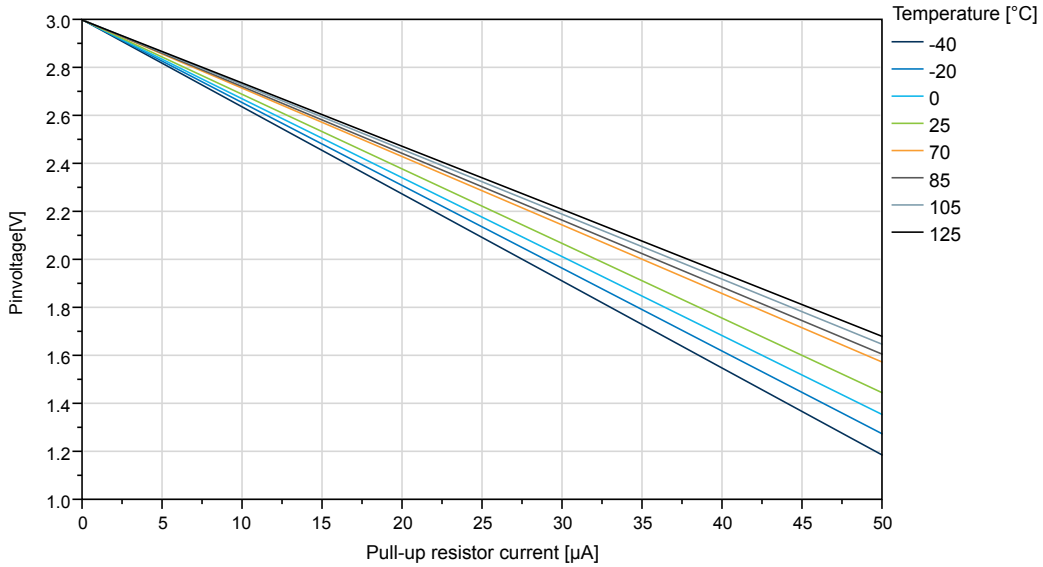
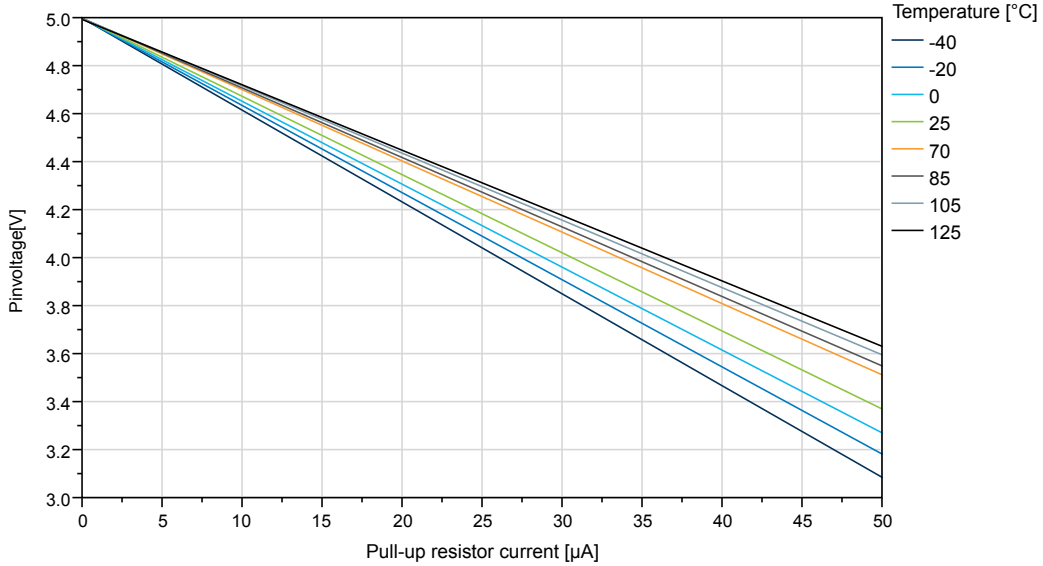



**Figure 34-30. IO Pin Pull-Up Resistor Current vs. Input Voltage ($V_{DD}$=5.0V)**



## 34.3 VREF Characteristics

**Figure 34-31. Internal 0.55V Reference vs. Temperature**

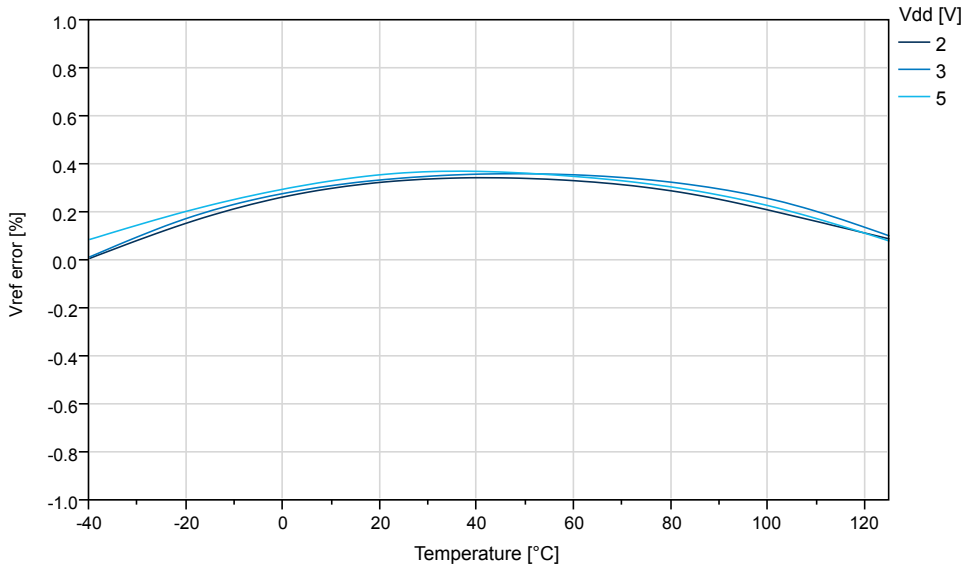



**Figure 34-32. Internal 1.1V Reference vs. Temperature**

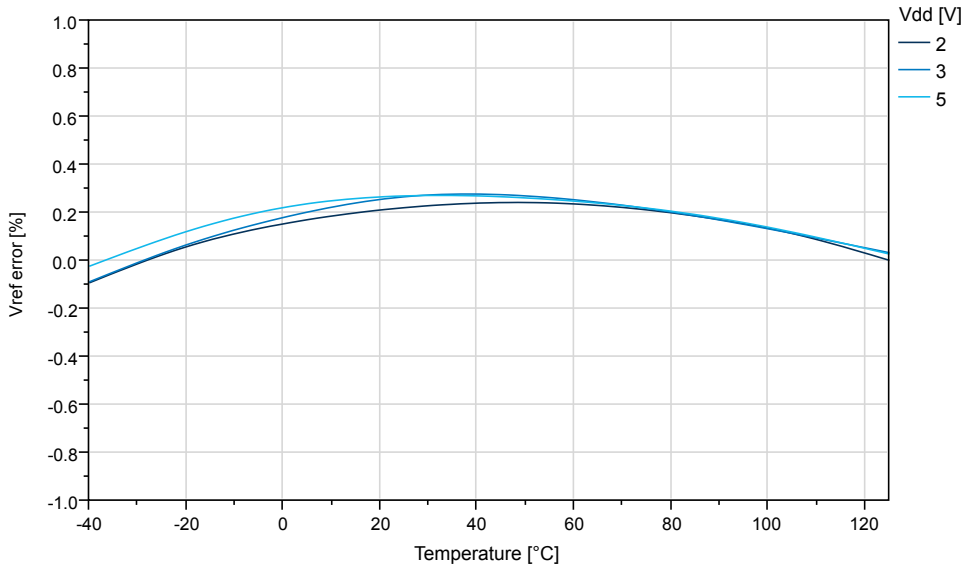


**Figure 34-33. Internal 2.5V Reference vs. Temperature**

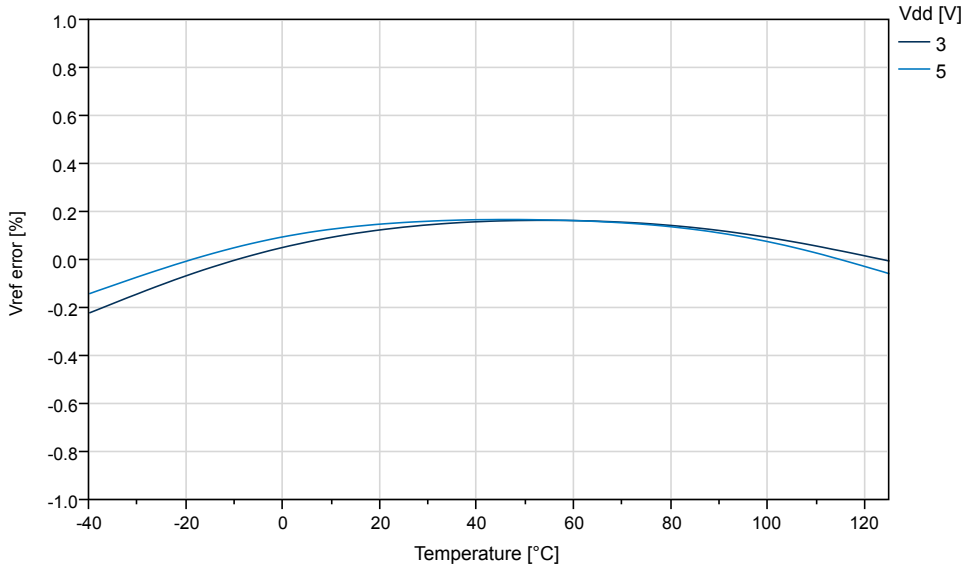
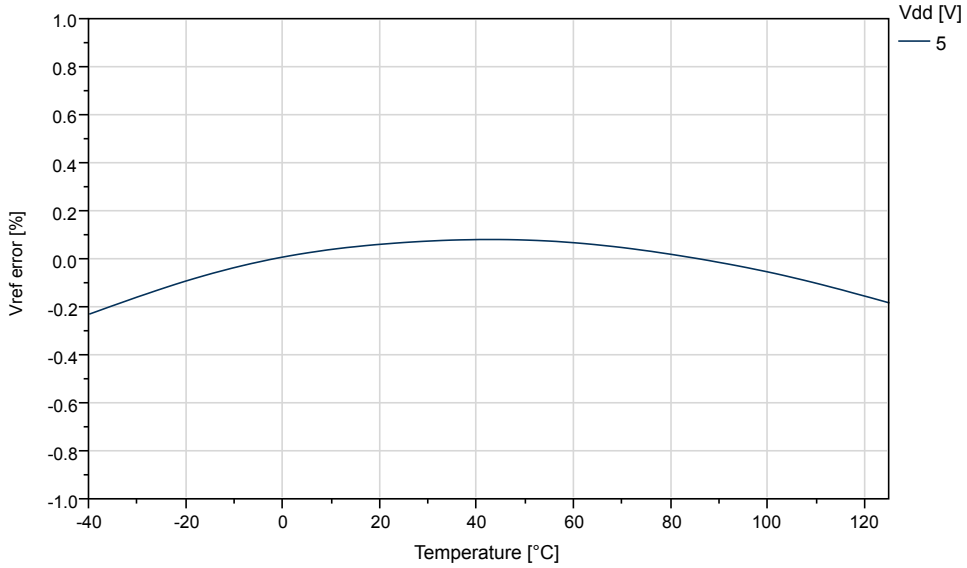



**Figure 34-34. Internal 4.3V Reference vs. Temperature**



## 34.4 BOD Characteristics

### BOD Current vs. $V_{DD}$

**Figure 34-35. BOD Current vs. $V_{DD}$ (Continuous Mode Enabled)**

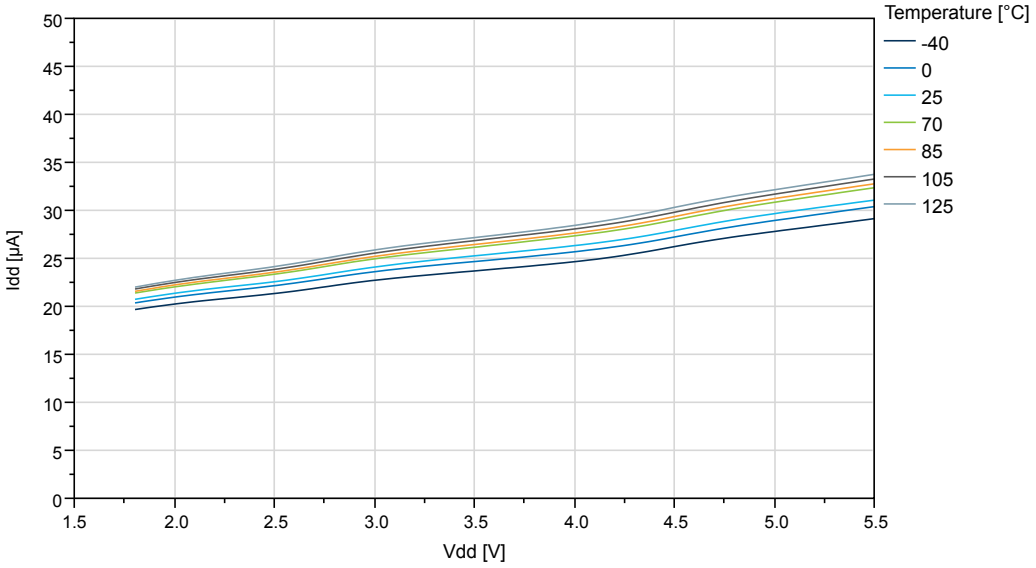



**Figure 34-36. BOD Current vs. $V_{DD}$ (Sampled BOD at 125Hz)**

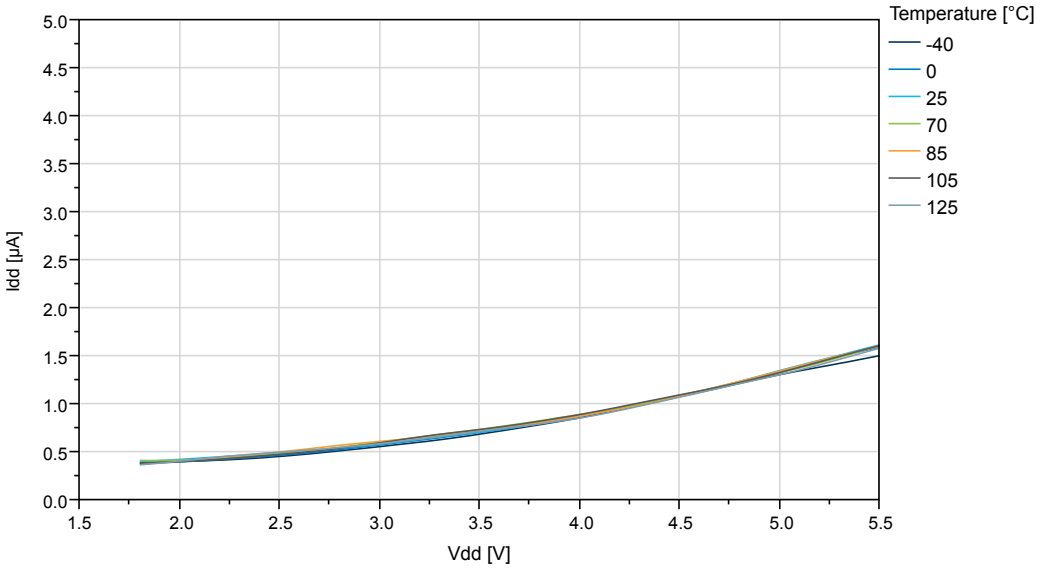
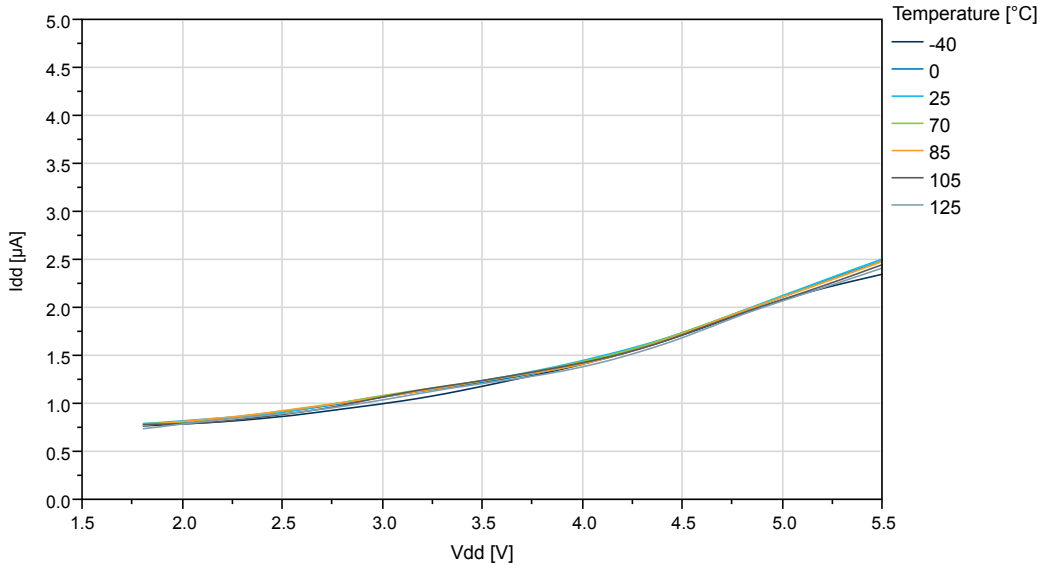


**Figure 34-37. BOD Current vs. $V_{DD}$ (Sampled BOD at 1kHz)**




**BOD Threshold vs. Temperature**

**Figure 34-38. BOD Threshold vs. Temperature (Level 1.8V)**

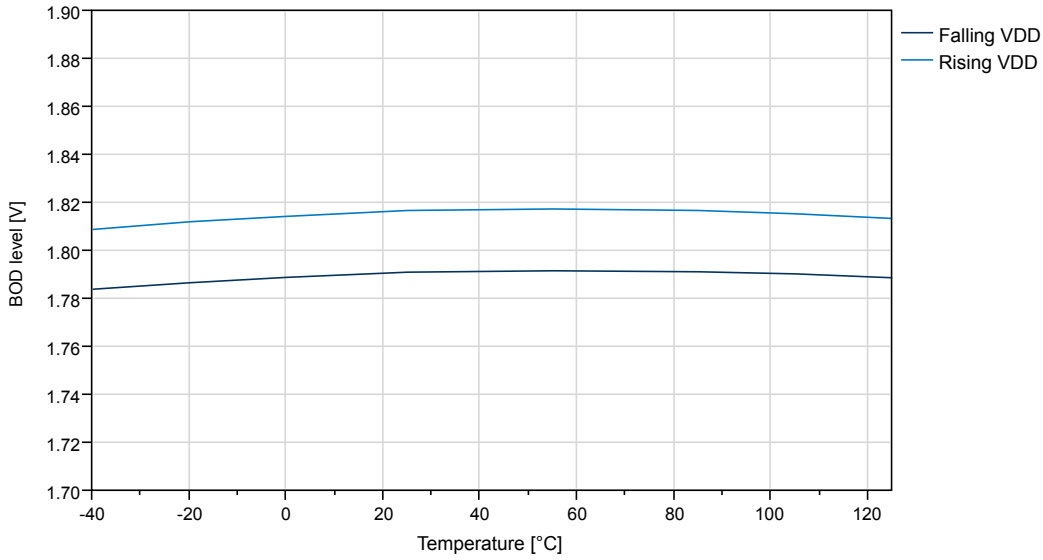


**Figure 34-39. BOD Threshold vs. Temperature (Level 2.6V)**

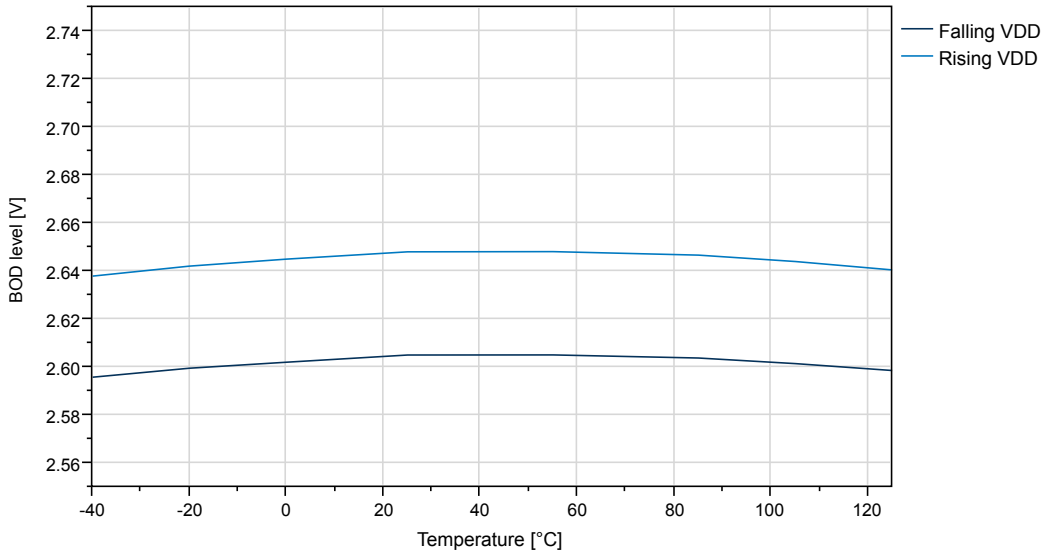
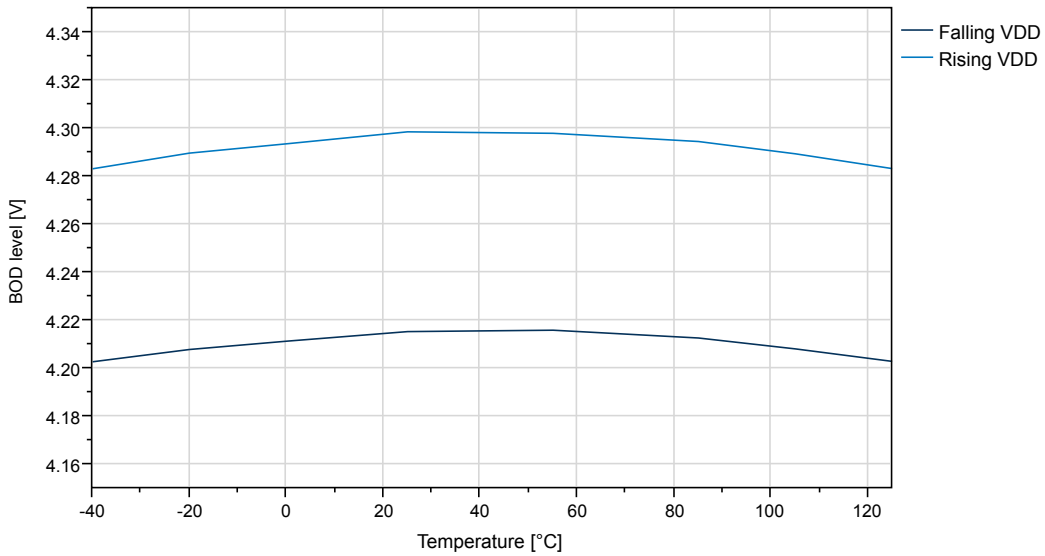



**Figure 34-40. BOD Threshold vs. Temperature (Level 4.3V)**



## 34.5 ADC Characteristics

**Figure 34-41. Absolute Accuracy vs. $V_{DD}$ (115ksps) at T=25°C**

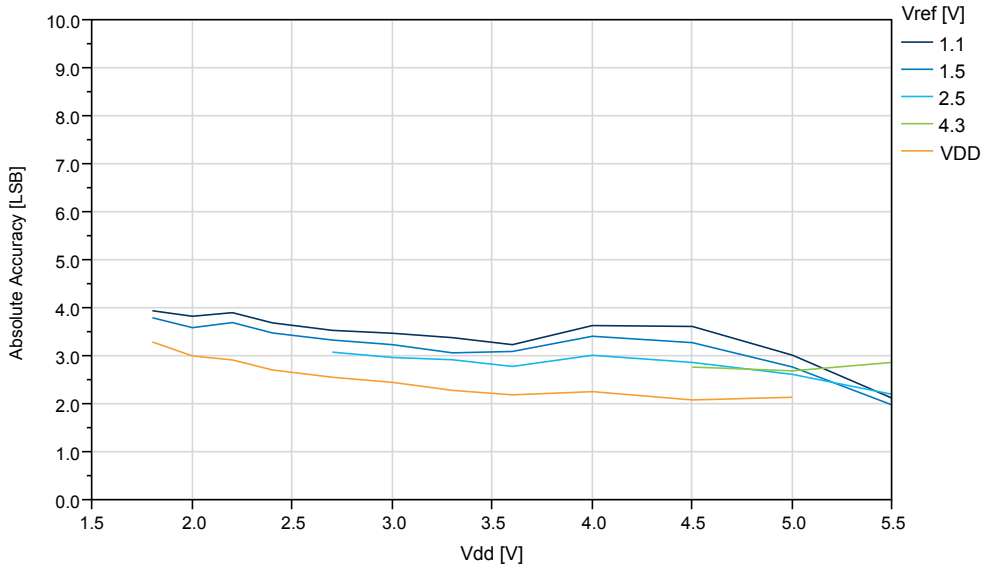



**Figure 34-42. Absolute Accuracy vs. $V_{ref}$ ($V_{DD}$=5.0V, 115ksps)**

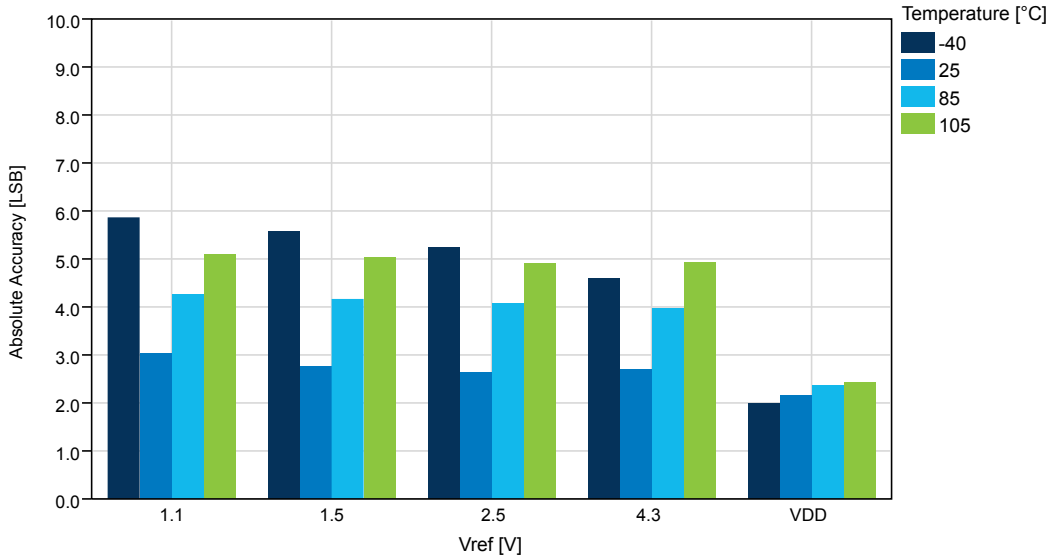


**Figure 34-43. DNL Error vs. $V_{DD}$ (115ksps) at T=25°C**

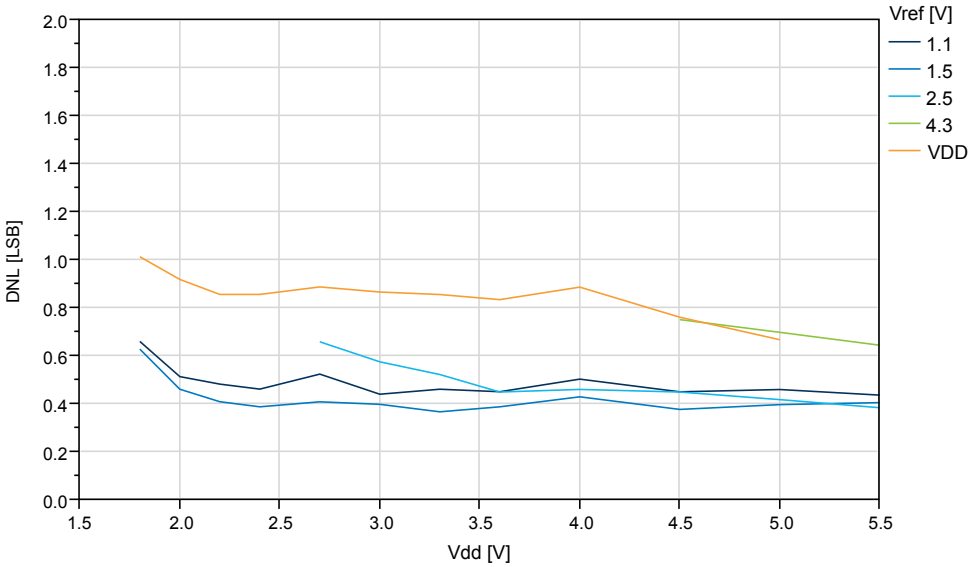



**Figure 34-44. DNL vs. $V_{ref}$ ($V_{DD}$=5.0V, 115ksps)**

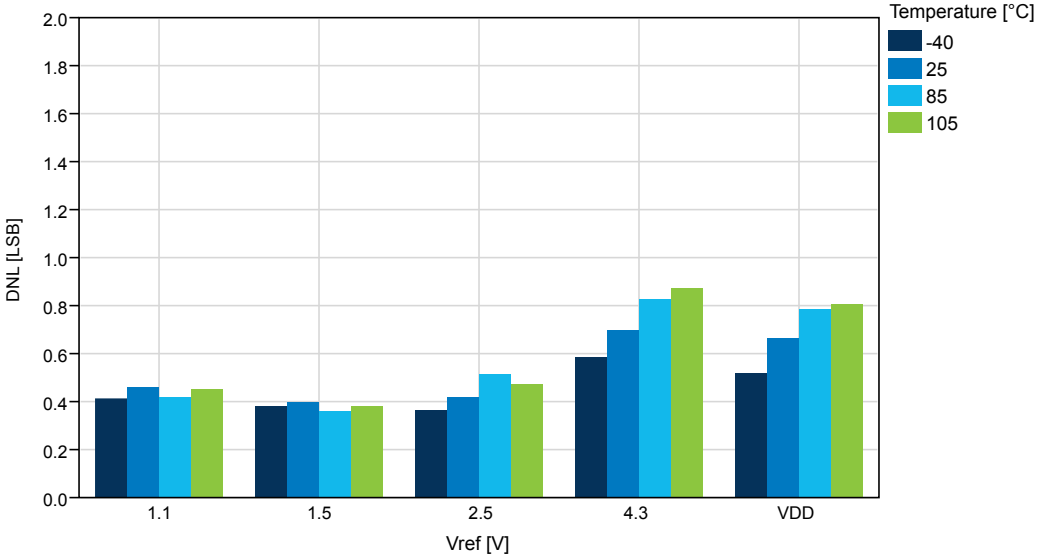


**Figure 34-45. Gain Error vs. $V_{DD}$ (115ksps) at T=25°C**

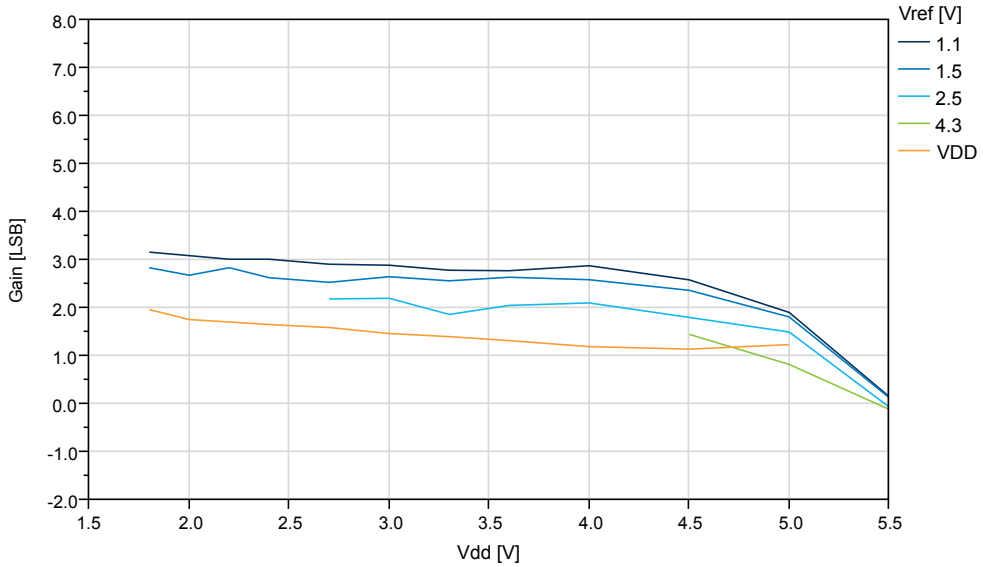



**Figure 34-46. Gain Error vs. $V_{ref}$ ($V_{DD}$=5.0V, 115ksps)**

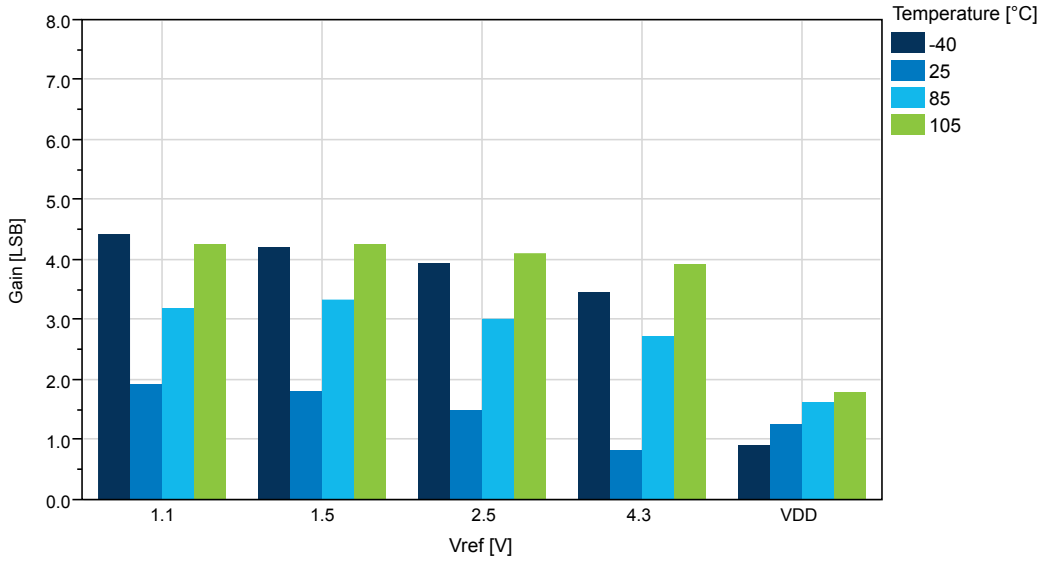


**Figure 34-47. INL vs. $V_{DD}$ (115ksps) at T=25°C**

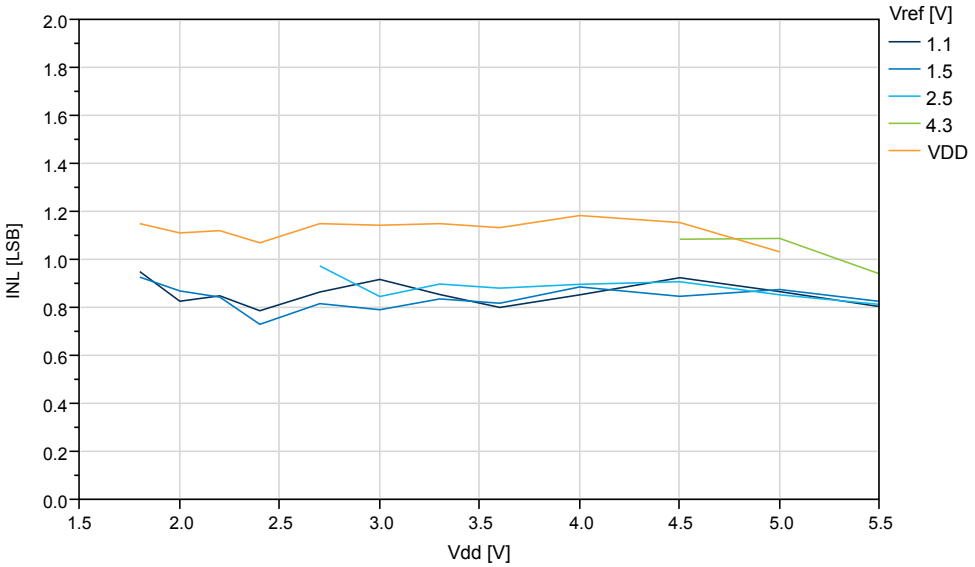



**Figure 34-48. INL vs. $V_{ref}$ ($V_{DD}$=5.0V, 115ksps)**

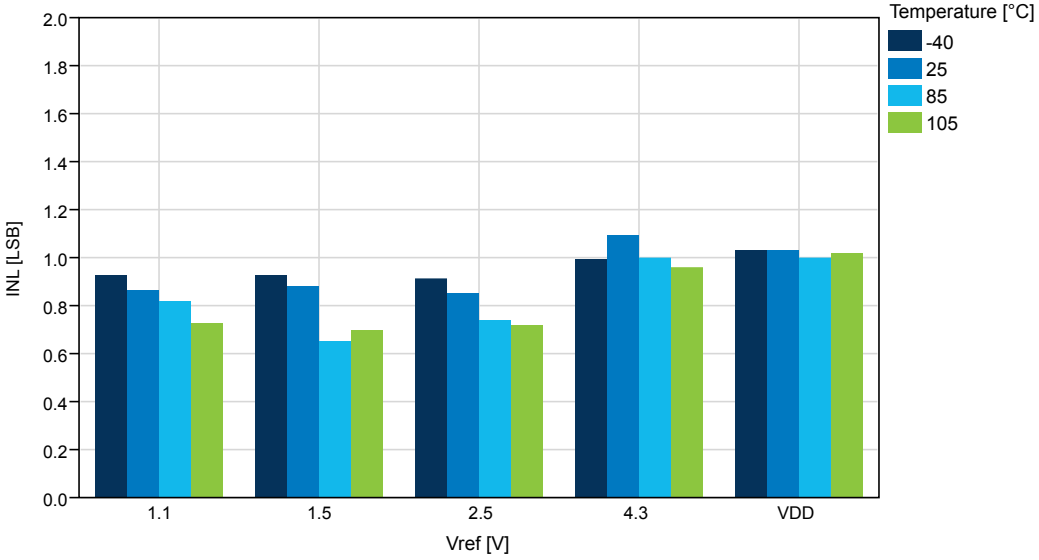


**Figure 34-49. Offset Error vs. $V_{DD}$ (115ksps) at T=25°C**

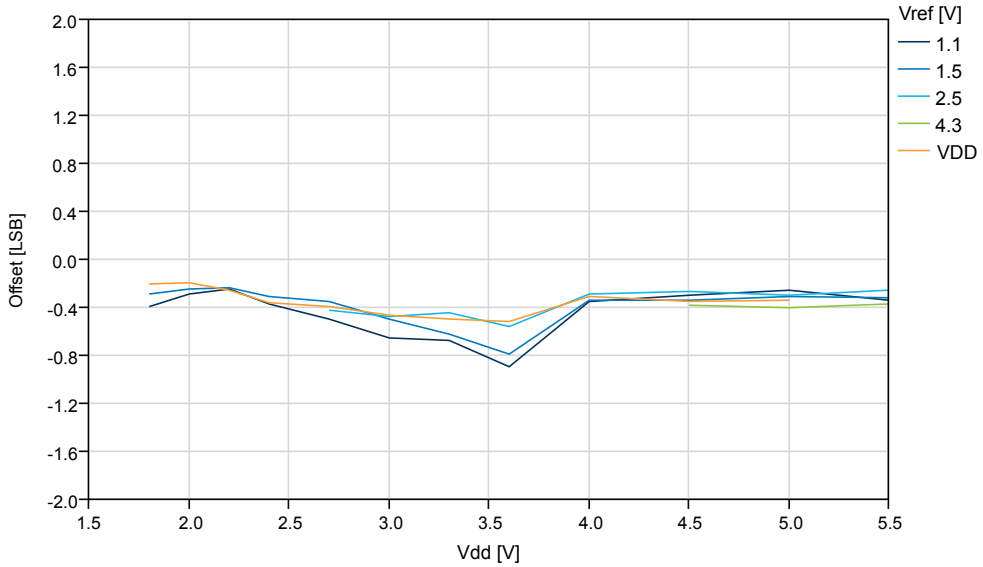
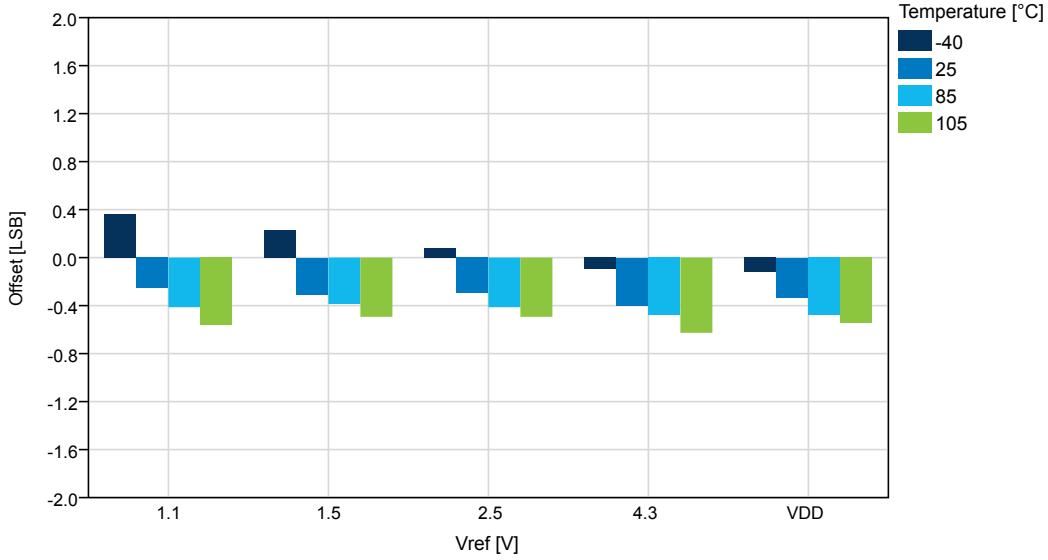



**Figure 34-50. Offset Error vs. $V_{ref}$ ($V_{DD}$=5.0V, 115ksps)**



## 34.6 AC Characteristics

**Figure 34-51. Hysteresis vs. $V_{CM}$ - 10mV ($V_{DD}$=5V)**

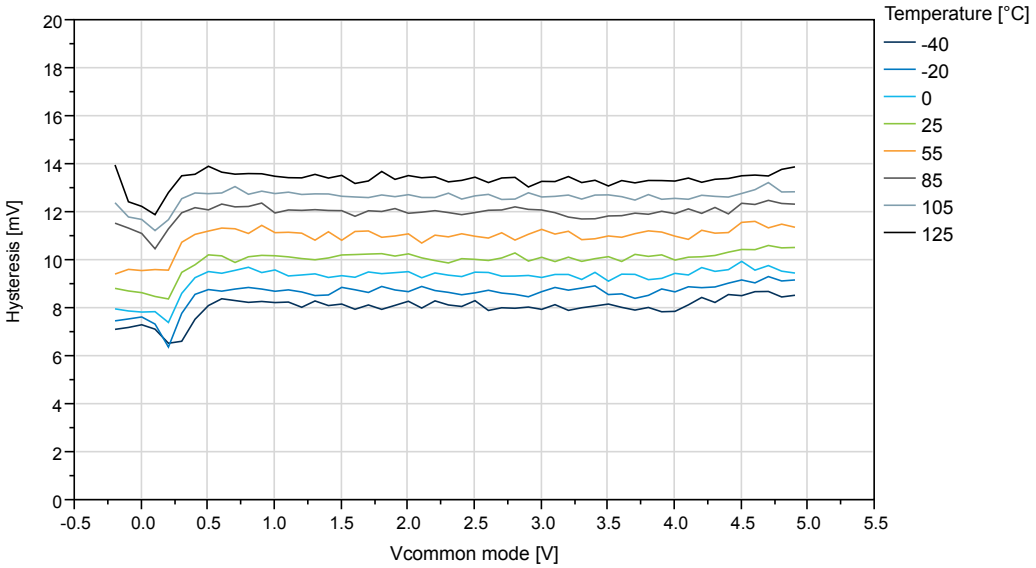



**Figure 34-52. Hysteresis vs. $V_{CM}$ - 10mV to 50mV ($V_{DD}$=5V, T=25°C)**

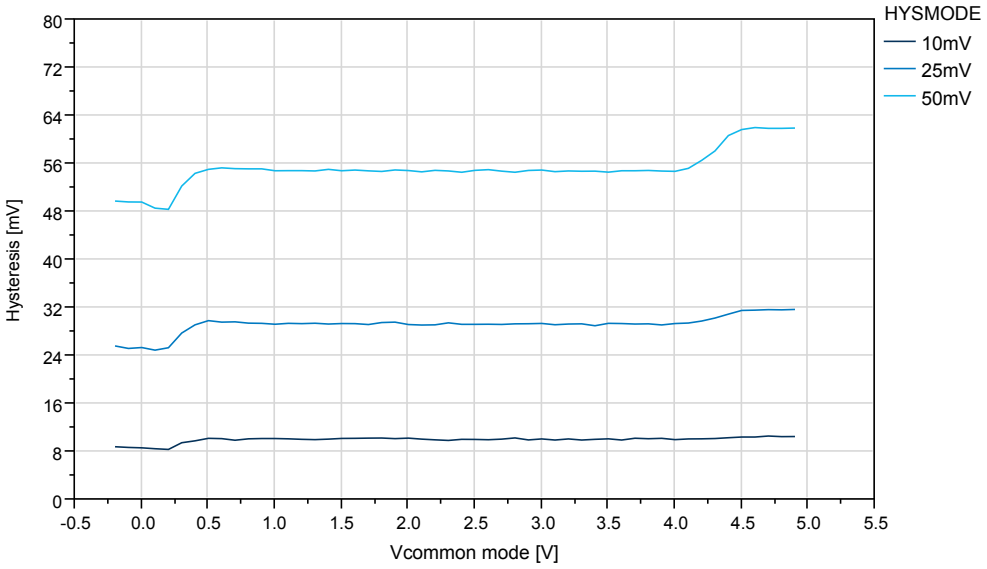


**Figure 34-53. Offset vs. $V_{CM}$ - 10mV ($V_{DD}$=5V)**

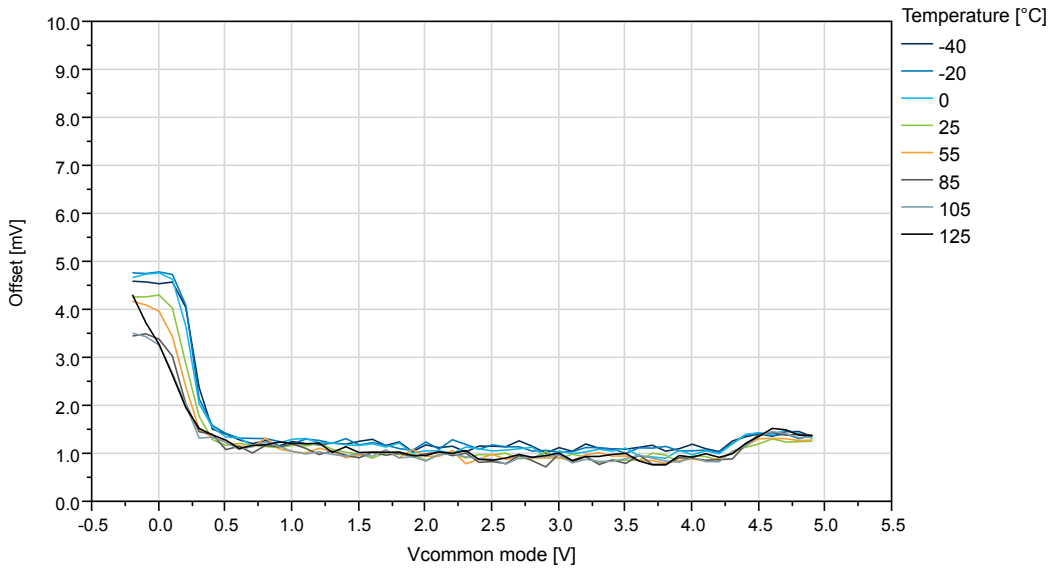
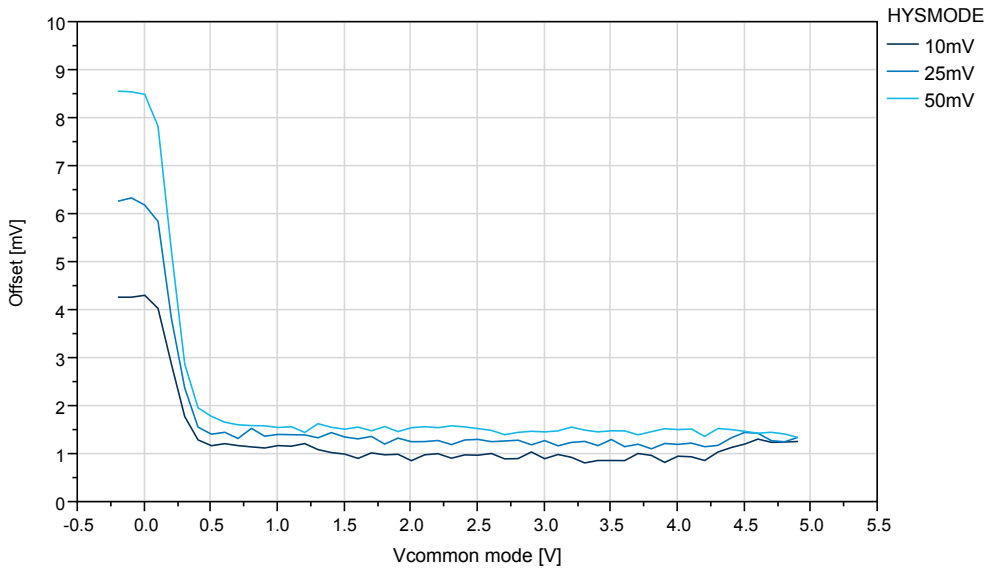



**Figure 34-54. Offset vs. $V_{CM}$ - 10mV to 50mV ($V_{DD}$=5V, T=25°C)**



## 34.7 OSC20M Characteristics

**Figure 34-55. OSC20M Internal Oscillator: Calibration Stepsize vs. Calibration Value ($V_{DD}$=3V)**

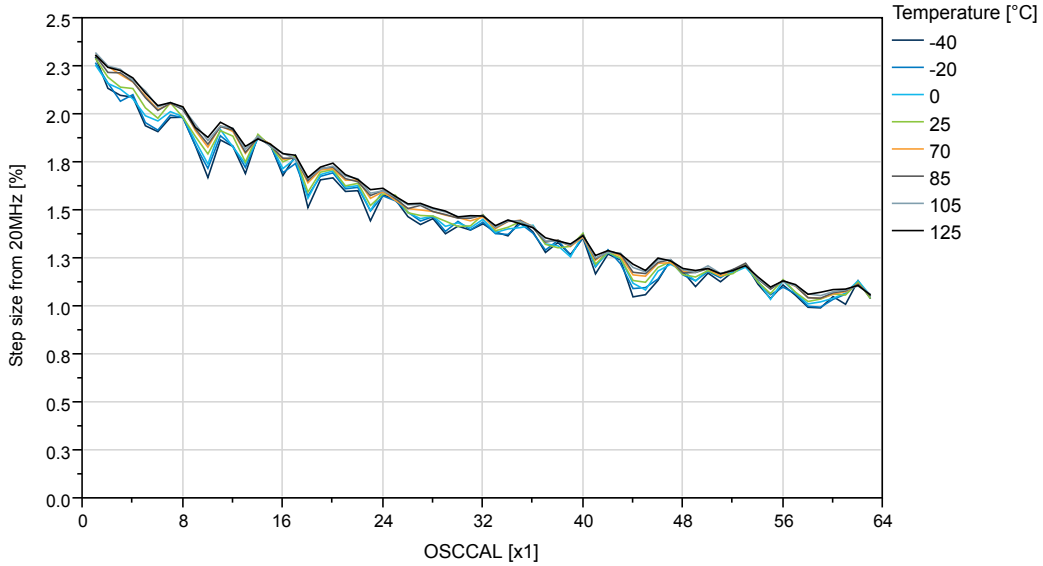



**Figure 34-56. OSC20M internal Oscillator: Frequency vs. Calibration Value ($V_{DD}$=3V)**

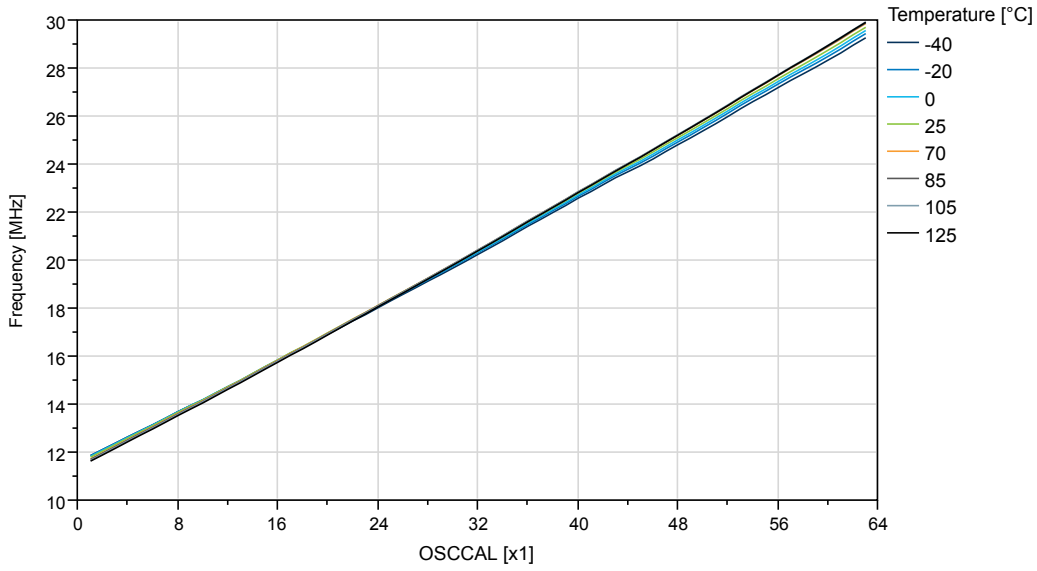


**Figure 34-57. OSC20M Internal Oscillator: Frequency vs. Temperature**

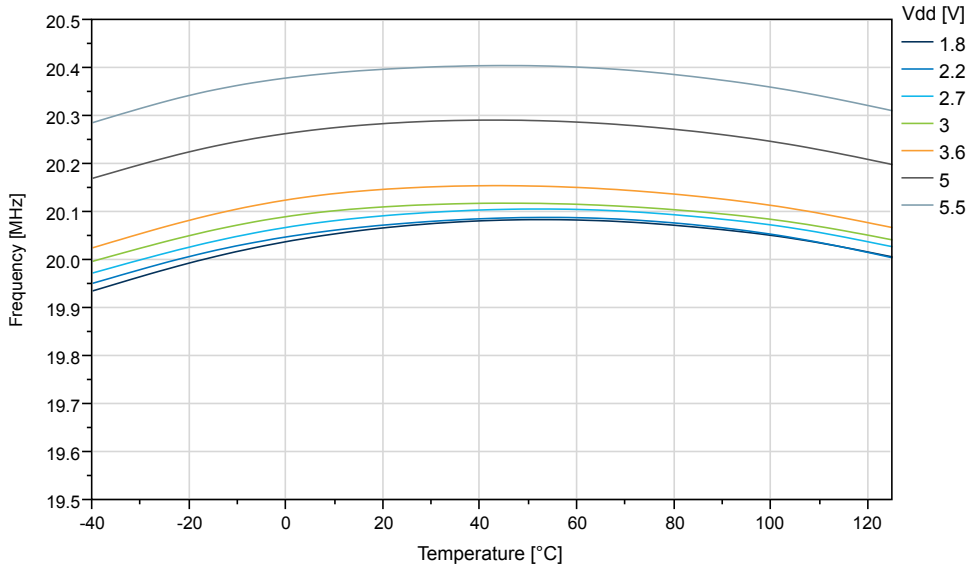
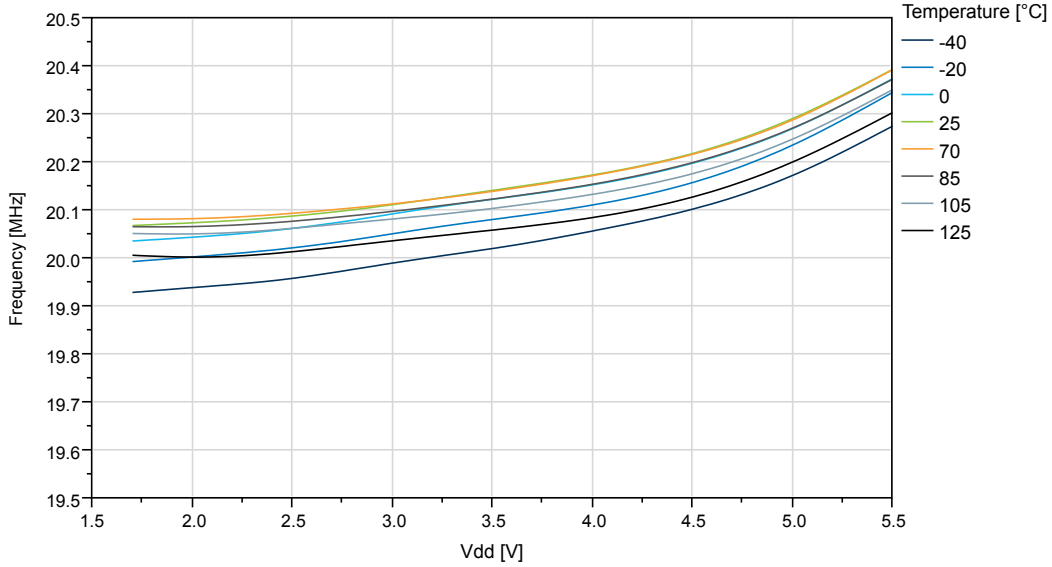



**Figure 34-58. OSC20M Internal Oscillator: Frequency vs. $V_{DD}$**



## 34.8 OSCULP32K Characteristics

**Figure 34-59. OSCULP32K Internal Oscillator Frequency vs. Temperature**

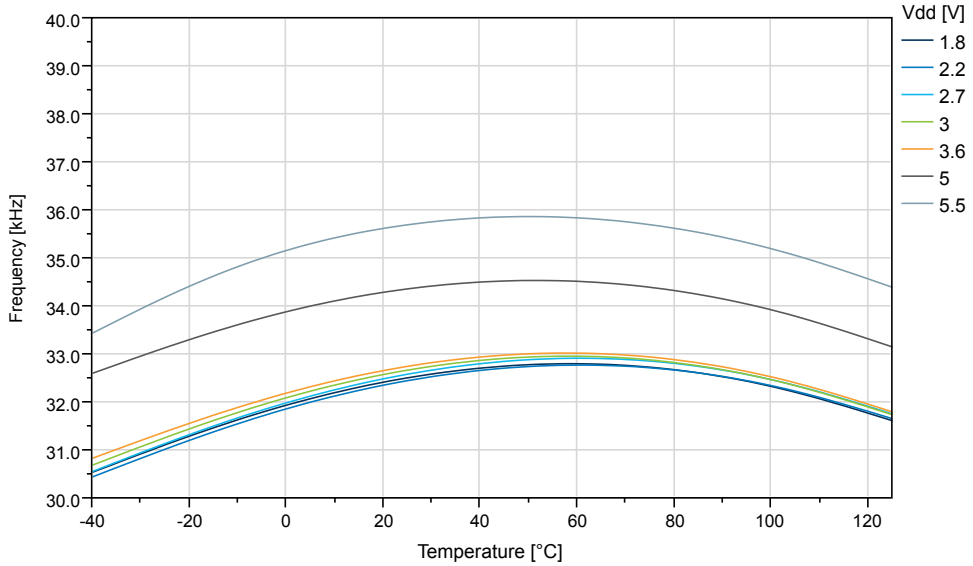
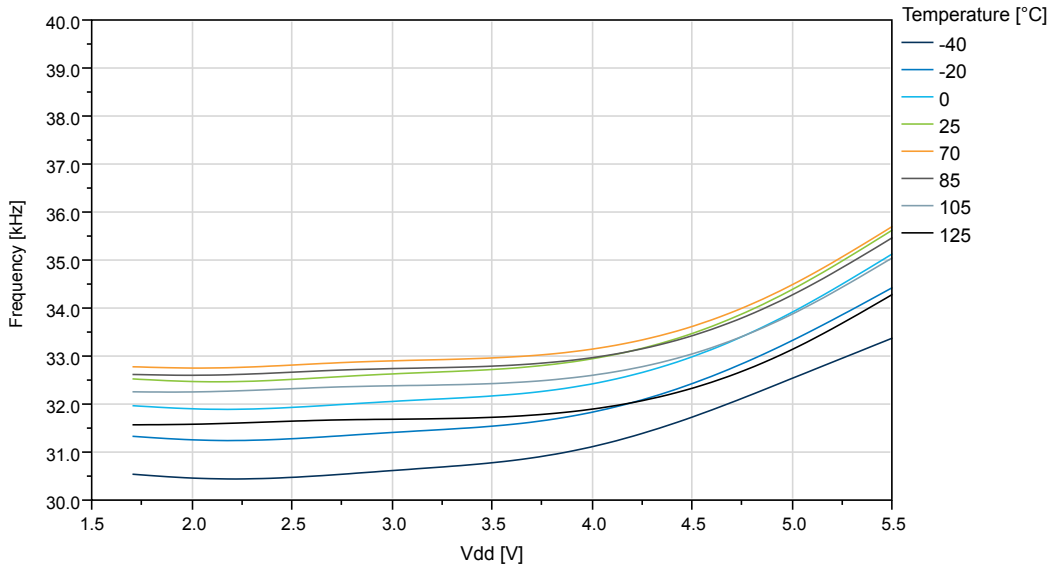


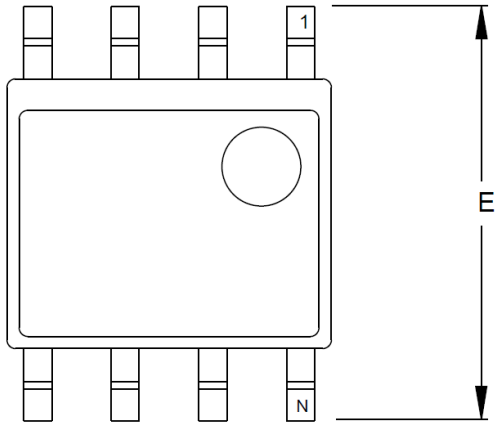

**Figure 34-60. OSCULP32K Internal Oscillator Frequency vs. $V_{DD}$**



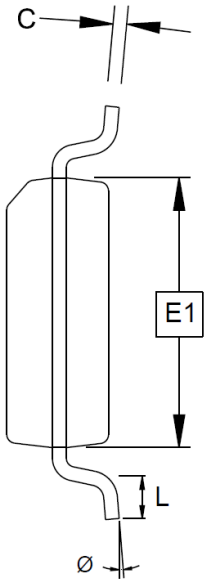
# 35. Packaging Information

## 35.1 Package Drawings

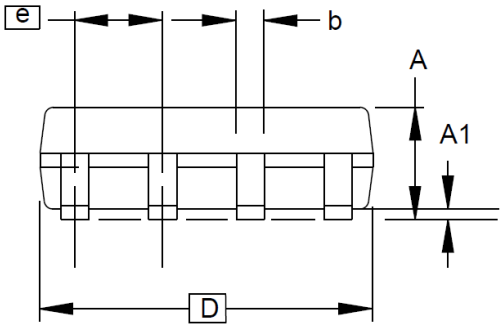
### 35.1.1 8-pin SOIC150




TOP VIEW




END VIEW




SIDE VIEW

Notes: This drawing is for general information only.
Refer to JEDEC Drawing MS-012, Variation AA
for proper dimensions, tolerances, datums, etc.

COMMON DIMENSIONS
(Unit of Measure = mm)

| SYMBOL | MIN | NOM | MAX | NOTE |
|---|---|---|---|---|
| A | 1.35 | – | 1.75 | |
| A1 | 0.10 | – | 0.25 | |
| b | 0.31 | – | 0.51 | |
| C | 0.17 | – | 0.25 | |
| D | 4.80 | – | 5.05 | |
| E1 | 3.81 | – | 3.99 | |
| E | 5.79 | – | 6.20 | |
| e | 1.27 BSC | | | |
| L | 0.40 | – | 1.27 | |
| Ø | 0° | – | 8° | |

6/22/11

| Atmel<br>Package Drawing Contact:<br>packagedrawings@atmel.com | TITLE<br>8S1, 8-lead (0.150” Wide Body), Plastic Gull Wing Small Outline (JEDEC SOIC) | GPC<br>SWB | DRAWING NO.<br>8S1 | REV.<br>G |
|---|---|---|---|---|

## 35.2 Thermal Considerations

### 35.2.1 Thermal Resistance Data

The following table summarizes the thermal resistance data depending on the package.

**Table 35-1. Thermal Resistance Data**

| Package Type | $\theta_{JA}$ [°C/W] | $\theta_{JC}$ [°C/W] |
|---|---|---|
| 8-pin SOIC150 (SWB) | 91.8 | 21.5 |

**Related Links**

Junction Temperature

### 35.2.2 Junction Temperature

The average chip-junction temperature, $T_J$, in °C can be obtained from the following:

1. $T_J = T_A + (P_D \times \theta_{JA})$
2. $T_J = T_A + (P_D \times (\theta_{HEATSINK} + \theta_{JC}))$

where:

- $\theta_{JA}$ = Package thermal resistance, Junction-to-ambient (°C/W), see Thermal Resistance Data
- $\theta_{JC}$ = Package thermal resistance, Junction-to-case thermal resistance (°C/W), see Thermal Resistance Data
- $\theta_{HEATSINK}$ = Thermal resistance (°C/W) specification of the external cooling device
- $P_D$ = Device power consumption (W)
- $T_A$ = Ambient temperature (°C)

From the first equation, the user can derive the estimated lifetime of the chip and decide if a cooling device is necessary or not. If a cooling device is to be fitted on the chip, the second equation should be used to compute the resulting average chip-junction temperature $T_J$ in °C.

**Related Links**

Thermal Resistance Data

# 36. Instruction Set Summary

### Table 36-1. Arithmetic and Logic Instructions

| Mnemonic | Operands | Description | | Op | | Flags | #Clocks |
|---|---|---|---|---|---|---|---|
| ADD | Rd, Rr | Add without Carry | Rd | ← | Rd + Rr | Z,C,N,V,S,H | 1 |
| ADC | Rd, Rr | Add with Carry | Rd | ← | Rd + Rr + C | Z,C,N,V,S,H | 1 |
| ADIW | Rd, K | Add Immediate to Word | Rd | ← | Rd + 1:Rd + K | Z,C,N,V,S | 2 |
| SUB | Rd, Rr | Subtract without Carry | Rd | ← | Rd - Rr | Z,C,N,V,S,H | 1 |
| SUBI | Rd, K | Subtract Immediate | Rd | ← | Rd - K | Z,C,N,V,S,H | 1 |
| SBC | Rd, Rr | Subtract with Carry | Rd | ← | Rd - Rr - C | Z,C,N,V,S,H | 1 |
| SBCI | Rd, K | Subtract Immediate with Carry | Rd | ← | Rd - K - C | Z,C,N,V,S,H | 1 |
| SBIW | Rd, K | Subtract Immediate from Word | Rd + 1:Rd | ← | Rd + 1:Rd - K | Z,C,N,V,S | 2 |
| AND | Rd, Rr | Logical AND | Rd | ← | Rd • Rr | Z,N,V,S | 1 |
| ANDI | Rd, K | Logical AND with Immediate | Rd | ← | Rd • K | Z,N,V,S | 1 |
| OR | Rd, Rr | Logical OR | Rd | ← | Rd v Rr | Z,N,V,S | 1 |
| ORI | Rd, K | Logical OR with Immediate | Rd | ← | Rd v K | Z,N,V,S | 1 |
| EOR | Rd, Rr | Exclusive OR | Rd | ← | Rd ⊕ Rr | Z,N,V,S | 1 |
| COM | Rd | One's Complement | Rd | ← | $FF - Rd | Z,C,N,V,S | 1 |
| NEG | Rd | Two's Complement | Rd | ← | $00 - Rd | Z,C,N,V,S,H | 1 |
| SBR | Rd,K | Set Bit(s) in Register | Rd | ← | Rd v K | Z,N,V,S | 1 |
| CBR | Rd,K | Clear Bit(s) in Register | Rd | ← | Rd • ($FFh - K) | Z,N,V,S | 1 |
| INC | Rd | Increment | Rd | ← | Rd + 1 | Z,N,V,S | 1 |
| DEC | Rd | Decrement | Rd | ← | Rd - 1 | Z,N,V,S | 1 |
| TST | Rd | Test for Zero or Minus | Rd | ← | Rd • Rd | Z,N,V,S | 1 |
| CLR | Rd | Clear Register | Rd | ← | Rd ⊕ Rd | Z,N,V,S | 1 |
| SER | Rd | Set Register | Rd | ← | $FF | None | 1 |
| MUL | Rd,Rr | Multiply Unsigned | R1:R0 | ← | Rd x Rr (UU) | Z,C | 2 |
| MULS | Rd,Rr | Multiply Signed | R1:R0 | ← | Rd x Rr (SS) | Z,C | 2 |
| MULSU | Rd,Rr | Multiply Signed with Unsigned | R1:R0 | ← | Rd x Rr (SU) | Z,C | 2 |
| FMUL | Rd,Rr | Fractional Multiply Unsigned | R1:R0 | ← | Rd x Rr<<1 (UU) | Z,C | 2 |

| Mnemonic | Operands | Description | | Op | | Flags | #Clocks |
|---|---|---|---|---|---|---|---|
| FMULS | Rd,Rr | Fractional Multiply Signed | R1:R0 | ← | Rd x Rr<<1 (SS) | Z,C | 2 |
| FMULSU | Rd,Rr | Fractional Multiply Signed with Unsigned | R1:R0 | ← | Rd x Rr<<1 (SU) | Z,C | 2 |

**Table 36-2. Branch Instructions**

| Mnemonic | Operands | Description | | Op | | Flags | #Clocks |
|---|---|---|---|---|---|---|---|
| RJMP | k | Relative Jump | PC | ← | PC + k + 1 | None | 2 |
| IJMP | | Indirect Jump to (Z) | PC(15:0)<br>PC(21:16) | ←<br>← | Z<br>0 | None | 2 |
| JMP | k | Jump | PC | ← | k | None | 3 |
| RCALL | k | Relative Call Subroutine | PC | ← | PC + k + 1 | None | 2 / 3 |
| ICALL | | Indirect Call to (Z) | PC(15:0)<br>PC(21:16) | ←<br>← | Z<br>0 | None | 2 / 3 |
| CALL | k | Call Subroutine | PC | ← | k | None | 3 / 4 |
| RET | | Subroutine Return | PC | ← | STACK | None | 4 / 5 |
| RETI | | Interrupt Return | PC | ← | STACK | I | 4 / 5 |
| CPSE | Rd,Rr | Compare, skip if Equal | if (Rd = Rr) PC | ← | PC + 2 or 3 | None | 1 / 2 / 3 |
| CP | Rd,Rr | Compare | Rd - Rr | | | Z,C,N,V,S,H | 1 |
| CPC | Rd,Rr | Compare with Carry | Rd - Rr - C | | | Z,C,N,V,S,H | 1 |
| CPI | Rd,K | Compare with Immediate | Rd - K | | | Z,C,N,V,S,H | 1 |
| SBRC | Rr, b | Skip if Bit in Register Cleared | if (Rr(b) = 0) PC | ← | PC + 2 or 3 | None | 1 / 2 / 3 |
| SBRS | Rr, b | Skip if Bit in Register Set | if (Rr(b) = 1) PC | ← | PC + 2 or 3 | None | 1 / 2 / 3 |
| SBIC | A, b | Skip if Bit in I/O Register Cleared | if (I/O(A,b) = 0) PC | ← | PC + 2 or 3 | None | 1 / 2 / 3 |
| SBIS | A, b | Skip if Bit in I/O Register Set | If (I/O(A,b) =1) PC | ← | PC + 2 or 3 | None | 1 / 2 / 3 |
| 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 |

| Mnemonic | Operands | Description | | Op | | Flags | #Clocks |
|---|---|---|---|---|---|---|---|
| BRPL | k | Branch if Plus | if (N = 0) then PC | ← | PC + k + 1 | None | 1 / 2 |
| BRGE | k | Branch if Greater or Equal, Signed | if (N ⊕ V= 0) then PC | ← | PC + k + 1 | None | 1 / 2 |
| BRLT | k | Branch if Less Than, Signed | if (N ⊕ V= 1) then PC | ← | PC + k + 1 | None | 1 / 2 |
| BRHS | k | Branch if Half Carry Flag Set | if (H = 1) then PC | ← | PC + k + 1 | None | 1 / 2 |
| BRHC | k | Branch if Half Carry Flag Cleared | if (H = 0) then PC | ← | PC + k + 1 | None | 1 / 2 |
| BRTS | k | Branch if T Flag Set | if (T = 1) then PC | ← | PC + k + 1 | None | 1 / 2 |
| BRTC | k | Branch if T Flag Cleared | if (T = 0) then PC | ← | PC + k + 1 | None | 1 / 2 |
| BRVS | k | Branch if Overflow Flag is Set | if (V = 1) then PC | ← | PC + k + 1 | None | 1 / 2 |
| BRVC | k | Branch if Overflow Flag is Cleared | if (V = 0) then PC | ← | PC + k + 1 | None | 1 / 2 |
| BRIE | k | Branch if Interrupt Enabled | if (I = 1) then PC | ← | PC + k + 1 | None | 1 / 2 |
| BRID | k | Branch if Interrupt Disabled | if (I = 0) then PC | ← | PC + k + 1 | None | 1 / 2 |

### Table 36-3. Data Transfer Instructions

| Mnemonic | Operands | Description | | Op | | Flags | #Clocks |
|---|---|---|---|---|---|---|---|
| MOV | Rd, Rr | Copy Register | Rd | ← | Rr | None | 1 |
| MOVW | Rd, Rr | Copy Register Pair | Rd+1:Rd | ← | Rr+1:Rr | None | 1 |
| LDI | Rd, K | Load Immediate | Rd | ← | K | None | 1 |
| LDS | Rd, k | Load Direct from data space | Rd | ← | (k) | None | 3(1) |
| LD | Rd, X | Load Indirect | Rd | ← | (X) | None | 2(1) |
| LD | Rd, X+ | Load Indirect and Post-Increment | Rd<br>X | ←<br>← | (X)<br>X + 1 | None | 2(1) |
| LD | Rd, -X | Load Indirect and Pre-Decrement | X<br>Rd | ←<br>← | X - 1<br>(X) | None | 2(1) |
| LD | Rd, Y | Load Indirect | Rd | ← | (Y) | None | 2(1) |
| LD | Rd, Y+ | Load Indirect and Post-Increment | Rd<br>Y | ←<br>← | (Y)<br>Y + 1 | None | 2(1) |
| LD | Rd, -Y | Load Indirect and Pre-Decrement | Y<br>Rd | ←<br>← | Y - 1<br>(Y) | None | 2(1) |
| LDD | Rd, Y+q | Load Indirect with Displacement | Rd | ← | (Y + q) | None | 2(1) |
| LD | Rd, Z | Load Indirect | Rd | ← | (Z) | None | 2(1) |
| LD | Rd, Z+ | Load Indirect and Post-Increment | Rd<br>Z | ←<br>← | (Z)<br>Z+1 | None | 2(1) |
| LD | Rd, -Z | Load Indirect and Pre-Decrement | Z<br>Rd | ←<br>← | Z - 1<br>(Z) | None | 2(1) |

| Mnemonic | Operands | Description | | Op | | Flags | #Clocks |
|---|---|---|---|---|---|---|---|
| LDD | Rd, Z+q | Load Indirect with Displacement | Rd | ← | (Z + q) | None | 2(1) |
| STS | k, Rr | Store Direct to Data Space | (k) | ← | Rd | None | 2(1) |
| ST | X, Rr | Store Indirect | (X) | ← | Rr | None | 1(1) |
| ST | X+, Rr | Store Indirect and Post-Increment | (X)<br>X | ←<br>← | Rr<br>X + 1 | None | 1(1) |
| ST | -X, Rr | Store Indirect and Pre-Decrement | X<br>(X) | ←<br>← | X - 1<br>Rr | None | 1(1) |
| ST | Y, Rr | Store Indirect | (Y) | ← | Rr | None | 1(1) |
| ST | Y+, Rr | Store Indirect and Post-Increment | (Y)<br>Y | ←<br>← | Rr<br>Y + 1 | None | 1(1) |
| ST | -Y, Rr | Store Indirect and Pre-Decrement | Y<br>(Y) | ←<br>← | Y - 1<br>Rr | None | 1(1) |
| STD | Y+q, Rr | Store Indirect with Displacement | (Y + q) | ← | Rr | None | 1(1) |
| ST | Z, Rr | Store Indirect | (Z) | ← | Rr | None | 1(1) |
| ST | Z+, Rr | Store Indirect and Post-Increment | (Z)<br>Z | ←<br>← | Rr<br>Z + 1 | None | 1(1) |
| ST | -Z, Rr | Store Indirect and Pre-Decrement | Z | ← | Z - 1 | None | 1(1) |
| STD | Z+q,Rr | Store Indirect with Displacement | (Z + q) | ← | Rr | None | 1(1) |
| LPM | | Load Program Memory | R0 | ← | (Z) | None | 3 |
| LPM | Rd, Z | Load Program Memory | Rd | ← | (Z) | None | 3 |
| LPM | Rd, Z+ | Load Program Memory and Post-Increment | Rd<br>Z | ←<br>← | (Z)<br>Z + 1 | None | 3 |
| IN | Rd, A | In From I/O Location | Rd | ← | I/O(A) | None | 1 |
| OUT | A, Rr | Out To I/O Location | I/O(A) | ← | Rr | None | 1 |
| PUSH | Rr | Push Register on Stack | STACK | ← | Rr | None | 1 |
| POP | Rd | Pop Register from Stack | Rd | ← | STACK | None | 2 |

**Table 36-4. Bit and Bit-Test Instructions**

| Mnemonic | Operands | Description | | Op | | Flags | #Clocks |
|---|---|---|---|---|---|---|---|
| LSL | Rd | Logical Shift Left | Rd(n+1)<br>Rd(0)<br>C | ←<br>←<br>← | Rd(n)<br>0<br>Rd(7) | Z,C,N,V,H | 1 |
| LSR | Rd | Logical Shift Right | Rd(n)<br>Rd(7)<br>C | ←<br>←<br>← | Rd(n+1)<br>0<br>Rd(0) | Z,C,N,V | 1 |

| Mnemonic | Operands | Description | | Op | | Flags | #Clocks |
|---|---|---|---|---|---|---|---|
| ROL | Rd | Rotate Left Through Carry | Rd(0)<br>Rd(n+1)<br>C | ←<br>←<br>← | C<br>Rd(n)<br>Rd(7) | Z,C,N,V,H | 1 |
| ROR | Rd | Rotate Right Through Carry | Rd(7)<br>Rd(n)<br>C | ←<br>←<br>← | C<br>Rd(n+1)<br>Rd(0) | Z,C,N,V | 1 |
| ASR | Rd | Arithmetic Shift Right | Rd(n) | ← | Rd(n+1), n=0..6 | Z,C,N,V | 1 |
| SWAP | Rd | Swap Nibbles | Rd(3..0) | ↔ | Rd(7..4) | None | 1 |
| SBI | A, b | Set Bit in I/O Register | I/O(A, b) | ← | 1 | None | 1 |
| CBI | A, b | Clear Bit in I/O Register | I/O(A, b) | ← | 0 | None | 1 |
| BST | Rr, b | Bit Store from Register to T | T | ← | Rr(b) | T | 1 |
| BLD | Rd, b | Bit load from T to Register | Rd(b) | ← | T | None | 1 |
| BSET | s | Flag Set | SREG(s) | ← | 1 | SREG(s) | 1 |
| BCLR | s | Flag Clear | SREG(s) | ← | 0 | SREG(s) | 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 |

© 2017 Microchip Technology Inc.

**Table 36-5. MCU Control Instructions**

| Mnemonic | Operands | Description | Operation | Flags | #Clocks |
|---|---|---|---|---|---|
| BREAK | | Break | (See also in Debug interface description) | None | 1 |
| NOP | | No Operation | | None | 1 |
| SLEEP | | Sleep | (see also power management and sleep description) | None | 1 |
| WDR | | Watchdog Reset | (see also Watchdog Controller description) | None | 1 |

**Note:**

1. Cycle time for data memory accesses assume internal RAM access, and are not valid for accesses through the NVM controller. A minimum of one extra cycle must be added when accessing memory through the NVM controller (such as Flash and EEPROM), but depending on simultaneous accesses by other masters or the NVM controller state, there may be more than one extra cycle.

# 37. Conventions

## 37.1 Numerical Notation

**Table 37-1. Numerical Notation**

| Symbol | Description |
|---|---|
| 165 | Decimal number |
| 0b0101 | Binary number (example 0b0101 = 5 decimal) |
| '0101' | Binary numbers are given without prefix if unambiguous. |
| 0x3B24 | Hexadecimal number |
| X | Represents an unknown or don't care value |
| Z | Represents a high-impedance (floating) state for either a signal or a bus |

## 37.2 Memory Size and Type

**Table 37-2. Memory Size and Bit Rate**

| Symbol | Description |
|---|---|
| KB (kbyte) | kilobyte ($2^{10}$ = 1024) |
| MB (Mbyte) | megabyte ($2^{20}$ = 1024*1024) |
| GB (Gbyte) | gigabyte ($2^{30}$ = 1024*1024*1024) |
| b | bit (binary '0' or '1') |
| B | byte (8 bits) |
| 1kbit/s | 1,000 bit/s rate (not 1,024 bit/s) |
| 1Mbit/s | 1,000,000 bit/s rate |
| 1Gbit/s | 1,000,000,000 bit/s rate |
| word | 16 bit |

## 37.3 Frequency and Time

| Symbol | Description |
|---|---|
| kHz | 1kHz = $10^3$Hz = 1,000Hz |
| KHz | 1KHz = 1,024Hz, 32KHz = 32,768Hz |
| MHz | $10^6$ = 1,000,000Hz |
| GHz | $10^9$ = 1,000,000,000Hz |

| Symbol | Description |
|---|---|
| s | second |
| ms | millisecond |
| µs | microsecond |
| ns | nanosecond |

## 37.4 Registers and Bits

**Table 37-3. Register and Bit Mnemonics**

| Symbol | Description |
|---|---|
| R/W | Read/Write accessible register bit. The user can read from and write to this bit. |
| R | Read-only accessible register bit. The user can only read this bit. Writes will be ignored. |
| W | Write-only accessible register bit. The user can only write this bit. Reading this bit will return an undefined value. |
| BIT | Bit names are shown in uppercase. (Example ENABLE) |
| FIELD[n:m] | A set of bits from bit n down to m. (Example: PINA[3:0] = {PINA3, PINA2, PINA1, PINA0} |
| Reserved | Reserved bits are unused and reserved for future use. For compatibility with future devices, always write reserved bits to zero when the register is written. Reserved bits will always return zero when read.<br>Reserved bit field values must not be written to a bit field. A reserved value won't be read from a read-only bit field. |
| PERIPHERAL*i* | If several instances of a peripheral exist, the peripheral name is followed by a number to indicate the number of the instance in the range 0-n. PERIPHERAL0 denotes one specific instance. |
| Reset | Value of a register after a power Reset. This is also the value of registers in a peripheral after performing a software Reset of the peripheral, except for the Debug Control registers. |
| SET/CLR | Registers with SET/CLR suffix allows the user to clear and set bits in a register without doing a read-modify-write operation. These registers always come in pairs. Writing a one to a bit in the CLR register will clear the corresponding bit in both registers, while writing a one to a bit in the SET register will set the corresponding bit in both registers. Both registers will return the same value when read. If both registers are written simultaneously, the write to the CLR register will take precedence. |

# 38. Acronyms and Abbreviations

The below table contains acronyms and abbreviations used in this document.

**Table 38-1. Acronyms and Abbreviations**

| Abbreviation | Description |
|---|---|
| AC | Analog Comparator |
| ACK | Acknowledge |
| ADC | Analog-to-Digital Converter |
| ADDR | Address |
| AES | Advanced Encryption Standard |
| ALU | Arithmetic Logic Unit |
| AREF | Analog reference voltage, also VREFA |
| BLB | Boot Lock Bit |
| BOD | Brown-out detector |
| CAL | Calibration |
| CCMP | Compare/Capture |
| CCL | Configurable Custom Logic |
| CCP | Configuration Change Protection |
| CLK | Clock |
| CLKCTRL | Clock Controller |
| CRC | Cyclic Redundancy Check |
| CTRL | Control |
| DAC | Digital-to-Analog Converter |
| DFLL | Digital Frequency Locked Loop |
| EEPROM | Electrically Erasable Programmable Read-Only Memory |
| EVSYS | Event System |
| GND | Ground |
| GPIO | General Purpose Input/Output |
| $I^2C$ | Inter-Integrated Circuit |
| IF | Interrupt flag |
| INT | Interrupt |
| IrDA | Infrared Data Association |
| IVEC | Interrupt Vector |
| LUT | Look-Up Table |

| Abbreviation | Description |
|---|---|
| MBIST | Memory built-in self-test |
| NACK | Not Acknowlege |
| NMI | Non-maskable interrupt |
| NVM | Non-Volatile Memory |
| NVMCTRL | Non-Volatile Memory Controller |
| OSC | Oscillator |
| PC | Program Counter |
| PER | Period |
| POR | Power-on reset |
| PORT | I/O Pin Configuration |
| PWM | Pulse Width Modulation |
| RAM | Random-Access Memory |
| REF | Reference |
| REQ | Request |
| RISC | Reduced Instruction Set Computer |
| RSTCTRL | Reset Controller |
| RTC | Real-Time Counter |
| RX | Receiver/Receive |
| SERCOM | Serial Communication Interface |
| SLPCTRL | Sleep Controller |
| SMBus™ | System Management Bus |
| SP | Stack Pointer |
| SPI | Serial Peripheral Interface |
| SRAM | Static Random-Access Memory |
| SYSCFG | System Configuration |
| TC | Timer/Counter (Optionally superseded by a letter indicating type of TC) |
| TRNG | True Random Number Generator |
| TWI | Two Wire Interface |
| TX | Transmitter/Transmit |
| ULP | Ultra-low power |
| UPDI | Unified Program and Debug Interface |
| USART | Universal Synchronous and Asynchronous Serial Receiver and Transmitter |

| Abbreviation | Description |
|---|---|
| $V_{DD}$ | Voltage to be applied to VDD |
| VREF | Voltage reference |
| WDT | Watchdog Timer |
| XOSC | Crystal Oscillator |

# 39. Errata

## 39.1 Errata - ATtiny212/ATtiny412

### 39.1.1 Die Revision A

#### 39.1.1.1 AC

**1 –**

**False triggers may occur under certain conditions.**

False triggers may occur on falling input pin:

- For common mode voltage below 0.5V
- For common mode voltage above 0.5V if slew rate is greater than 1V/us

**Fix/Workaround:**

None.

**2 –**

**AC interrupt flag not set unless interrupt is enabled**

ACn.STATUS.CMP is not set if the ACn.INTCTRL.CMP is not set.

**Fix/Workaround:**

Enable ACn.INTCTRL.CMP or use ACn.STATUS.STATE for polling.

#### 39.1.1.2 ADC

**1 –**

**One extra measurement performed after disabling ADC free running mode**

The ADC may perform one additional measurement after clearing ADCn.CTRLA.FREERUN

**Fix/Workaround:**

Write ADCn.CTRLA.ENABLE to zero to stop the free running mode immediately.

**2 –**

**Changing ADC control bits during free running mode not working**

If control signals are changed during free running mode, the new configuration is not properly taken into account in the next measurement. This is valid for registers ADC.CTRLB, ADC.CTRLC, ADC.SAMPCTRL and ADC.MUXPOS, ADC.WINLT and ADC.WINHT.

**Fix/Workaround:**

Disable ADC free running mode before update of ADC.CTRLB, ADC.CTRLC, ADC.SAMPCTRL, ADC.MUXPOS, ADC.WINLT or ADC.WINHT.

**3 –**

**SAMPDLY and ASDV does not work together with SAMPLEN**

Using SAMPCTRL.SAMPLEN at the same time as CTRLD.SAMPDLY or CTRLD.ASDV will cause an unpredictable sampling length.

**Fix/Workaround:**

When setting SAMPCTRL.SAMPLEN greater than zero the CTRLD.SAMPDLY and CTRLD.ASDV must be cleared.

#### 39.1.1.3 TCB

**1 –**

**TCB interrupt flag cleared when reading CCMPH**

TCBn.INTFLAGS.CAPT is cleared when reading TCBn.CCMPH instead of CCMPL.

**Fix/Workaround:**

Read both TCBn.CCMPL and TCBn.CCMPH.

**2 –**

**TCB input capture frequency and pulse-width measurement mode not working with prescaled clock**

The TCB input capture frequency and pulse-width measurement mode may lock to freeze state if CLKSEL in TCB.CTRLA is set to other value than 0x0.

**Fix/Workaround:**

Only use CLKSEL equal to 0x0 when using Input capture frequency and pulse-width measurement mode.

#### 39.1.1.4 TWI

**1 –**

**TIMEOUT bits in TWI.MCTRLB register is not accessible**

The TIMEOUT bits in TWI.MCTRLB register is not accessible from software.

**Fix/Workaround:**

When initializing TWI, BUSSTATE in TWI.MSTATUS should be brought into IDLE state by writing 0x1 to it.

**2 –**

**TWI Smart Mode gives extra clock pulse**

TWI Master with Smart Mode enabled gives an extra clock pulse on SCL line after sending NACK.

**Fix/Workaround:**

None.

**3 –**

**TWI Master Mode wrongly detect start bit as a stop bit**

If TWI is enabled in master mode followed by an immediate write to the MADDR register, the bus monitor recognize the start bit as a stop bit.

**Fix/Workaround:**

Wait minimum two clock cycles from TWI.MCTRLA.ENABLE until TWI.MADDR is written.

**4 –**

**TWI Master Enable Quick Command not accessible**

TWI.MCTRLA.QCEN is not accessible from software.

**Fix/Workaround:**

None.

#### 39.1.1.5 USART

**1 –**

**Frame error on previous message may cause false start bit detection**

If receiving a frame with RXDATAH.FERR set and reading the RXDATAL before the RxD line goes high, will trigger a false start bit detection.

**Fix/Workaround:**

Wait for the RxD pin to go high before reading RXDATA, for instance by polling the bit in PORTn.IN where the RxD pin is located.

# 40. Datasheet Revision History

**Note:** The datasheet revision is independent of the die revision and the device variant (last letter of the ordering number).

## 40.1 Rev. A - 01/2017

Initial release.

## The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

## Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

## Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

## Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.

© 2017 Microchip Technology Inc.

- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

## Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

## Trademarks

The Microchip name and logo, the Microchip logo, AnyRate, AVR, AVR logo, AVR Freaks, BeaconThings, BitCloud, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, Heldo, JukeBlox, KeeLoq, KeeLoq logo, Kleer, LANCheck, LINK MD, maXStylus, maXTouch, MediaLB, megaAVR, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, Prochip Designer, QTouch, RightTouch, SAM-BA, SpyNIC, SST, SST Logo, SuperFlash, tinyAVR, UNI/O, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

ClockWorks, The Embedded Control Solutions Company, EtherSynch, Hyper Speed Control, HyperLight Load, IntelliMOS, mTouch, Precision Edge, and Quiet-Wire are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BodyCom, chipKIT, chipKIT logo, CodeGuard, CryptoAuthentication, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, Mindi, MiWi, motorBench, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PureSilicon, QMatrix, RightTouch logo, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2017, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-1687-6

## Quality Management System Certified by DNV

**ISO/TS 16949**

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

© 2017 Microchip Technology Inc.

# Worldwide Sales and Service

| AMERICAS | ASIA/PACIFIC | ASIA/PACIFIC | EUROPE |
|---|---|---|---|
| **Corporate Office**<br>2355 West Chandler Blvd.<br>Chandler, AZ 85224-6199<br>Tel: 480-792-7200<br>Fax: 480-792-7277<br>Technical Support:<br>http://www.microchip.com/support<br>Web Address:<br>www.microchip.com<br>**Atlanta**<br>Duluth, GA<br>Tel: 678-957-9614<br>Fax: 678-957-1455<br>**Austin, TX**<br>Tel: 512-257-3370<br>**Boston**<br>Westborough, MA<br>Tel: 774-760-0087<br>Fax: 774-760-0088<br>**Chicago**<br>Itasca, IL<br>Tel: 630-285-0071<br>Fax: 630-285-0075<br>**Dallas**<br>Addison, TX<br>Tel: 972-818-7423<br>Fax: 972-818-2924<br>**Detroit**<br>Novi, MI<br>Tel: 248-848-4000<br>**Houston, TX**<br>Tel: 281-894-5983<br>**Indianapolis**<br>Noblesville, IN<br>Tel: 317-773-8323<br>Fax: 317-773-5453<br>Tel: 317-536-2380<br>**Los Angeles**<br>Mission Viejo, CA<br>Tel: 949-462-9523<br>Fax: 949-462-9608<br>Tel: 951-273-7800<br>**Raleigh, NC**<br>Tel: 919-844-7510<br>**New York, NY**<br>Tel: 631-435-6000<br>**San Jose, CA**<br>Tel: 408-735-9110<br>Tel: 408-436-4270<br>**Canada - Toronto**<br>Tel: 905-695-1980<br>Fax: 905-695-2078 | **Asia Pacific Office**<br>Suites 3707-14, 37th Floor<br>Tower 6, The Gateway<br>Harbour City, Kowloon<br>**Hong Kong**<br>Tel: 852-2943-5100<br>Fax: 852-2401-3431<br>**Australia - Sydney**<br>Tel: 61-2-9868-6733<br>Fax: 61-2-9868-6755<br>**China - Beijing**<br>Tel: 86-10-8569-7000<br>Fax: 86-10-8528-2104<br>**China - Chengdu**<br>Tel: 86-28-8665-5511<br>Fax: 86-28-8665-7889<br>**China - Chongqing**<br>Tel: 86-23-8980-9588<br>Fax: 86-23-8980-9500<br>**China - Dongguan**<br>Tel: 86-769-8702-9880<br>**China - Guangzhou**<br>Tel: 86-20-8755-8029<br>**China - Hangzhou**<br>Tel: 86-571-8792-8115<br>Fax: 86-571-8792-8116<br>**China - Hong Kong SAR**<br>Tel: 852-2943-5100<br>Fax: 852-2401-3431<br>**China - Nanjing**<br>Tel: 86-25-8473-2460<br>Fax: 86-25-8473-2470<br>**China - Qingdao**<br>Tel: 86-532-8502-7355<br>Fax: 86-532-8502-7205<br>**China - Shanghai**<br>Tel: 86-21-3326-8000<br>Fax: 86-21-3326-8021<br>**China - Shenyang**<br>Tel: 86-24-2334-2829<br>Fax: 86-24-2334-2393<br>**China - Shenzhen**<br>Tel: 86-755-8864-2200<br>Fax: 86-755-8203-1760<br>**China - Wuhan**<br>Tel: 86-27-5980-5300<br>Fax: 86-27-5980-5118<br>**China - Xian**<br>Tel: 86-29-8833-7252<br>Fax: 86-29-8833-7256 | **China - Xiamen**<br>Tel: 86-592-2388138<br>Fax: 86-592-2388130<br>**China - Zhuhai**<br>Tel: 86-756-3210040<br>Fax: 86-756-3210049<br>**India - Bangalore**<br>Tel: 91-80-3090-4444<br>Fax: 91-80-3090-4123<br>**India - New Delhi**<br>Tel: 91-11-4160-8631<br>Fax: 91-11-4160-8632<br>**India - Pune**<br>Tel: 91-20-3019-1500<br>**Japan - Osaka**<br>Tel: 81-6-6152-7160<br>Fax: 81-6-6152-9310<br>**Japan - Tokyo**<br>Tel: 81-3-6880- 3770<br>Fax: 81-3-6880-3771<br>**Korea - Daegu**<br>Tel: 82-53-744-4301<br>Fax: 82-53-744-4302<br>**Korea - Seoul**<br>Tel: 82-2-554-7200<br>Fax: 82-2-558-5932 or<br>82-2-558-5934<br>**Malaysia - Kuala Lumpur**<br>Tel: 60-3-6201-9857<br>Fax: 60-3-6201-9859<br>**Malaysia - Penang**<br>Tel: 60-4-227-8870<br>Fax: 60-4-227-4068<br>**Philippines - Manila**<br>Tel: 63-2-634-9065<br>Fax: 63-2-634-9069<br>**Singapore**<br>Tel: 65-6334-8870<br>Fax: 65-6334-8850<br>**Taiwan - Hsin Chu**<br>Tel: 886-3-5778-366<br>Fax: 886-3-5770-955<br>**Taiwan - Kaohsiung**<br>Tel: 886-7-213-7830<br>**Taiwan - Taipei**<br>Tel: 886-2-2508-8600<br>Fax: 886-2-2508-0102<br>**Thailand - Bangkok**<br>Tel: 66-2-694-1351<br>Fax: 66-2-694-1350 | **Austria - Wels**<br>Tel: 43-7242-2244-39<br>Fax: 43-7242-2244-393<br>**Denmark - Copenhagen**<br>Tel: 45-4450-2828<br>Fax: 45-4485-2829<br>**Finland - Espoo**<br>Tel: 358-9-4520-820<br>**France - Paris**<br>Tel: 33-1-69-53-63-20<br>Fax: 33-1-69-30-90-79<br>**France - Saint Cloud**<br>Tel: 33-1-30-60-70-00<br>**Germany - Garching**<br>Tel: 49-8931-9700<br>**Germany - Haan**<br>Tel: 49-2129-3766400<br>**Germany - Heilbronn**<br>Tel: 49-7131-67-3636<br>**Germany - Karlsruhe**<br>Tel: 49-721-625370<br>**Germany - Munich**<br>Tel: 49-89-627-144-0<br>Fax: 49-89-627-144-44<br>**Germany - Rosenheim**<br>Tel: 49-8031-354-560<br>**Israel - Ra'anana**<br>Tel: 972-9-744-7705<br>**Italy - Milan**<br>Tel: 39-0331-742611<br>Fax: 39-0331-466781<br>**Italy - Padova**<br>Tel: 39-049-7625286<br>**Netherlands - Drunen**<br>Tel: 31-416-690399<br>Fax: 31-416-690340<br>**Norway - Trondheim**<br>Tel: 47-7289-7561<br>**Poland - Warsaw**<br>Tel: 48-22-3325737<br>**Romania - Bucharest**<br>Tel: 40-21-407-87-50<br>**Spain - Madrid**<br>Tel: 34-91-708-08-90<br>Fax: 34-91-708-08-91<br>**Sweden - Gothenberg**<br>Tel: 46-31-704-60-40<br>**Sweden - Stockholm**<br>Tel: 46-8-5090-4654<br>**UK - Wokingham**<br>Tel: 44-118-921-5800<br>Fax: 44-118-921-5820 |

# Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

Microchip:

ATTINY212-SSNR ATTINY212-SSFR ATTINY412-SSNR ATTINY412-SSFR ATTINY212-SSF ATTINY212-SSN ATTINY412-SSF ATTINY412-SSN

Общество с ограниченной ответственностью «МосЧип» ИНН 7719860671 / КПП 771901001
Адрес: 105318, г.Москва, ул.Щербаковская д.3, офис 1107

**Данный компонент на территории Российской Федерации**

**Вы можете приобрести в компании MosChip.**

Для оперативного оформления запроса Вам необходимо перейти по данной ссылке:

http://moschip.ru/get-element

Вы можете разместить у нас заказ для любого Вашего проекта, будь то серийное производство или разработка единичного прибора.

В нашем ассортименте представлены ведущие мировые производители активных и пассивных электронных компонентов.

Нашей специализацией является поставка электронной компонентной базы двойного назначения, продукции таких производителей как XILINX, Intel (ex.ALTERA), Vicor, Microchip, Texas Instruments, Analog Devices, Mini-Circuits, Amphenol, Glenair.

Сотрудничество с глобальными дистрибьюторами электронных компонентов, предоставляет возможность заказывать и получать с международных складов практически любой перечень компонентов в оптимальные для Вас сроки.

На всех этапах разработки и производства наши партнеры могут получить квалифицированную поддержку опытных инженеров.

Система менеджмента качества компании отвечает требованиям в соответствии с ГОСТ Р ИСО 9001, ГОСТ РВ 0015-002 и ЭС РД 009

**Офис по работе с юридическими лицами:**

105318, г.Москва, ул.Щербаковская д.3, офис 1107, 1118, ДЦ «Щербаковский»

Телефон: +7 495 668-12-70 (многоканальный)

Факс: +7 495 668-12-70 (доб.304)

E-mail: info@moschip.ru

Skype отдела продаж:
moschip.ru
moschip.ru_4
moschip.ru_6
moschip.ru_9