MICROCHIP

Migration Guide

Migration from the megaAVR® to AVR® Dx Microcontroller Families

Introduction

Author: Cristian Pop, Microchip Technology Inc.

This document will help application designers familiar with the megaAVR® families to migrate to the AVR® Dx MCU families, covering both differences and similarities. The comparison is applicable for most of the megaAVR vs. the AVR Dx products but, in this document, the focus is on the ATmega128 and AVR128DA64, two generations of 128 KB Flash MCUs available in 64-pin packages.

Most of the AVR Dx peripherals are functionally compatible with the megaAVR peripherals (including WDT, RTC, AC, ADC, SPI, USART, TWI, and Timers), but updates to the source code will be required when migrating. The following sections provide details on a few updates, but the migrated code must be fully tested to ensure the target application's intended behavior is the same. The megaAVR and AVR Dx families are not pin-to-pin compatible.

For the AVR Dx family, the names of the pins are the same, but their position has changed from the megaAVR family. For more details, see the *Pin Configurations* and *Pinout* sections, respectively, in the data sheet of each device.

Table of Contents

1. Relevant Devices

This section lists the relevant devices for this document. The following figures show the different family devices, laying out pin count variants and memory sizes:

- Vertical migration upwards is possible without code modification, as these devices are pin-compatible and provide the same or more features
- Horizontal migration to the left reduces the pin count and, therefore, the available features
- Devices with different Flash memory sizes typically also have different SRAM and EEPROM

Figure 1-1. megaAVR® Family Overview

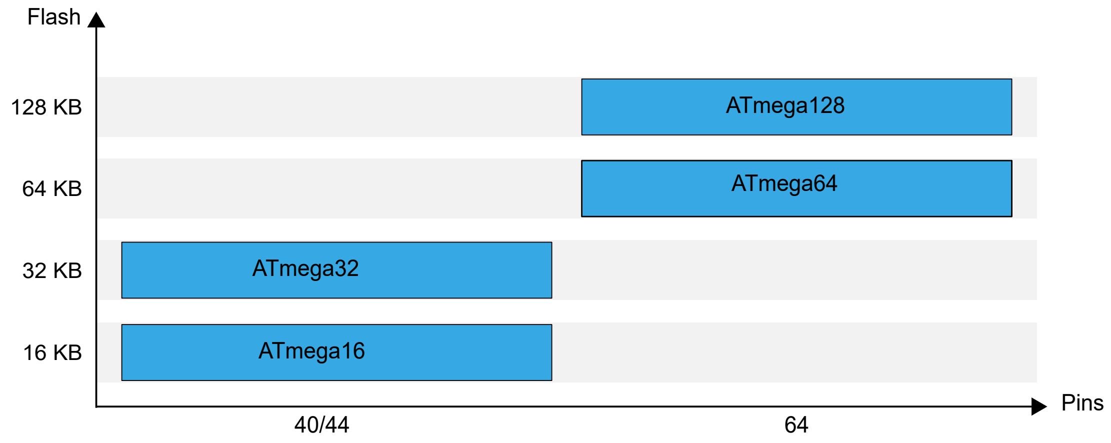


Figure 1-2. AVR® DA Family Overview

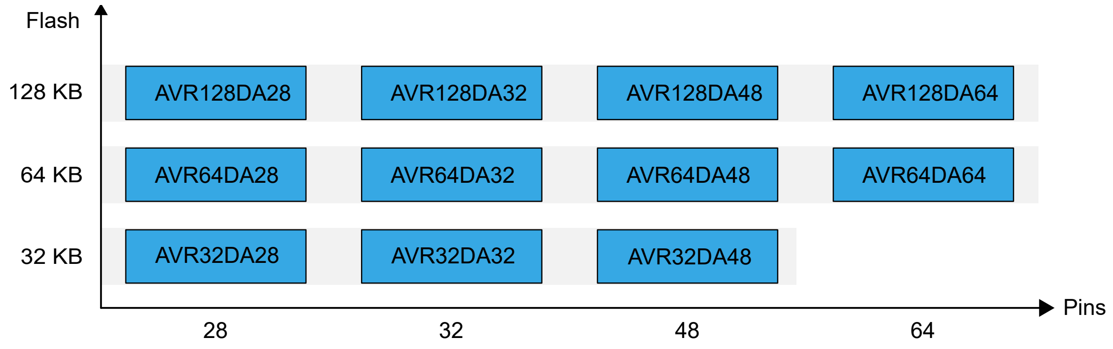
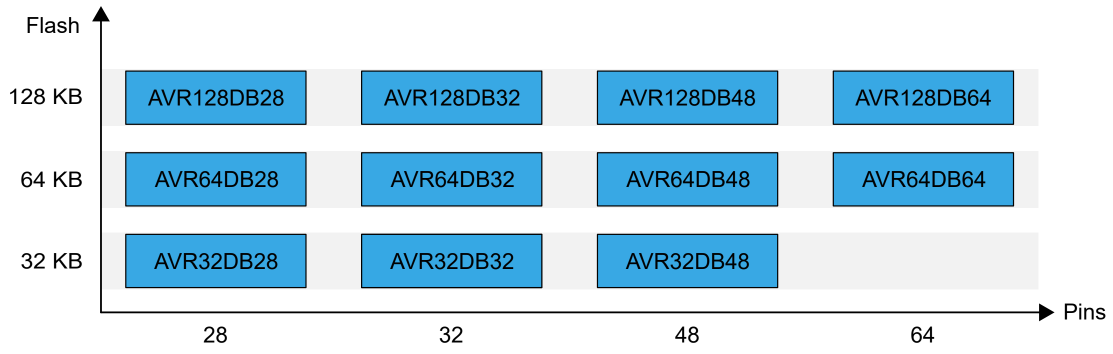


Figure 1-3. AVR® DB Family Overview



2. Common Peripherals

2.1 Common Peripherals

The following table shows the list of peripherals that are present in both families:

Peripherals/Features	ATmega16	ATmega32	ATmega64	ATmega128	AVR® DA Family	AVR® DB Family
Maximum Frequency	16 MHz (V_{DD} ≥ 4.5V)	16 MHz (V_{DD} ≥ 4.5V)	16 MHz (V_{DD} ≥ 4.5V)	16 MHz (V_{DD} ≥ 4.5V)	24 MHz	24 MHz
Supply voltage (V_{DD})	1.8–5.5V	1.8–5.5V	1.8–5.5V	1.8–5.5V	1.8–5.5V	1.8–5.5V
Package (pin number)	40, 44	40, 44	64	64	28, 32, 48, 64	28, 32, 48, 64
Flash memory **(Read-while-write section)**	16 KB (14 KB)	32 KB (28 KB)	64 KB (56 KB)	128 KB (120 KB)	32-128 KB -	32-128 KB -
RAM memory	1 KB	2 KB	4 KB	4 KB	4-16 KB	4-16 KB
EEPROM	512 bytes	1 KB	2 KB	4 KB	512 bytes	512 bytes
GPIO	32	32	53	53	51	51
USART	1	1	2	2	5	5
SPI	1	1	1	1	2	2
TWI (I^2C)	1	1	1	1	2	2
8-bit Timer/Counter	2	2	2	2	-(1)	-(1)
12-bit Timer/Counter	-	-	-	-	1	1
16-bit Timer/Counter	1	1	1	1	7(1)	7(1)
Real Time Counter (RTC)	1	1	1	1	1	1
Analog Comparator (AC)	1	1	1	1	3	3
Analog-to-Digital Converter (ADC)	1	1	1	1	1	1
Watchdog Timer (WDT)	1	1	1	1	1	1

Note:

1. For AVR Dx, each TCA in the Split mode works as two separate 8-bit timers, each of them having three compare channels for PWM generation.

2.2 System

2.2.1 CPU

The megaAVR and AVR Dx families are based on the well-recognized AVR CPU, but the AVR Dx benefits from an updated version of the classic core. The megaAVR instruction set is fully supported by the AVR Dx core, and the execution time is improved. The differences are handled by a C/C++ compiler, but the users that are porting time-critical or cycle-accurate code must refer to the *AVR® Instruction Set Manual* to verify that the execution time is within the accepted range.

Note: For details about the instruction set and execution time, refer to the *AVR® Instruction Set Manual*. Refer to the CPU version called *AVRxt* for details regarding the AVR Dx devices and the version called *AVR* for megaAVR devices.

2.2.2 Interrupts

Both megaAVR and AVR Dx families offer a comprehensive interrupt system, with a configurable interrupt vector table, but the architecture and features are different between families. The interrupt vectors and flags are not identical in both families, even if they are used by the same peripheral.

For the megaAVR devices, the Reset vector can be placed either at address 0x0000 or Boot Reset address, selectable using Boot Reset Fuse (BOOTRST), and Interrupt Vector Select (IVSEL) fuses. That allows the usage of different interrupt vector locations for the bootloader and the application, with the flexibility of commuting between them in software.

The interrupt vector table of the AVR Dx devices is always placed at address 0x0000 after Reset, even if the bootloader is used. That is possible because the bootloader section is placed at the beginning of the Flash area and not at the end, like for the megaAVR devices. The interrupt vector table is software re-mappable to the beginning of the application area using the IVSEL bit from the CPUINT.CTRLA register.

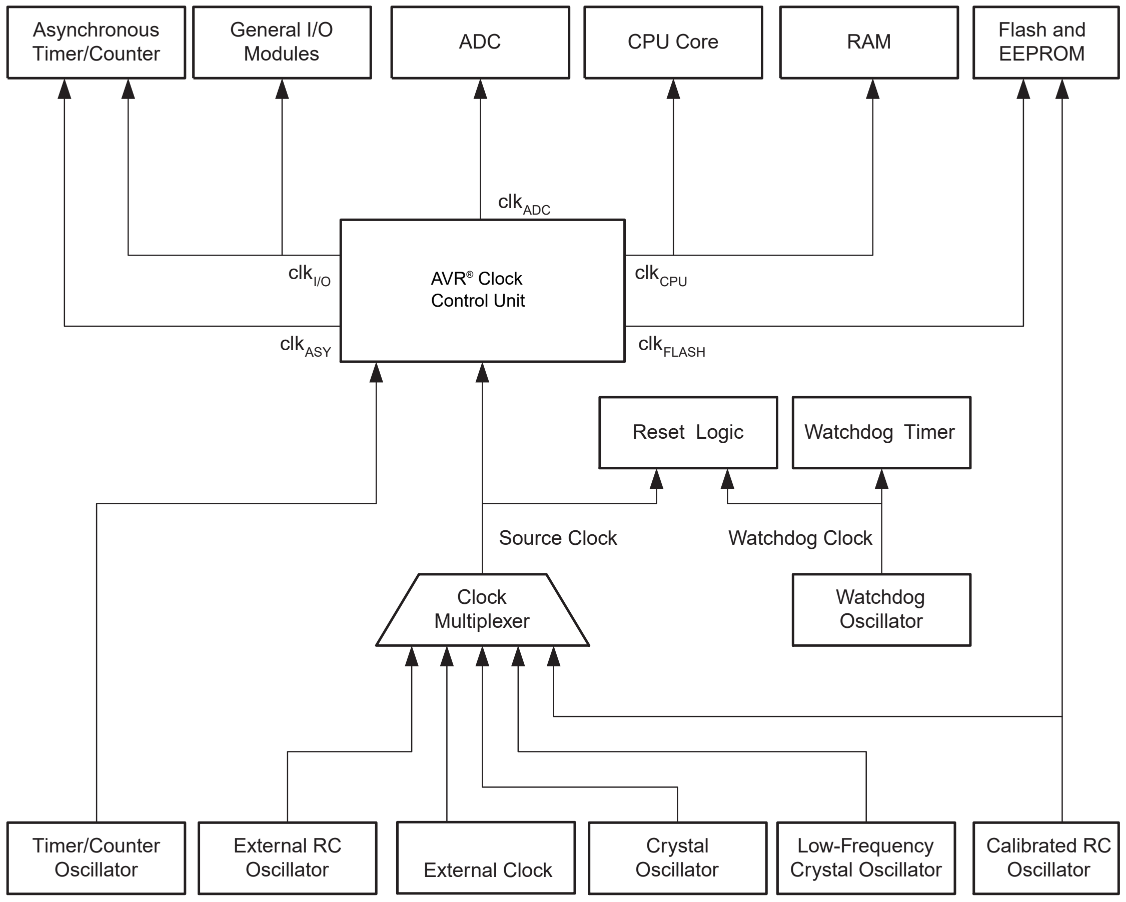
The interrupt vector table is always placed in the generated code, even if only a limited number of interrupts are used. To allow writing of compact code, the interrupt controller of the AVR Dx devices offers a selectable Compact Vector Table (CVT). This feature reduces the number of interrupt handlers and thereby frees up memory that can be used for the application code. When the CVT feature is used, all interrupts share the same interrupt vector number/ Interrupt Service Routine (ISR).

Additionally to the CVT feature, the AVR Dx interrupt controller offers a non-maskable interrupt for critical functions, one selectable high-priority interrupt, and an optional round robin scheduling scheme for normal priority interrupts.

2.2.3 Clock Sources

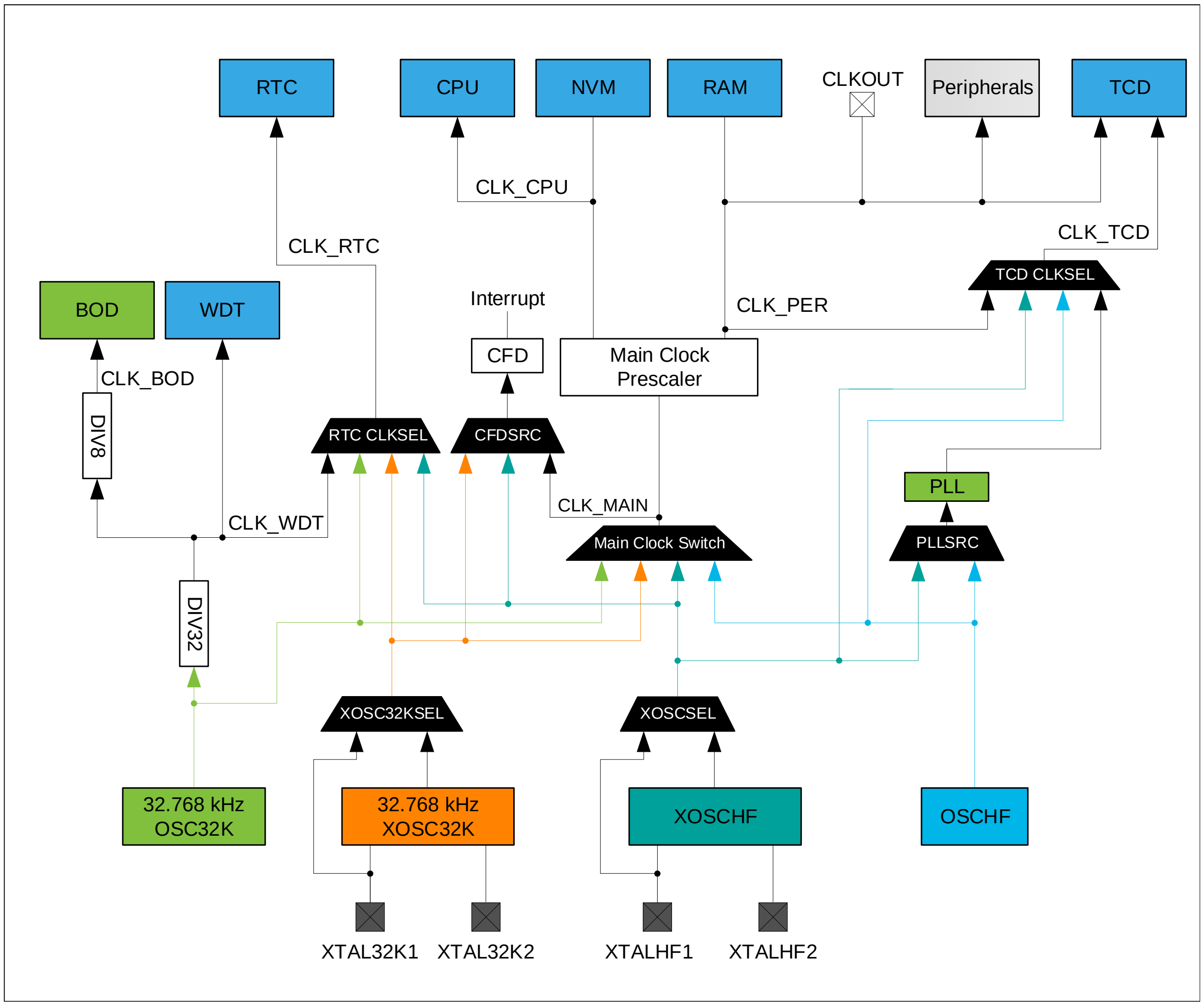
The megaAVR microcontrollers allow multiple clock options for system clock, selectable from the external RC oscillator, crystal oscillator, external clock, and calibrated internal RC oscillator. The clock source is selectable by fuses (CKOPT and CKSEL[3:0]) and cannot be changed by software.

Figure 2-1. ATmega128 Clock – Block Diagram



The AVR Dx family uses a different clock system compared to the megaAVR family. Even if the default oscillator is selected using fuses, this can be safely changed by software during normal operation. The frequency of the internal High-Frequency oscillator (OSCHF) is also selectable by software to 1, 2, 3, 4 MHz, and multiples of 4, up to 24 MHz. For more details, see the *CLKCTRL - Clock Controller* section from the device data sheet.

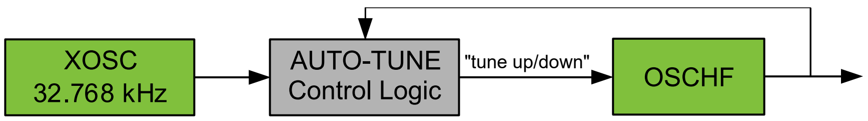
Figure 2-2. AVR® Dx Clock – Block Diagram



Additionally, the AVR Dx devices provide a Phase-Locked Loop (PLL) that allows clock multiplication by 2x or 3x (maximum frequency 48 MHz) and can serve as input for the Timer/Counter type D (TCD).

Not all AVR Dx devices provide the option for a High-Frequency crystal oscillator (XOSCHF), but the accuracy of the AVR Dx's internal High-Frequency oscillator (OSCHF) can be improved using the built-in auto-tune logic combined with the 32.768 kHz crystal oscillator. This is done by comparing the internal 1 MHz clock with a reference derived from the 32.768 kHz crystal:

Figure 2-3. OSCHF Auto-Tune Block Diagram



Note: For details about how to use the auto-tune feature, refer to *TB3234 - Internal High-Frequency Oscillator Calibration Using the Auto-Tune Feature*.

2.3 Memories

The AVR architectures have two main memory spaces: The Data Memory and the Program Memory. The CPU Data Memory space allows faster access and is accessible using `LD/ST` instructions in assembly, while the Program Memory space (code space) is accessible through `LPM/SPM` instructions.

For the megaAVR devices, the first 4352 bytes from Data Memory are reserved for the Register file, the I/O memory, the extended I/O memory, and the internal data SRAM. The first 32 locations are used to access the Register file, and the next 64 locations access the standard I/O memory, then 160 locations of extended I/O memory, and the next 4096 locations access the internal data SRAM.

For the megaAVR devices with at least 64 KB Flash memory, an external memory interface is available for interfacing with other devices. This external interface allows the access of an area in the remaining address locations up to the 64 KB address space. This area starts at the address following the internal SRAM.

For the AVR Dx devices, the peripherals, SRAM, EEPROM and the I/O registers are all located in the Data Memory space, while the Flash is located in the Program Memory space. The Flash memory can be also mapped into the Data Memory space in blocks of 32 KB.

Unlike the megaAVR devices, the AVR Dx devices do not have support for the external memory interface.

2.3.1 In-System Programmable Flash Memory and Bootloader Support

Both AVR families allow updates of the program memory using software. The on-chip in-system reprogrammable Flash memory is used mainly for program storage. For software security, the Flash Program Memory space is divided into sections with different access rights: megaAVR devices have two sections (the Boot section and Application Program section), while AVR Dx devices provide three configurable sections (Boot Code section, Application Code section, and Application Data section).

For the megaAVR devices, the size of the Boot section can be configured using BOOTSZ[1:0] fuses to any from the following values: 512, 1024, 2048 or 4096 words. This section is located at the end of the Flash memory.

For the AVR Dx devices, the size of the Boot section can be configured to any value with a granularity of 512 bytes using the Boot Size (BOOTSIZE) fuse, and it is located at the beginning of the Flash area. The remaining Flash can be split into an Application Program section and Application Data section using the CODESIZE fuse. For security reasons, the code executing from the Boot section can write the Application section or Data section, the code executing from the Application section can write only the Data section, and the code executing from the Data section cannot write to any Flash section. This mechanism prevents unintentional alteration of sections. Additional to this hardware mechanism, the device implements software options that can prevent an unwanted read of the boot code from other sections.

2.3.2 SRAM and EEPROM Memories

For both families, the SRAM memory is located in the Data space. The address offset and the size of the SRAM memory can be different from device to device (even in the same family). Refer to the device data sheet for details about available size and address offset.

For the megaAVR devices, the EEPROM is located in a separate data space and is accessible through the I/O space. See the following code examples:

Example 2-1. megaAVR® - EEPROM Read/Write Byte

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

The EEPROM is located in the Data space for the AVR Dx devices. That allows a linear addressing of the entire area and faster access using the `LD/ST` instructions:

Example 2-2. AVR® Dx - EEPROM Read/Write Byte

```
uint8_t FLASH_0_read_eeprom_byte(eeprom_adr_t eeprom_adr)
{
    /* Read operation will be stalled by hardware if any write is in progress 
*/
    return *(uint8_t *)(EEPROM_START + eeprom_adr);
}

nvmctrl_status_t FLASH_0_write_eeprom_byte(eeprom_adr_t eeprom_adr, uint8_t 
data)
{
    /* Wait for completion of previous operation */
    while (NVMCTRL.STATUS & (NVMCTRL_EEBUSY_bm | NVMCTRL_FBUSY_bm))
        ;

    /* Program the EEPROM with desired value(s) */
    ccp_write_spm((void *)&NVMCTRL.CTRLA, NVMCTRL_CMD_EEERWR_gc);

    /* Write byte to EEPROM */
    *(uint8_t *)(EEPROM_START + eeprom_adr) = data;

    /* Clear the current command */
    ccp_write_spm((void *)&NVMCTRL.CTRLA, NVMCTRL_CMD_NONE_gc);

    return NVM_OK;
}
```

2.4 I/O Ports and Pinouts

2.4.1 Common Functionalities

2.4.1.1 GPIO Basic Functionality

The megaAVR and AVR Dx families are not pin-to-pin compatible. Thus, the GPIO basic functionality is similar and is configurable using three registers for both families:

megaAVR®	AVR® Dx	Description
DDRx	PORTx.DIR	Data direction - controls the data direction (output driver)
PORTx	PORTx.OUT	Data out - controls the output driver level for each PORTx pin
PINx	PORTx.IN	Data in - shows the state of the PORTx pin

The following code snippets show how to configure the PORTA pin 7 as output-driven high for each family.

Example 2-3. megaAVR® - Port A, Pin 7 Configured as Output and Driven High

```
{
DDRA = 0x80;
PORTA = 0x80;
}
```

Example 2-4. AVR® Dx - Port A, Pin 7 Configured as Output and Driven High

```
{
PORTA.DIR = 0x80;
PORTA.OUT = 0x80;
}
```

2.4.1.2 Pin Override

The AVR microcontrollers are using the I/O PORT pins for basic operations (I/O access) and peripherals. The megaAVR microcontroller uses Alternate Port Function registers to switch between the two functionalities. It is the user's responsibility to properly configure the Alternate Port Function registers when the I/O pins are used by a peripheral.

For the AVR Dx devices, the peripheral configuration overwrites the basic functionality of the GPIO pin when the peripheral is enabled and uses the I/O pin. There are some peripherals (i.e., Timers, CCL, SPI) that do not mandatory require an I/O pin when they are used; the features of those peripherals are available internally as inputs to other peripherals (i.e., Event System, Interrupt, CCL) and/or for software access. In this case, an additional output enable option is provided into the peripheral configuration registers.

2.4.1.3 Pull-Up Option

For the megaAVR microcontrollers, the pull-ups are enabled by default at power-up. The software can enable or disable all ports simultaneously using the Pull-Up Disable (PUD) bit in the Special Function I/O (SFIOR) register.

Unlike megaAVR microcontrollers, the AVR Dx families provide a pull-up option for each pin individually. The pull-up is disabled at Reset and can be enabled by software using the Pull-Up Enable bit in the PORTx.PINnCTRL register.

2.4.2 AVR® Dx Additional Features

2.4.2.1 Direct Pin Configuration

The GPIO basic functionality is controlled using the three registers that reside in the extended I/O Register space. This space does not allow bit manipulation instructions, and the configuration update for one pin is done using the Read-Modify-Write instructions. The hardware Read-Modify-Write functionality ensures a safe and correct change of the drive values and/or input and sense configuration, but it is translated into three assembler instructions. The following code snippet shows the PA7 pin configuration as output:

Example 2-5. C Code Example:

```
    PORTA.DIR |= 0x80;
```

Example 2-6. Assembler Code:

```
    ;Load PORTA.DIR address into Z-pointer
    LDI R30, 0x00;
    LDI R31, 0x04;
    LDI R24, Z      ;Read content of PORTA.DIR register
    ORI R24, 0x80   ;Logic or with 0x80
    ST  Z, R24      ;Store result into PORTA.DIR register
```

To speed up the access and to offer more flexibility, the new AVR Dx microcontrollers offer a new subset of registers that allow the update of the GPIO configuration for multiple pins in one instruction:

Register (Note)	Description
PORTx.DIRSET	Writing a '1' to any bit in this bit field will set the corresponding bit in PORTx.DIR, which will configure the corresponding pin as an output pin and enable the output driver
PORTx.DIRCLR	Writing a '1' to any bit in this bit field will clear the corresponding bit in PORTx.DIR, which will configure the corresponding pin as an input-only pin and disable the output driver

...........continued

Register (Note)	Description
PORTx.DIRTGL	Writing a '1' to any bit in this bit field will toggle the corresponding bit in PORTx.DIR
PORTx.OUTSET	Writing a '1' to any bit in this bit field will set the corresponding bit in PORTx.OUT, which will configure the corresponding pin to be driven high
PORTx.OUTCLR	Writing a '1' to any bit in this bit field will clear the corresponding bit in PORTx.OUT, which will configure the corresponding pin to be driven low
PORTx.OUTTGL	Writing a '1' to any bit in this bit field will toggle the corresponding bit in PORTx.OUT

Note: Writing '0' to any bit position in the registers has no effect.

Using the direct pin configuration feature, the code for the configuration of the PA7 pin as output becomes:

Example 2-7. C Code:

```
PORTA.DIRSET = 0x80;
```

Example 2-8. Assembler Code:

```
;Load PORTA.DIRSET address into Z-pointer
LDI R30, 0x01;
LDI R31, 0x04;
LDI R24, 0x80      ;Load mask
ST  Z, R24         ;Write to the PORTA.DIRSET register
```

2.4.2.2 Virtual Ports

The often-used PORT registers are also mapped into bit-accessible I/O memory space. The access to the Virtual PORT registers has the same outcome as the access to the regular registers, but it allows for memory-specific instructions, such as bit manipulation instructions. These instructions cannot be used in the extended I/O Register space where the regular PORT registers reside.

Regular Port Registers	Virtual Port Registers	Description
PORTx.DIR	VPORTx.DIR	Data direction - controls the data direction (output driver)
PORTx.OUT	VPORTx.OUT	Data Out - controls the output driver level for each PORTx pin
PORTx.IN	VPORTx.IN	Data In - shows the state of the PORTx pin
PORTx.INTFLAGS	VPORTx.INTFLAGS	Pin Interrupt flag - is set when the change or state of the PORTx pin matches the pin's Input/Sense Configuration (ISC) in PORTx.PINnCTRL

2.4.2.3 Multipin Configuration

The multipin configuration function is used to configure the multiple PORT pins in one operation. The wanted pin configuration is first written to the PORTx.PINCONFIG register, followed by a register write, with the selected pins to modify. This allows changing the configuration (PORTx.PINnCTRL) for up to eight pins in one write. The following code snippet shows the pin configuration for PORTA:

```
 /* Pin configuration - inverted I/O enabled, Pull-up enabled, interrupt sensing on rising
edge */
    PORTA.PIN7CTRL = PORT_INVEN_bm | PORT_PULLUPEN_bm | PORT_ISC1_bm;
    PORTA.PIN5CTRL = PORT_INVEN_bm | PORT_PULLUPEN_bm | PORT_ISC1_bm;
    PORTA.PIN2CTRL = PORT_INVEN_bm | PORT_PULLUPEN_bm | PORT_ISC1_bm;
    PORTA.PIN0CTRL = PORT_INVEN_bm | PORT_PULLUPEN_bm | PORT_ISC1_bm;
```

The previous code can be replaced, using the multipin configuration feature, with:

```
/* Pin configuration - inverted I/O enabled, Pull-up enabled, interrupt sensing on rising
edge */
    PORTA.PINCONFIG  = PORT_INVEN_bm | PORT_PULLUPEN_bm | PORT_ISC1_bm;
/* Update PINxCTRL for pins PA7, PA5, PA2 and PA0 */
    PORTA.PINCTRLUPD = PIN7_bm | PIN5_bm | PIN2_bm | PIN0_bm;
```

2.4.2.4 PORTMUX

The AVR Dx devices provide multiple options for peripheral inputs/outputs. Those options are user-selectable using the PORTMUX registers and offer flexibility during design implementation (the user can choose the best option for PCB routing). For more details about the available inputs/outputs, refer to the specific AVR Dx data sheet.

2.5 Timers

The timers available on the AVR Dx families are improved versions of the timers present on ATmega128. Additionally, the AVR Dx devices present an increased number of timers and multiple options for their output, which simplifies the schematic design. The following type of timers are present on the AVR Dx devices:

- Timer/Counter type A (TCA)
- Timer/Counter type B (TCB)
- Timer/Counter type D (TCD)

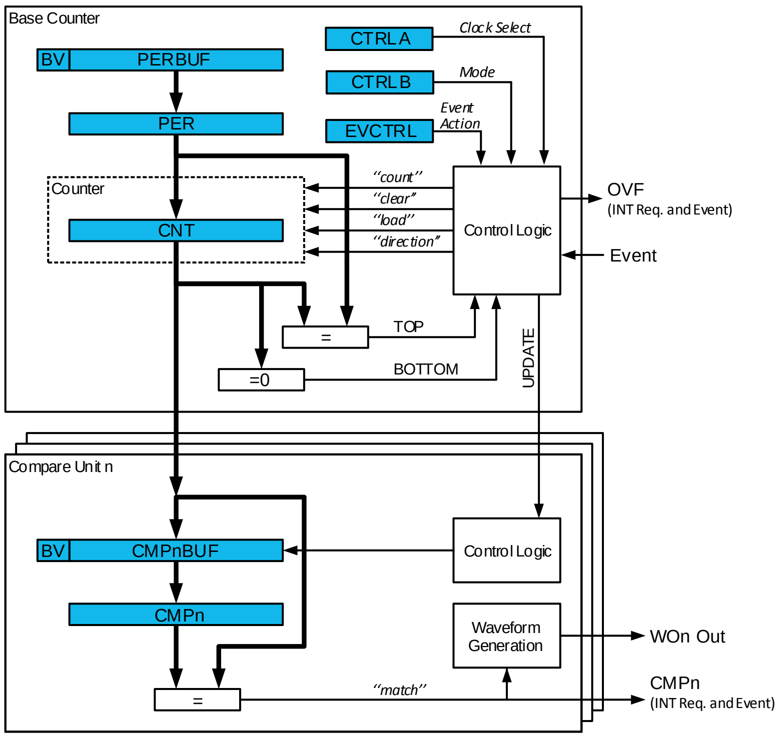
The timers are not code-compatible (a different set of registers), but all functionalities present on ATmega128 are replicable using AVR Dx timers. Thus, the following sections will describe how to obtain different functionalities using AVR Dx timers.

2.5.1 TCA - Timer/Counter Type A

The Timer Counter type A (TCA) is an improved version of Timer 0 of megaAVR devices. The TCA consists of a base counter and a set of compare channels. The compare channels can be used together with the base counter to do a compare match control, frequency generation, and pulse-width waveform modulation.

Unlike Timer 0 of the megaAVR devices, which is an 8-bit timer, the TCA is a 16-bit timer/counter with three compare channels. It 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.

Figure 2-4. AVR® Dx - TCA Block Diagram



Additionally, the TCA has a Split mode feature that splits the 16-bit timer into two separate 8-bit timers, each having three compare channels for PWM generation. The Split mode will only work with single-slope down-count. The event controlled operation is not supported in Split mode.

The following subsection will show a few possible use cases for the TCA.

2.5.1.1 Normal Mode (Counter Mode)

In this mode, the TCA is used as an up/down counter having peripheral clock or events as a clock source. Depending on the mode of operation used, the internal counter is cleared, incremented, or decremented at each timer clock or event. The Counter (TCAn.CNT) Register value can be read by the CPU anytime. The write access to TCAn.CNT has a 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 TCAn.CTRLE.

In Normal mode, the TCA Overflow output is not present on the pin, but the TCA can be used to generate an interrupt when the number of counting events reached the TCAn.CMPn value. The following code shows the configuration of TCA to generate periodic interrupts (interrupt code not included):

Example 2-9. AVR® Dx - TCA Initialization, Periodic Interrupt Enabled

```
void TCA0_init(void)
{
    /* enable overflow interrupt */
    TCA0.SINGLE.INTCTRL = TCA_SINGLE_OVF_bm;

    /* set Normal mode */
    TCA0.SINGLE.CTRLB = TCA_SINGLE_WGMODE_NORMAL_gc;

    /* disable event counting */
    TCA0.SINGLE.EVCTRL &= ~(TCA_SINGLE_CNTEI_bm);

    /* set the period */
    TCA0.SINGLE.PER = PERIOD_EXAMPLE_VALUE;

    TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV256_gc         /* set clock 
source (sys_clk/256) */
                      | TCA_SINGLE_ENABLE_bm;               /* start timer */
}
```

2.5.1.2 Waveform Generation Modes

The compare channels can be used for waveform generation on the corresponding port pins. Each Compare Channel n continuously compares the counter value (TCAn.CNT) with the Compare n (TCAn.CMPn) register. If TCAn.CNT equals TCAn.CMPn, the Comparator n signals a match. This signal is then used for waveform generation, function by the WGMODE bit field selection in the TCAn.CTRLB register.

Note: One single waveform mode is available for all the compare channels since the WGMODE bit field is common to all channels.

Example 2-10. AVR® Dx - TCA Initialization, Single-Scope PWM Mode

```
void TCA0_init(void)
{
    /* set waveform output on PORT A */
    PORTMUX.TCAROUTEA = PORTMUX_TCA0_PORTA_gc;

    TCA0.SINGLE.CTRLB = TCA_SINGLE_CMP0EN_bm     /* enable compare channel 0 */
            | TCA_SINGLE_WGMODE_SINGLESLOPE_gc;  /* set single slope PWM mode 
*/

    /* disable event counting */
    TCA0.SINGLE.EVCTRL &= ~(TCA_SINGLE_CNTEI_bm);

    /* set PWM frequency and duty cycle (50%) */
    TCA0.SINGLE.PERBUF = PERIOD_EXAMPLE_VALUE;
    TCA0.SINGLE.CMP0BUF = DUTY_CYCLE_EXAMPLE_VALUE;

    TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV4_gc /* set clock source 
(sys_clk/4) */
            | TCA_SINGLE_ENABLE_bm;              /* start timer */
}
```

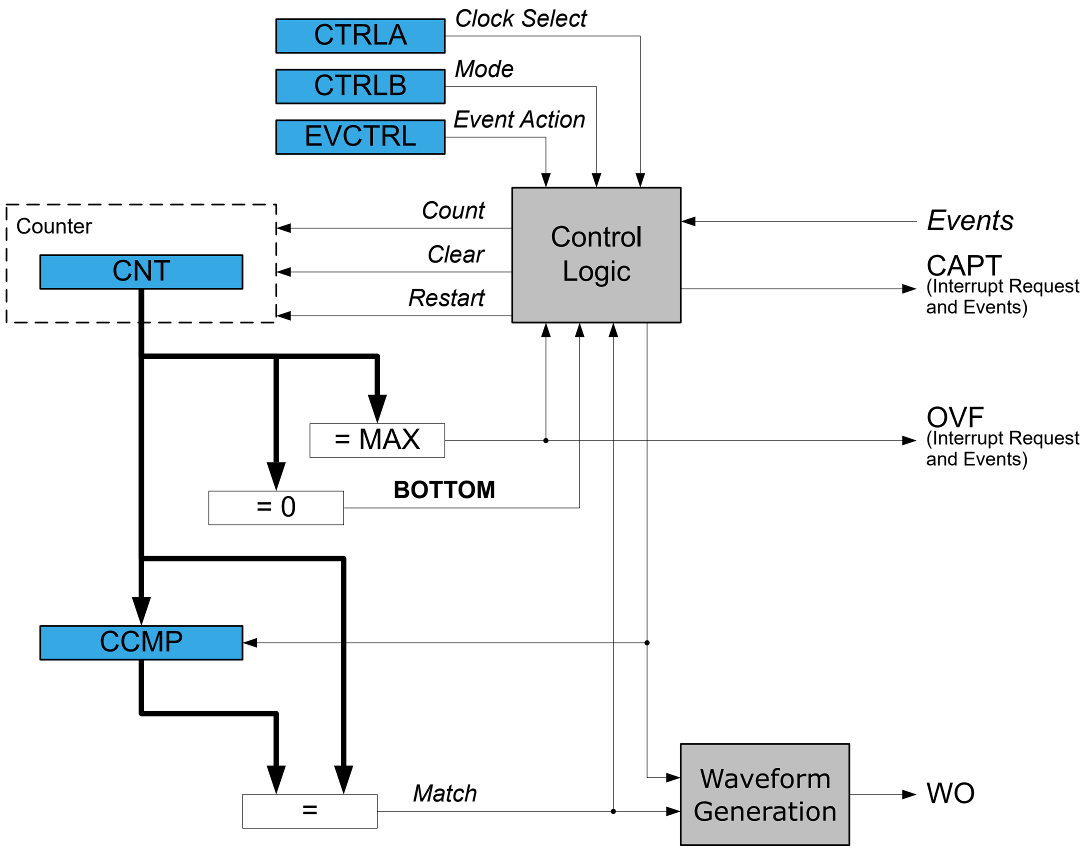
Note: For more details and code examples, refer to *TB3217 - Getting Started with TCA*.

2.5.2 TCB - Timer/Counter Type B

The Timer/Counter type B (TCB) of the AVR Dx devices can be used to replace all the functionalities of Timer 1 from the megaAVR devices.

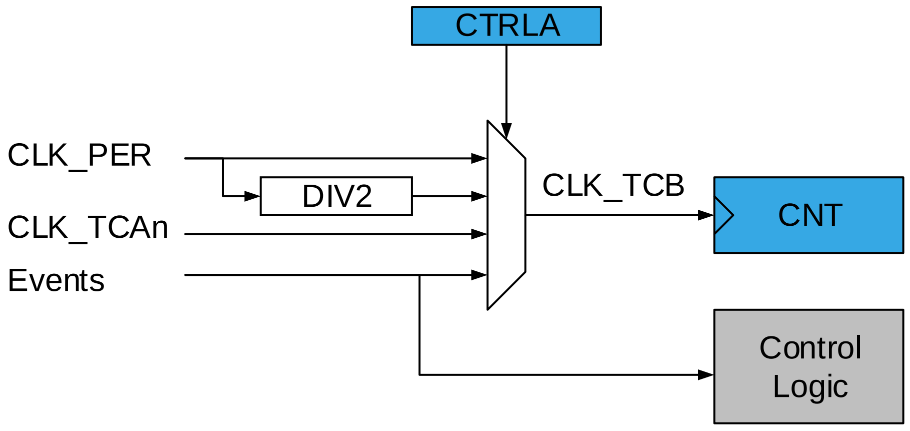
The capabilities of the TCB include frequency and waveform generation and input capture on an event with time and frequency measurement of the digital signals. The TCB consists of a base counter and control logic that can be set in one of eight different modes, each mode providing unique functionality.

Figure 2-5. AVR® Dx - TCB Block Diagram



The TCB can be clocked from the Peripheral Clock (CLK_PER), from a 16-bit Timer/Counter type A (CLK_TCAn) or the Event System (EVSYS), selectable through the Clock Select (CLKSEL) bit field in the Control A (TCBn.CTRLA) register:

Figure 2-6. AVR® Dx - TCB Clock Logic



If the waveform output (WO) is required to be available on a pin, the Compare/Capture Output Enable (CCMPEN) from the Control B (TCBn.CTRLB) register must be written to ‘1’. The location of WO is also selectable using the PORTMUX.TCBROUTEA register, but the direction of the WO pin needs to be configured as an output (the TCB overwrites the functionality of the output pin but does not overwrite the pin direction).

The following code snippets show the TCB initialization examples for a few use cases.

2.5.2.1 TCB in Input Capture on Event Mode

In this mode, the counter will count from BOTTOM to MAX continuously. When an event is detected, the CNT is transferred to the Capture/Compare (TCBn.CCMP) register, and a CAPT interrupt and event is generated. This mode can be used, but not limited, to measure the interval between successive events.

Since the counter is working continuously, the Overflow (OVF) bit from the TCBn.INTFLAGS register is used to signalize that. It is the user’s responsibility to take into account when the overflow appears between successive events.

Example 2-11. AVR® Dx - TCB initialization for Input Capture on Event Mode

```
void TCB2_Init(void)
{
    TCB2.CTRLB = 0 << TCB_ASYNC_bp       /* Asynchronous Enable: disabled */
                | 0 << TCB_CCMPEN_bp     /* Pin Output Enable: disabled */
                | 0 << TCB_CCMPINIT_bp   /* Pin Initial State: disabled */
                | TCB_CNTMODE_CAPT_gc;   /* Input Capture Event */

    TCB2.EVCTRL = 1 << TCB_CAPTEI_bp    /* Event Input Enable: enabled */
                 | 0 << TCB_EDGE_bp     /* Event Edge: positive */
                 | 0 << TCB_FILTER_bp; /* Input Capture Noise Cancellation
Filter: disabled */

    TCB2.INTCTRL = 1 << TCB_CAPT_bp     /* Capture or Timeout: enabled */
                  | 0 << TCB_OVF_bp;   /* OverFlow Interrupt: disabled */

    TCB2.CTRLA = TCB_CLKSEL_DIV1_gc     /* CLK_PER */
                | 1 << TCB_ENABLE_bp    /* Enable: enabled */
                | 0 << TCB_RUNSTDBY_bp /* Run Standby: disabled */
                | 0 << TCB_SYNCUPD_bp  /* Synchronize Update: disabled */
                | 0 << TCB_CASCADE_bp; /* Cascade Two Timer/Counters:
disabled */
}
```

2.5.2.2 TCB in Input Capture for Frequency Measurement Mode

In Input Capture for Frequency Measurement mode, the TCB captures the counter value and restarts on either a positive or negative edge of the event input signal (the active edge is selectable using the Event Edge bit from the TCBn.EVCTRL register). Using this mode, the period of the input signal is available to software on each signal period, helping the software to monitor the signal’s period/frequency.

Example 2-12. AVR® Dx - TCB initialization for Input Capture Frequency Measurement Mode

```
void TCB2_Init(void)
{
    TCB2.CTRLB = 0 << TCB_ASYNC_bp       /* Asynchronous Enable: disabled */
                | 0 << TCB_CCMPEN_bp     /* Pin Output Enable: disabled */
                | 0 << TCB_CCMPINIT_bp   /* Pin Initial State: disabled */
                | TCB_CNTMODE_FRQ_gc;    /* Input Capture Frequency
measurement */

    TCB2.EVCTRL = 1 << TCB_CAPTEI_bp    /* Event Input Enable: enabled */
                 | 0 << TCB_EDGE_bp     /* Event Edge: disabled */
                 | 0 << TCB_FILTER_bp; /* Input Capture Noise Cancellation
Filter: disabled */

    TCB2.INTCTRL = 1 << TCB_CAPT_bp     /* Capture or Timeout: enabled */
                  | 0 << TCB_OVF_bp;   /* OverFlow Interrupt: disabled */

    TCB2.CTRLA = TCB_CLKSEL_DIV1_gc     /* CLK_PER */
                | 1 << TCB_ENABLE_bp    /* Enable: enabled */
                | 0 << TCB_RUNSTDBY_bp /* Run Standby: disabled */
```

```
                      | 0 << TCB_SYNCUPD_bp  /* Synchronize Update: disabled */
                      | 0 << TCB_CASCADE_bp; /* Cascade Two Timer/Counters:
disabled */
}
```

2.5.2.3 TCB in Single-Shot Mode

The Single-Shot mode can be used to generate a pulse with a duration defined by the Compare (TCBn.CCMP) register every time a rising or falling edge is observed on a connected event channel.

Example 2-13. AVR® Dx - TCB Initialization in Single-Shot Mode

```
void TCB2_Init(void)
{
    TCB2.CCMP = 0x400;          /* Pulse width:  0x400*/

    TCB2.CTRLB = 0 << TCB_ASYNC_bp          /* Asynchronous Enable: disabled */
               | 0 << TCB_CCMPEN_bp         /* Pin Output Enable: disabled */
               | 0 << TCB_CCMPINIT_bp       /* Pin Initial State: disabled */
               | TCB_CNTMODE_SINGLE_gc;     /* Single Shot*/

    TCB2.EVCTRL = 1 << TCB_CAPTEI_bp       /* Event Input Enable: enabled */
                | 0 << TCB_EDGE_bp         /* Event Edge: disabled */
                | 0 << TCB_FILTER_bp;      /* Input Capture Noise Cancellation
Filter: disabled */

    TCB2.INTCTRL = 1 << TCB_CAPT_bp        /* Capture or Timeout: enabled */
                 | 0 << TCB_OVF_bp;        /* OverFlow Interrupt: disabled */

    TCB2.CTRLA = TCB_CLKSEL_DIV1_gc        /* CLK_PER */
               | 1 << TCB_ENABLE_bp        /* Enable: enabled */
               | 0 << TCB_RUNSTDBY_bp      /* Run Standby: disabled */
               | 0 << TCB_SYNCUPD_bp       /* Synchronize Update: disabled */
               | 0 << TCB_CASCADE_bp;      /* Cascade Two Timer/Counters:
disabled */

}
```

For all previous uses cases, the TCB must be used in correlation with the Event System. The following sample of code shows the configuration of the Event System (the pin PA6 is used as an input event for TCB2):

Example 2-14. AVR® Dx - Event System Initialization for TCB

```
int8_t EVENT_SYSTEM_0_init()
{
    EVSYS.CHANNEL0 = EVSYS_CHANNEL0_PORTA_PIN6_gc; /* Port A Pin 6 */
    EVSYS.USERTCB2CAPT = EVSYS_USER_CHANNEL0_gc;   /* Connect TCB2 to event
channel 0 */
```

Additionally to the use cases presented, the TCB can be used for periodic interrupts, time-out checks, PWM generation, or as a 32-bit timer in conjunction with another TCB peripheral. Refer to the AVR Dx data sheet for the complete list of operation modes.

Note: For more details and code examples, refer to *TB3214 - Getting Started with TCB*.

2.5.3 TCD - Timer/Counter Type D

The Timer/Counter type D (TCD) is a high-performance waveform generator that consists of an asynchronous counter, a prescaler, and compare, capture and control logic. The TCD counter can run on a clock which is asynchronous to the peripheral clock. It contains compare logic that generates 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.

Similar to other AVR Dx timers, the TCD can be used to generate PWM waveforms. Even if the waveform generation modes are similar to TCA or TCB (one ramp, two ramp, four ramp, dual slope), the TCD compare logic allows conditional waveform generation on external events like Fault handling, input blanking, overload protection, and fast emergency stop by hardware. Those features, together with dead-time support, make the TCD an ideal choice for applications that require hardware support for half-bridge and full-bridge signals.

The TCD waveform output signals are controlled by the Compare Enable (CMPxEN) bits in the TCDn.FAULTCTRL register and are available internally or on the pins. Similar to other AVR Dx peripherals, the TCD overwrites the functionality of the pin but does not configure the pin as an output. If the waveform outputs are needed on the pins, the software must configure those pins as outputs.

The following code shows TCD initialization for the generation of complementary driving signals, with dead-time:

Example 2-15. AVR® Dx - TCD Initialization in Half-Bridge Mode

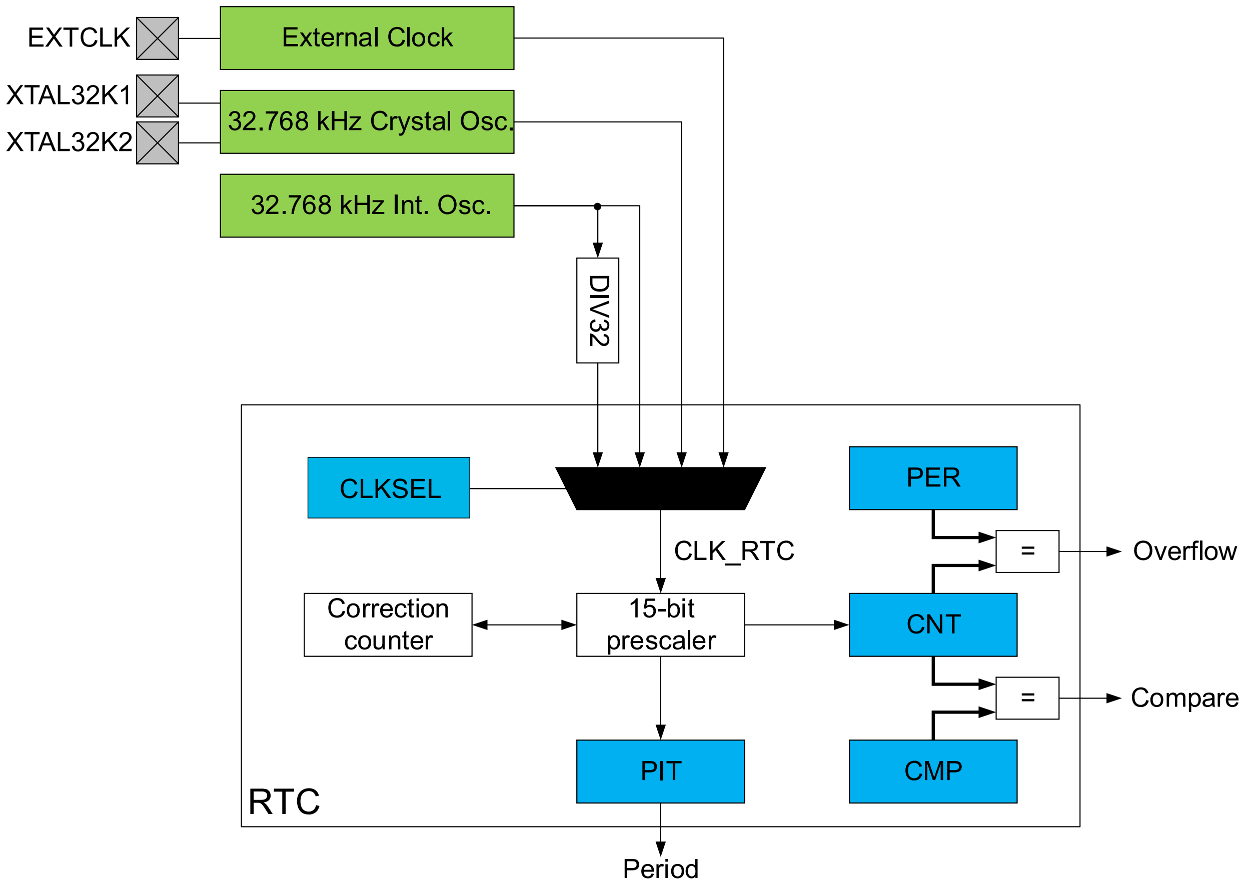
```
void TCD0_init(void)
{
    /* set the waveform mode */
    TCD0.CTRLB = TCD_WGMODE_DS_gc;

    /* set the signal period */
    TCD0.CMPBCLR = SIGNAL_PERIOD_EXAMPLE_VALUE;

    /* the signals are alternatively active and a small symmetric dead-time is 
needed */
    TCD0.CMPBSET = SIGNAL_DUTY_CYCLE_EXAMPLE_VALUE + 1;
    TCD0.CMPASET = SIGNAL_DUTY_CYCLE_EXAMPLE_VALUE - 1;

    /* ensure ENRDY bit is set */
    while(!(TCD0.STATUS & TCD_ENRDY_bm))
    {
    ;
    }

    TCD0.CTRLA = TCD_CLKSEL_20MHZ_gc     /* choose the timer's clock */
               | TCD_CNTPRES_DIV1_gc     /* choose the prescaler */
               | TCD_ENABLE_bm;          /* enable the timer */
}
```

Note: For more details and code examples, refer to *TB3212 - Getting Started with TCD*.

2.6 RTC - Real-Time Counter

The Real-Time Counter (RTC) is a peripheral that typically runs continuously, including in Low-Power sleep modes, to keep track of time. It offers two timing functions: The Real-Time Counter (RTC) and a Periodic Interrupt Timer (PIT). It can wake up the device from sleep modes and/or interrupt the device at regular intervals.

Figure 2-7. AVR® Dx - RTC Block Diagram



The RTC peripheral works in Idle and Standby sleep modes. A wide range of resolutions and time-out periods can be configured for it. With a 32.768 kHz clock source, the maximum resolution is 30.5 µs, and time-out periods can be up to two seconds. With a resolution of 1s, the maximum time-out period is more than 18 hours (65536 seconds).

The following code snippets show the initialization of RTC to generate interrupts with a 1-second periodicity (the interrupt code is not included):

Example 2-16. AVR® Dx - RTC Initialization for 1s Periodic Interrupt

```
void RTC_0_init()
{
    while (RTC.STATUS > 0) {    /* Wait for all register to be synchronized */
    }
    RTC.PER = 0x3FF;            /* Period: 0x3FF - 1024 RTC clock cycles */
    RTC.CLKSEL = RTC_CLKSEL_OSC1K_gc;  /* 32kHz divided by 32 */
    RTC.INTCTRL = 0 << RTC_CMP_bp    /* Compare Match Interrupt enable: 
disabled */
                | 1 << RTC_OVF_bp;    /* Overflow Interrupt enable: enabled */
}
```

In addition to the RTC functionality, the RTC of the AVR Dx devices includes a Periodic Interrupt Timer (PIT) that can run even in Power-Down mode. If enabled, it can generate an interrupt request on every 2^n clock period, where *n* can have any value from 2 to 15, selectable via the RTC.PITCTRLA register. This mode allows periodic interrupts/wake-up from sleep when the device is in Power-Down mode.

Note: For details about using the RTC and code examples, refer to *TB3213 - Getting Started with RTC*.

2.7 SPI

2.7.1 Common Functionalities

The functionalities of the AVR Dx SPI peripheral in Host/Client mode are similar to ATmega128 when Normal mode is used, but the register names and bit order can be different. The following table shows the mapping of the megaAVR SPI configuration bits into the AVR Dx SPI register structure:

megaAVR®	AVR® Dx	Description
SPCR.DORD	SPI.CTRLA.DORD	SPI Data order
SPCR.MSTR	SPI.CTRLA.MASTER	Host/Client Select
SPCR.CPOL	SPI.CTRLB.MODE[1:0]	SPI Mode Select
SPCR.CPHA		
SPCR.SPR1	SPI.CTRLA.PRESC[1:0]	SPI Clock Prescaler (divider)
SPCR.SPR0		
SPSR.SPI2X	SPI.CTRLA.CLK2X	Clock Double
SPCR.SPE	SPI.CTRLA.ENABLE	SPI Enable
SPCR.SPIE	SPI.INTCTRL.IE	SPI Interrupt Enable
SPSR.SPIF	SPI.INTFLAGS.IF	SPI Interrupt Flag
SPSR.WCOL	SPI.INTFLAGS.WRCOL	Write Collision Flag
SPDR	SPI.DATA	SPI Data Register

Another difference between families is the I/O pin configuration for use with the SPI peripheral. The megaAVR needs a proper configuration of used pins, while AVR Dx devices overwrite the I/O pin functionality when the pin is used by the SPI peripheral. Refer to the 2.4.1.2 Pin Override section for details.

The following code snippets show initialization of the SPI peripheral in Host mode:

Example 2-17. megaAVR® - SPI Initialization in Host Mode

```
void SPI_MasterInit(void)
{
    /* Set MOSI and SCK output, all others input */
    DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);
    /* Enable SPI, Master, set clock rate fck/16 */
    SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}

void SPI_MasterTransmit(char cData)
{
    /* Start transmission */
    SPDR = cData;
    /* Wait for transmission complete */
    while(!(SPSR & (1<<SPIF)))
    ;
}
```

Example 2-18. AVR® Dx - SPI Initialization in Host Mode

```
void SPI0_init(void)
{
    PORTA.DIR &= ~PIN4_bm;           /* Set MOSI pin direction to input */
    PORTA.DIR |= PIN5_bm;            /* Set MISO pin direction to output */
    PORTA.DIR &= ~PIN6_bm;           /* Set SCK pin direction to input */
    PORTA.DIR &= ~PIN7_bm;           /* Set SS pin direction to input */
    SPI0.CTRLA = SPI_DORD_bm          /* LSB is transmitted first */
               | SPI_ENABLE_bm        /* Enable module */
               & (~SPI_MASTER_bm);    /* SPI module in Client mode */
}

uint8_t SPI0_exchangeData(uint8_t data)
{
    SPI0.DATA = data;
    while (!(SPI0.INTFLAGS & SPI_IF_bm))  /* waits until data is exchanged*/
    {
    ;
    }
    return SPI0.DATA;
}
```

2.7.2 AVR® Dx Additional Features

The SPI peripheral of the AVR Dx families has support for data buffering, thus influencing the data handling. The Buffer mode is enabled by writing the BUFEN bit in the SPIn.CTRLB register to '1'. There are additional bits in the SPI.INTCTRL and SPI.INTFLAGS registers to support Buffer mode. Refer to the specific AVR Dx data sheet for details.

Note: For more details about using the SPI and code examples, refer to *TB3215 - Getting Started with SPI*.

2.8 USART

2.8.1 Common Functionality

The basic operation of USART is similar for both families, but the register and bit names are different. The AVR Dx devices have an improved register structure that allows a more efficient access to the configuration/status bits.

The following section details the mapping of ATmega128 registers into AVR Dx register structure (only ATmega registers and AVR Dx correspondences are figured):

megaAVR®	AVR® Dx	Description
RXBn (UDRn read)	USARTn.RXDATAL	Receiver Data Register Low Byte
TXBn (UDRn write)	USARTn.TXDATAL	Transmit Data Register Low Byte
UCSRnB.RXB8n	USARTn.RXDATAH.DATA[8]	Receive Data Bit 8
UCSRnB.TXB8n	USARTn.TXDATAH.DATA[8]	Transmit Data Bit 8
UCSRnB.UCSZn[2]	USARTn.CTRLC.CHSIZE[2:0][(1)]	USART Character Size
UCSRnC.UCSZn[1:0]		
UCSRnB.RXENn	USARTn.CTRLC.RXEN	Receiver Enable
UCSRnB.TXENn	USARTn.CTRLC.TXEN	Transmitter Enable
UCSRnB.RXCIENn	USARTn.CTRLA.RXCIE	RX Complete Interrupt Enable
UCSRnB.TXCIENn	USARTn.CTRLA.TXCIE	TX Complete Interrupt Enable
UCSRnB.UDRIENn	USARTn.CTRLA.DREIE	USART Data Register Empty Interrupt Enable
UCSRnA.RXCn	USART.STATUS.RXCIF	USART Receive Complete Flag
UCSRnA.TXCn	USART.STATUS.TXCIF	USART Transmit Complete Flag
UCSRnA.UDREn	USART.STATUS.DREIF	USART Data Register Empty Flag
UCSRnA.FEn	USARTn.RXDATAH.FERR	Framing Error
UCSRnA.DORn	USARTn.RXDATAH.BUFOVF	Data OverRun

...........continued

megaAVR®	AVR® Dx	Description
UCSRnA.UPEn	USARTn.RXDATAH.PERR	Parity Error
UCSRnA.U2Xn	USARTn.CTRLB.RXMODE[1:0] = CLK2X	Double the USART operation Speed
UCSRnA.MPCMn	USARTn.CTRLB.MPCM	Multi-Processor Communication mode
UCSRnC.UMSELn	USARTn.CTRLC.CMODE[1:0][(2)]	USART Mode Select
UCSRnC.UPMn[1:0]	USARTn.CTRLC.PMODE[1:0]	Parity Mode
UCSRnC.UBSn	USARTn.CTRLC.SBMODE	Stop Bit Mode
UCSRnC.UCPOLn	USARTn.CTRLC.UCPHA[(3)]	Clock Polarity (SYNC MODE)
UBRRn[11:0]	USARTn.BAUD[15:0][(4)]	Baud Rate registers

Notes:
1. The settings are different for 9-bit operations.
2. ASYNCH and SYNCH modes only.
3. Only in Synchronous mode.
4. The baud rate formula is different. Refer to the device data sheet for details.

The following code snippets shows the initialization of the USART peripheral for both families:

Example 2-19. megaAVR® - USART Initialization (9600N1)

```
#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

void USART_Init(void){
   /* Set baud rate */
   UBRRL = BAUD_PRESCALE;             /* Load lower 8 bits into the low byte
of the UBRR register */
   UBRRH = (BAUD_PRESCALE >> 8);      /* Load upper 8 bits into the high byte
of the UBRR register */

   UCSRB = (1<<RXEN)|(1<<TXEN);       /* Enable receiver and transmitter */
   UCSRC = (1<<USBS)|(3<<UCSZ0);      /* Set frame format: 8data, 1 stop bit */
}
```

Example 2-20. AVR® Dx - USART Initialization (9600N1)

```
#define USART1_BAUD_RATE(BAUD_RATE)      ((float)(64 * 4000000 / (16 *
(float)BAUD_RATE)) + 0.5)

void USART1_init(void)
{
    USART1.BAUD = (uint16_t)(USART1_BAUD_RATE(9600));   /* set the baud rate*/

    USART1.CTRLC = USART_CHSIZE0_bm
                 | USART_CHSIZE1_bm; /* set the data format to 8-bit*/

    USART1.CTRLB |= USART_RXEN_bm | USART_TXEN_bm;      /* enable receiver and
transmitter*/
}
```

2.8.2 AVR® Dx Additional Features

The AVR Dx devices contain an improved version of the USART peripheral with additional features like:

- Half-Duplex Operation (One-Wire and RS-485 Modes)
- Support Host SPI Mode

- Auto-Baud Feature (Generic and LIN)
- RCOM Module for IrDA® Compliant Communication

Note: For details and additional examples, refer to *TB3216 - Getting Started with USART*.

2.9 TWI - Two-Wire Serial Interface

The TWI peripheral allows the systems designer to interconnect up to 128 individually addressable devices using only two bidirectional bus lines: one for clock (SCL) and one for data (SDA). The functionality of the AVR Dx TWI peripheral in Host/Client mode is similar to ATmega128 for basic operations, but the peripheral architecture, register names, and bit order are different. Because of those differences, the software procedures that interact with the TWI peripheral must be updated and tested during the integration phase.

The following code snippets show the usage of the TWI peripheral in Host mode:

Example 2-21. megaAVR® - TWI in Host Mode

```
/* Function to initialize master */
void TWI_init_master(void)
{
    /* SCL freq= F_CPU/(16+2(TWBR).4^TWPS) */
    TWBR = 0x01; // Bit rate
    TWSR = (0<<TWPS1)|(0<<TWPS0);   /* Setting prescalar bits */
}

void TWI_start(void)
{
    /* Clear TWI interrupt flag, Put start condition on SDA, Enable TWI */
    TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);
    while (!(TWCR & (1<<TWINT)));  /* Wait till start condition is transmitted
*/
    while ((TWSR & 0xF8)!= 0x08);  /* Check for the acknowledgment */
}

void TWI_write_address(unsigned char data)
{
    TWDR = data;                  /* Address and write instruction */
    TWCR=(1<<TWINT)|(1<<TWEN);    /* Clear TWI interrupt flag,Enable TWI */
    while (!(TWCR & (1<<TWINT))); /* Wait till complete TWDR byte transmitted
*/
    while ((TWSR & 0xF8)!= 0x18); /* Check for the acknowledgment */
}

void TWI_write_data(unsigned char data)
{
    TWDR = data;                  /* Put data in TWDR */
    TWCR=(1<<TWINT)|(1<<TWEN);    /* Clear TWI interrupt flag,Enable TWI */
    while (!(TWCR & (1<<TWINT))); /* Wait till complete TWDR byte transmitted
*/
    while ((TWSR & 0xF8)!= 0x28); /* Check for the acknowledgment */
}

unsigned char TWI_read_data(void)
{
unsigned char recv_data;
    TWCR = (1<<TWINT)|(1<<TWEN);    /* Clear TWI interrupt flag,Enable TWI */
    while (!(TWCR & (1<<TWINT)));   /* Wait till complete TWDR byte
transmitted */
    while ((TWSR & 0xF8) != 0x58);  /* Check for the acknowledgment */
    recv_data = TWDR;
    return recv_data;
}

void TWI_stop(void)
{
    /* Clear TWI interrupt flag, Put stop condition on SDA, Enable TWI */
    TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);
    while(!(TWCR & (1<<TWSTO)));  /* Wait till stop condition is transmitted */
}
```

Example 2-22. **AVR® Dx - TWI in Host Mode**

```
void TWI0_init()
{
    TWI0.MBAUD = (uint8_t)TWI0_BAUD(100000, 0); /* set MBAUD register */
    TWI0.MCTRLA = 1 << TWI_ENABLE_bp   /* Enable TWI Master: enabled */
                | 0 << TWI_QCEN_bp     /* Quick Command Enable: disabled */
                | 0 << TWI_RIEN_bp     /* Read Interrupt Enable: disabled */
                | 0 << TWI_SMEN_bp     /* Smart Mode Enable: disabled */
                | TWI_TIMEOUT_DISABLED_gc /* Bus Timeout Disabled */
                | 0 << TWI_WIEN_bp;    /* Write Interrupt Enable: disabled */
}

void TWI0_start(void)
{
    /* The start condition is generated by hardware, kept for compatibility */
}

static uint8_t TWI0_WaitW(void)
{
    uint8_t state = 0;
    do
    {
        if(TWI0.MSTATUS & (TWI_WIF_bm | TWI_RIF_bm))
        {
            if(!(TWI0.MSTATUS & TWI_RXACK_bm))
            {
                /* slave responded with ack - TWI goes to M1 state */
                state = I2C_ACKED;
            }
            else
            {
                /* address sent but no ack received - TWI goes to M3 state */
                state = I2C_NACKED;
            }
        }
        else if(TWI0.MSTATUS & (TWI_BUSERR_bm | TWI_ARBLOST_bm))
        {
            /* get here only in case of bus error or arbitration lost - M4
state */
            state = I2C_ERROR;
        }
    } while(!state);
    return state;
}

uint8_t TWI0_write_address(uint8_t address)
{
    uint8_t state = 0;
    TWI0.MADDR = address;      /* Transmitting the slave address */
    state = TWI0_WaitW();      /* wait for error code */
    return state;
}

uint8_t TWI0_write_data(uint8_t data)
{
   uint8_t state = 0;
   TWI0.DATA = data;          /* Transmitting the data */
   state = TWI0_WaitW();      /* wait for error code */
   return state;
}

static uint8_t TWI0_WaitR(void)
{
    uint8_t state = I2C_INIT;
    do
    {
        if(TWI0.MSTATUS & (TWI_WIF_bm | TWI_RIF_bm))
        {
            state = I2C_READY;
        }
        else if(TWI0.MSTATUS & (TWI_BUSERR_bm | TWI_ARBLOST_bm))
        {
            /* get here only in case of bus error or arbitration lost - M4
state */
```

```
                state = I2C_ERROR;
            }
     } while(!state);
     return state;
}

uint8_t TWI0_read_data(uint8_t* data, bool last_byte)
{
    uint8_t state;
    state = TWI0_WaitR();
    if(state == I2C_READY)
        {
             *data = TWI0.DATA;
             if (last_byte) TWI0.MCTRLB = TWI_ACKACT_bm | TWI_MCMD_STOP_gc;
             else TWI0.MCTRLB = TWI_MCMD_RECVTRANS_gc;
         }
     return state;
}

void TWI0_stop(void)
{
     TWI0.MCTRLB |= TWI_MCMD_STOP_gc;
}
```

2.9.1 AVR® Dx Additional Features

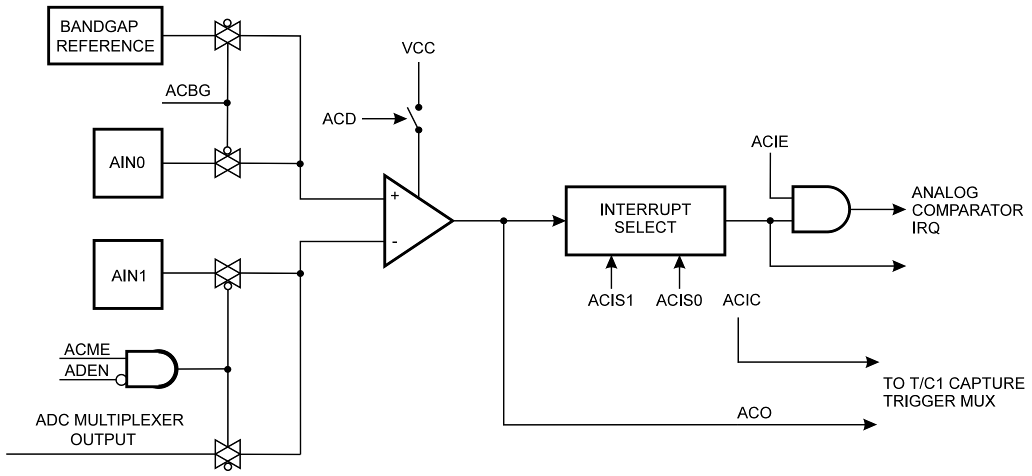
The TWI peripheral of the AVR Dx families is equipped with an improved version of the TWI peripheral, providing support for a new set of features like:

- SMBus
- Multi-Host
- Smart Mode
- Dual Mode
- 10-bit Address
- Quick Command Mode

2.10 AC - Analog Comparator

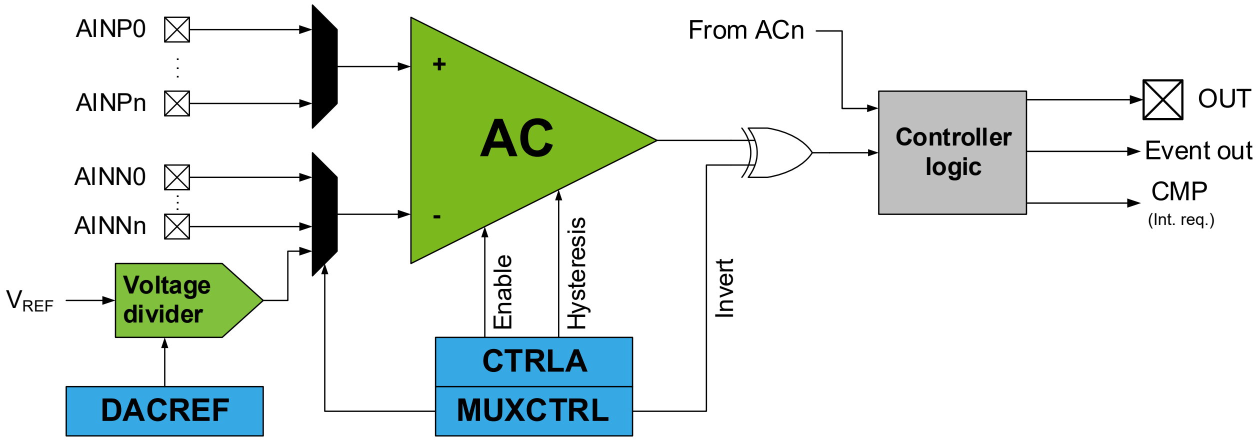
The Analog Comparator (AC) of the megaAVR devices has different register sets and limited features compared to the AVR Dx devices. The positive input of the megaAVR Analog Comparator is selectable between the external pin (AIN0) and the internal reference (fixed, 1.23V typical) using the ACBG bit from the ACSR register, while the negative one can be selected between the dedicated external pin (AIN1) and the ADC input pins using the Analog Comparator Multiplexer Enable (ACME) bit in the SFIOR register. When the ACME bit is set and the ADC is switched off (the ADEN bit in the ADCSRA register is zero), the MUX[2..0] bit field in the ADMUX register selects the input pin to replace the negative input to the Analog Comparator.

Figure 2-8. Analog Comparator Block Diagram for megaAVR® Devices



The AVR Dx devices provide a larger selection for positive and negative inputs, using the ACn.MUXCTRL register. Each Analog Comparator is equipped with a dedicated multiplexer, thus allowing unrestricted use of the available analog inputs. A dedicated internal voltage reference (V_{REF}) is provided for the Analog Comparator. Additionally, each comparator is equipped with an on-chip 8-bit DAC for a precise selection of the reference voltage (DACREF):

Figure 2-9. Analog Comparator Block Diagram for AVR® Dx Devices



Despite those differences, the functionality for the megaAVR AC can be obtained by a proper configuration of the AC peripheral of AVR Dx. The following code example shows the initialization of the AC peripheral for both families (input compare with a fixed voltage reference):

Example 2-23. megaAVR® - AC Basic Initialization

```
#define AINpin PA3
void ACInit(void)
{
    DDRA   &=~(1<<AINpin);                        /* set pin as input */
    PORTA  &=~(1<<AINpin);                        /* no pull-up */
    SFIOR  |= (1<<ACME);                          /* enable multiplexer */
    ADCSRA &= ~(1<<ADEN);                         /* make sure ADC is OFF */
    ADMUX  |=(0<<MUX2)|(1<<MUX1)|(1<<MUX0); /* select ADC3 as negative AIN */
    ACSR  |= (0<<ACD)|                         /* Comparator ON */
             (1<<ACBG)|                        /* Connect 1.23V reference to AIN0 */
             (0<<ACIE)|                        /* Comparator Interrupt disabled */
             (0<<ACIC)|                        /* input capture disabled */
             (0<<ACIS1)|
             (0<<ACIS0);
}
```

Example 2-24. AVR® Dx - AC Basic Initialization

```
/* set DACREF to 1.23 Volts for VREF = 1.5 Volts */
#define DACREF_VALUE (1.23 * 256 / 1.5)

void AC0_init(void)
{
        /* Positive Input - Disable digital input buffer */
         PORTD.PIN2CTRL = PORT_ISC_INPUT_DISABLE_gc;

         /* Negative input uses internal reference - voltage reference must be
enabled */
         VREF.CTRLA = VREF_AC0REFSEL_1V5_gc;    /* Voltage reference at 1.5V */
         VREF.CTRLB = VREF_AC0REFEN_bm;         /* AC0 DACREF reference enable:
enabled */

         AC0.DACREF = DACREF_VALUE;             /* Set DAC voltage reference */
         /*Select proper inputs for comparator*/
         AC0.MUXCTRLA = AC_MUXPOS_PIN0_gc       /* Positive Input - Analog
Positive Pin 0 */
                        | AC_MUXNEG_DACREF_gc;  /* Negative Input - DAC Voltage
Reference */

         AC0.CTRLA = AC_ENABLE_bm               /* Enable Analog Comparator */
                   | AC_OUTEN_bm;               /* Output Buffer Enable: enabled
*/
 }
```

2.10.1 AVR® Dx Additional Features

The AVR Dx devices provide up to three Analog Comparators, each two of them configurable in Window mode. In Window mode, a voltage window can be defined, and the selected comparator indicates whether an input signal is within this range or not.

The devices are equipped with a voltage reference for Analog Comparators, selectable via the Voltage Reference (VREF) peripheral (the same value will apply to all ACs in the device). Each comparator is equipped with an additional 8-bit DAC that allows a fine adjusting of the voltage reference levels.

To improve noise immunity, the hysteresis feature is available using the HYSMODE[1:0] bit field in the ACn.CTRLA register. This feature has selectable levels and helps prevent constant toggling of the output when the noise-afflicted input signals are close to each other.

Note: For additional information and code examples, refer to *TB3211 - Getting Started with AC*.

2.11 ADC - Analog-to-Digital Converter

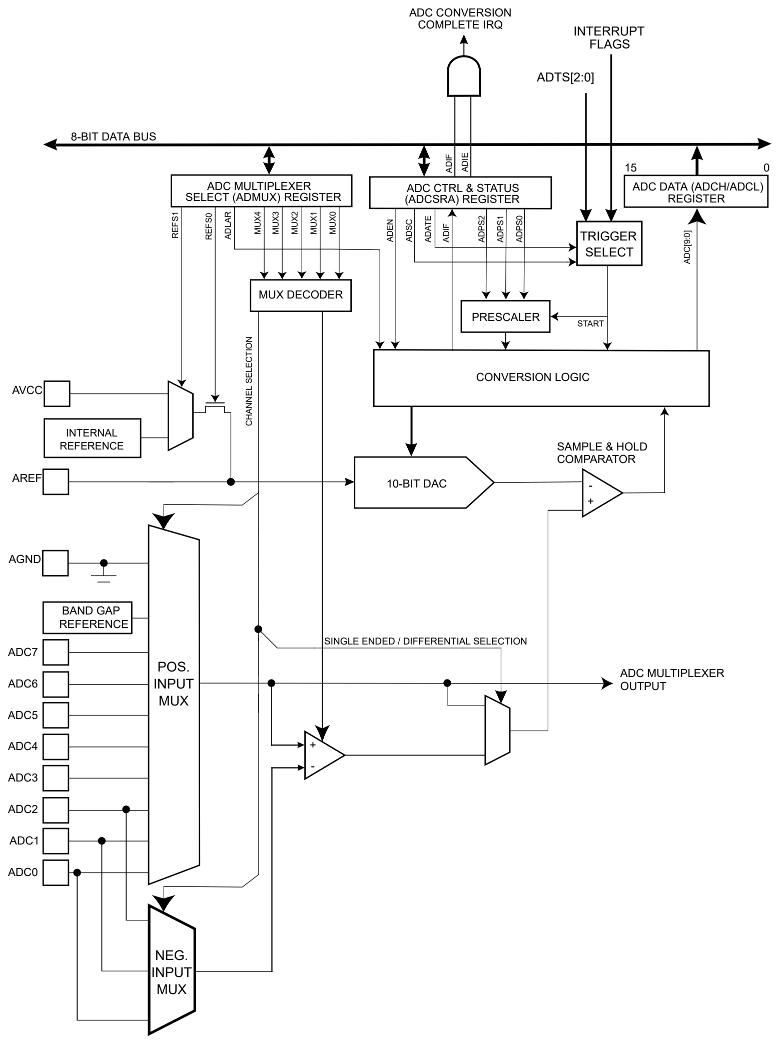
2.11.1 megaAVR® Devices

The Analog-to-Digital Converter (ADC) of the megaAVR devices has different register sets and limited features compared to the AVR Dx devices. Due to those differences, the migrated software must be fully tested and validated to ensure similar functionality on both families.

For the megaAVR devices, the ADC has a 10-bit resolution and a sample rate of up to 15 ksps. The ADC is connected to an 8-channel Analog Multiplexer, which allows eight single-ended voltage inputs constructed using the pins of Port A. The device also supports 16 differential voltage input combinations, two of them being equipped with a programmable gain stage. The input configuration is selected using the Analog Channel and Gain Selection bit field in the ADC Multiplexer Selection (ADMUX) register.

The ADC has the option for internal or external references, selectable using the Reference Selection (REFS) bit field in the ADMUX register. The internal voltage reference can be selected between on-chip 2.56V fixed voltage or the AVCC pin and is available on the AREF pin for decoupling when used. Therefore, the internal voltage reference cannot be used if an external reference voltage is being applied to the AREF pin.

Figure 2-10. megaAVR® - ADC Block Diagram

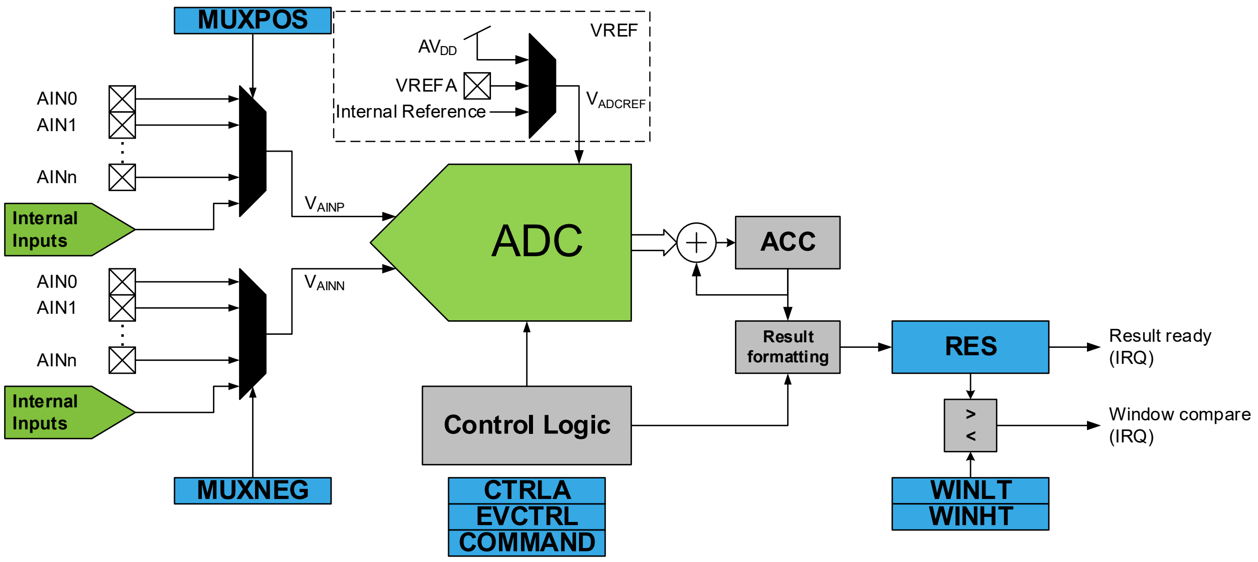


2.11.2 AVR® Dx Devices

The ADC of AVR Dx devices has a 12-bit resolution and a sample rate of up to 130 ksps. Its multiple internal and external inputs can be used in Single or Differential mode, selectable using the Conversion Mode (CONVM) bit in the Control A (ADCn.CTRLA) register. Unlike the megaAVR devices, in the Differential mode, any combination of available positive and negative inputs is allowed and can be configured using the Multiplexer Selection (MUXPOS and MUXNEG) registers.

The ADC voltage reference is also improved, a dedicated voltage reference (V_{REF}) being available. This reference is selectable using the ADC0 Reference (ADC0REF) register of the Voltage Reference (VREF) peripheral.

Figure 2-11. AVR® Dx - ADC Block Diagram



The following code shows the ADC initialization in Single-Ended mode, internal voltage reference:

Example 2-25. megaAVR® - ADC Initialization in Single-Ended Mode

```
void ADC_Init(void)
{
    /* Set ADC prescalar to 128 */
    ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);

    /* Set ADC reference to 2.56V internal reference */
    ADMUX |= (1 << REFS0);
    /* Left adjust ADC result to allow easy 8-bit reading */
    ADMUX |= (1 << ADLAR);

    // No MUX values needed to be changed to use ADC0 input

    ADCSRA |= (1 << ADFR);  /* Set ADC to Free-Running mode */
    ADCSRA |= (1 << ADEN);  /* Enable ADC */
    ADCSRA |= (1 << ADSC);  /* Start A2D conversions */
}
```

Example 2-26. AVR® Dx - ADC Initialization in Single-Ended Mode

```
void ADC0_init(void)
{
     /* Disable digital input buffer */
     PORTD.PIN0CTRL &= ~PORT_ISC_gm;
     PORTD.PIN0CTRL |= PORT_ISC_INPUT_DISABLE_gc;

     /* Disable pull-up resistor */
     PORTD.PIN0CTRL &= ~PORT_PULLUPEN_bm;

     ADC0.CTRLC = ADC_PRESC_DIV4_gc      /* CLK_PER divided by 4 */
```

```
                 | ADC_REFSEL_INTREF_gc; /* Internal reference */

    ADC0.CTRLA = ADC_ENABLE_bm          /* ADC Enable: enabled */
                 | ADC_RESSEL_12BIT_gc; /* 12-bit mode */

    ADC0.MUXPOS = ADC_MUXPOS_AIN0_gc;   /* Select ADC channel 0*/
    ADC0.CTRLA |= ADC_FREERUN_bm;       /* Enable Free-Running mode */
    ADC0.COMMAND = ADC_STCONV_bm;       /* Start conversion */
}
```

2.11.3 AVR® Dx - Additional Features

In addition to the megaAVR features, the ADC of AVR Dx devices provides other features like:

- Accumulation of up to 128 Samples per Conversion
- On-Chip Temperature Sensor Channel
- Programmable Input Sampling Duration
- Configurable Threshold and Window Comparator
- Event-Triggered Conversion

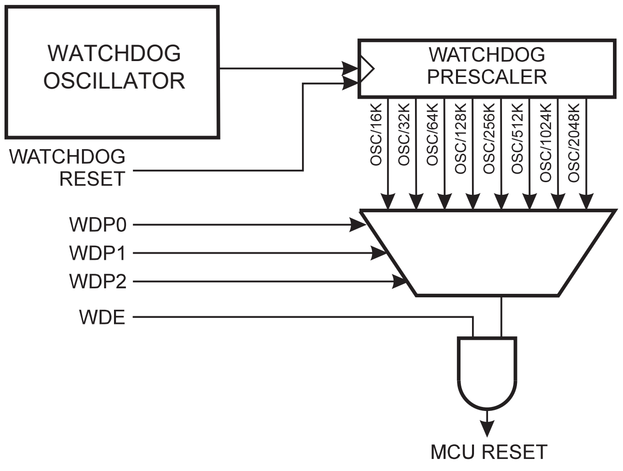
Note: For more details and code examples, refer to *TB3209 - Getting Started with ADC*.

2.12 WDT - Watchdog Timer

Both families are equipped with a Watchdog Timer (WDT) peripheral, but the feature and register sets are different. Thus, the software procedures that configure the megaAVR WDT operation must be fully replaced to support AVR Dx devices.

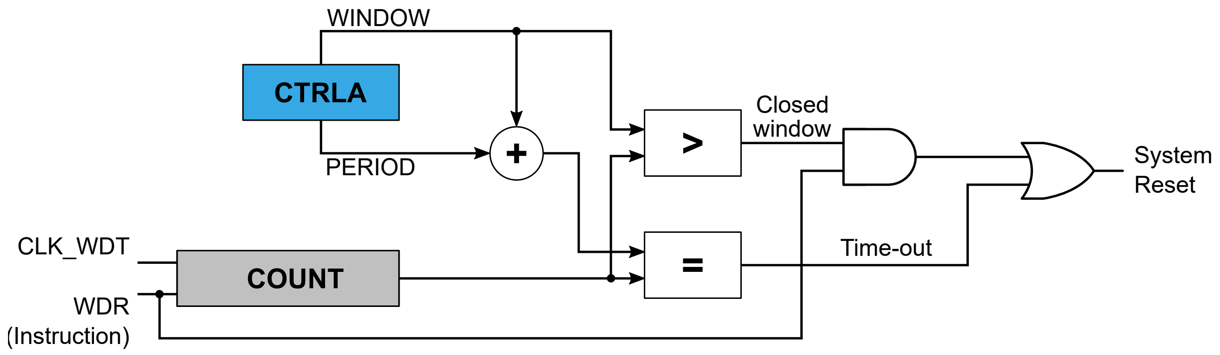
The watchdog of the megaAVR devices is clocked from an on-chip oscillator that runs at 1 MHz. The Watchdog Reset interval is selectable from 16 ms to 2.2 seconds using the Watchdog Timer Prescaler (WDP[2:0]) bits in the Watchdog Timer Control (WDTCTRL) register.

Figure 2-12. megaAVR® - WDT Block Diagram



For the AVR Dx devices, the Watchdog module is improved and has an additional set of features. It operates asynchronously from the peripheral clock using an independent oscillator. It is clocked from an on-chip ultra-low power oscillator for improved power consumption and provides 11 selectable time-out intervals from 16 ms up to 8 seconds.

Figure 2-13. AVR® Dx - WDT Block Diagram



The WDT for both families has a write protection mechanism ensuring the WDT settings cannot be changed by accident. The following code examples show the Watchdog disable procedures:

Example 2-27. megaAVR® - Watchdog Disable Code Example

```
void WDT_off(void)
{
    /* Reset WDT*/
    _WDR();
    /* Write logical one to WDTOE and WDE */
    WDTCR |= (1<<WDTOE) | (1<<WDE);
    /* Turn off WDT */
    WDTCR = 0x00;
}
```

Example 2-28. AVR® Dx - Watchdog Disable Code Example

```
void WDT_0_off(void)
{
    /* Reset WDT */
    asm("WDR");
    /* Disable WDT */
    ccp_write_io((void *)&(WDT.CTRLA),
                 WDT_PERIOD_OFF_gc     /* Off */
                 | WDT_WINDOW_OFF_gc   /* Off */);
}
```

2.12.1 AVR® Dx Additional Features

The AVR Dx devices have the option to configure the watchdog functionality using Watchdog Config (WDTCFG) fuses. This option ensures the WDT can be configured to run immediately after Reset without software support but not eliminate the need for the Watchdog Reset (`WDR`) instruction. The software must periodically run the `WDR` instruction to avoid a system reset triggered by the WDT.

Note: The configuration of the WDT cannot be modified by software if it is enabled from fuses (the LOCK bit in the WDT.STATUS register is set automatically). Refer to the AVR Dx devices data sheet for details.

In addition to the normal operation, the AVR Dx watchdog has a Window mode. The Window mode defines a time slot or window inside the time-out 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 operation, the Window mode can catch situations where a code error causes constant `WDR` execution.

3. AVR® Dx - Additional Peripherals

3.1 Overview

The AVR Dx families are equipped with additional peripherals that replace the need for external hardware components in some applications. Those components allow a more compact design, lowering the cost and power consumption.

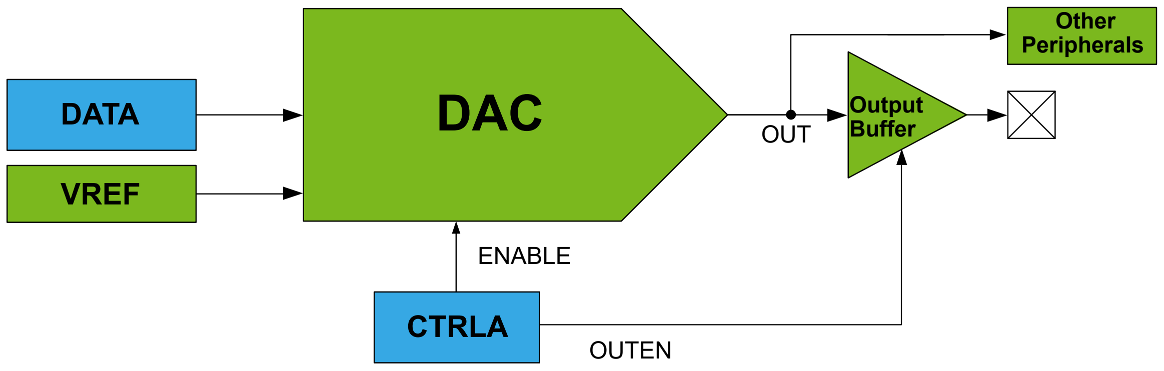
Table 3-1. AVR® Dx - Additional peripherals

Peripherals/Features	AVR® DA Family	AVR® DB Family
Digital-to-Analog Converter (DAC)	1 (10 bits)	1 (10 bits)
Configurable Custom Logic (CCL)	6 LUTs	6 LUTs
Event System (EVSYS)	1 (10 channels)	1 (10 channels)
Cyclic Redundancy Check (CRCSCAN)	1	1
Zero-Cross Detector (ZCD)	3	3
Peripheral Touch Controller (PTC)	1	-
Multi-Voltage I/O (MVIO)	-	1
Analog Signal Conditioning (OPAMP)	-	3

3.2 DAC - Digital-to-Analog Converter

The AVR Dx devices are equipped with a 10-bit Digital-to-Analog Converter (DAC) that can run up to 140 ksps conversion rate. The output range is between GND and selected voltage reference and can be available on an external pin or to be used by other internal peripherals (like ADC).

Figure 3-1. AVR® Dx - DAC Block Diagram



The analog output of the DAC can be connected to a pin by writing a '1' to the Output Buffer Enable (OUTEN) bit in the Control A (DACn.CTRLA) register. The pin used by the DAC must have the input disabled from the Port peripheral. There is an output buffer between the DAC output and the pin, which ensures the analog value does not depend on the load of the pin. The output buffer can only source current, and it has very limited sinking capability.

The following code snippets show the initialization code for DAC to generate a fixed voltage on the DAC analog output pin:

Example 3-1. AVR® Dx - DAC Initialization to Generate Fixed Voltage

```
/* DAC Value */
#define DAC_EXAMPLE_VALUE         (0x258)
/* Mask needed to get the 2 LSb for DAC Data Register */
#define LSB_MASK                  (0x03)

void DAC0_init(void)
 {
    VREF.DAC0REF = VREF_REFSEL_2V048_gc /* Select the 2.048V Internal Voltage
Reference for DAC */
                 | VREF_ALWAYSON_bm;    /* Set the Voltage Reference in Always
On mode */
    /* Disable digital input buffer */
    PORTD.PIN6CTRL &= ~PORT_ISC_gm;
    PORTD.PIN6CTRL |= PORT_ISC_INPUT_DISABLE_gc;
    /* Disable pull-up resistor */
    PORTD.PIN6CTRL &= ~PORT_PULLUPEN_bm;
    DAC0.CTRLA = DAC_ENABLE_bm          /* Enable DAC */
               | DAC_OUTEN_bm           /* Enable output buffer */
               | DAC_RUNSTDBY_bm;       /* Enable Run in Standby mode */

    /* Store the two LSbs in DAC0.DATAL */
    DAC0.DATAL = (DAC_EXAMPLE_VALUE & LSB_MASK) << 6;
    /* Store the eight MSbs in DAC0.DATAH */
    DAC0.DATAH = DAC_EXAMPLE_VALUE >> 2;
 }
```

Note: For code examples and more details about using the DAC peripheral, refer to *TB3235 - Using 10-bit DAC for generating Analog Signals*.

3.3 CCL - Configurable Custom Logic

The Configurable Custom Logic (CCL) is a programmable logic peripheral that can be connected to the device pins, events, or 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 (CIPs) to handle the most time-critical parts of the application independent of the CPU.

Note: For details about using the CCL peripheral and code examples, refer to the Custom Logic on PIC® and AVR® Microcontrollers webpage.

3.4 EVSYS - Event System

The Event System (EVSYS) is a routing network enabling inter-peripheral communication without involving the CPU. 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. Events are latency-free and never lost, providing fast and predictable signaling, enabling a deterministic system ideal for real-time applications. It allows for autonomous peripheral control and interaction and synchronized timing of actions in several peripheral modules. Thus, it is a powerful tool for reducing the complexity, size, and execution time of the software.

Note: For details about using the EVSYS peripheral and code examples, refer to the Event System (EVSYS) webpage.

3.5 CRCSCAN - Cyclic Redundancy Check Memory Scan

The AVR Dx devices provide a Cyclic Redundancy Check mechanism, which is an important safety feature. By ensuring no code corruption has occurred, a potentially unintended behavior in the application that can cause a dangerous situation can be avoided.

The CRCSCAN peripheral scans the Nonvolatile Memory (NVM), making sure the code is not corrupted. It generates a checksum that is compared to a pre-calculated one. If the two checksums match, the Flash is OK, and the application code can start running.

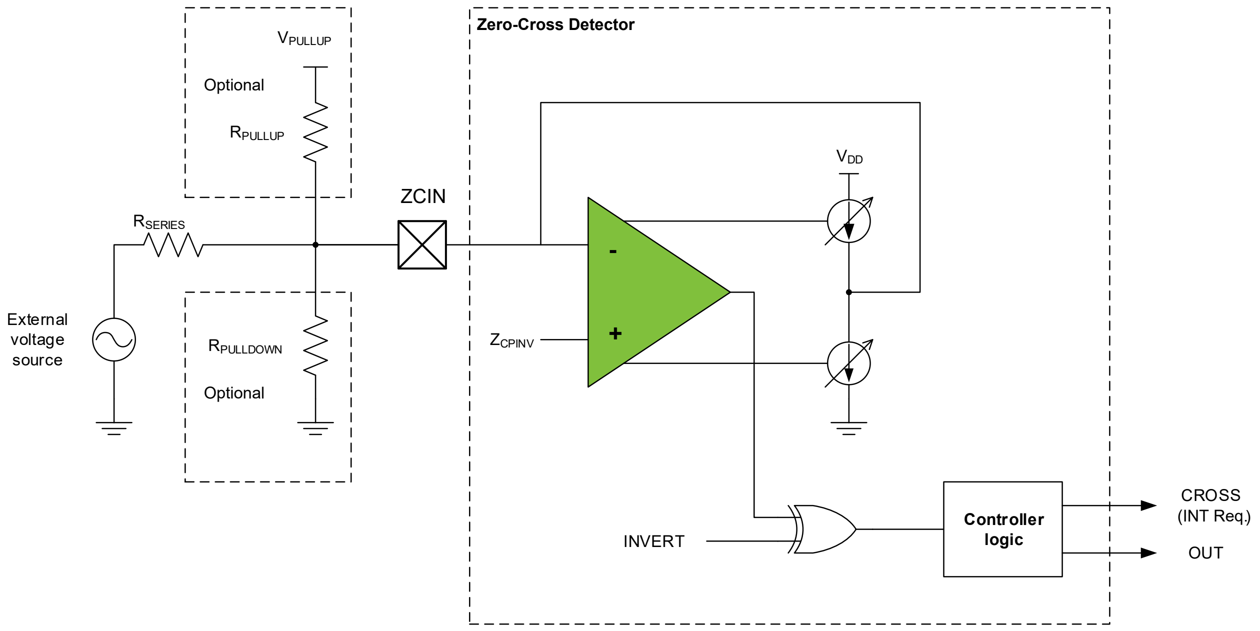
Note: For details about using the CRCSCAN peripheral and code examples, refer to the Cyclic Redundancy Check (CRC/SCAN) webpage.

3.6 ZCD - Zero-Cross Detector

The Zero-Cross Detector (ZCD) detects when an alternating voltage crosses through a threshold voltage near the ground potential. The ZCD can be used when monitoring an alternating waveform for, but not limited to, the following purposes:

- Period Measurement
- Accurate Long-Term Time Measurement
- Dimmer Phase-Delayed Drive
- Low-EMI Cycle Switching

Figure 3-2. AVR® Dx - ZCD Block Diagram



To detect zero-crossing, it requires only a current limiting resistor in series (R_{SERIES}), thus simplifying the design.

Note: For code examples and details about using the ZCD peripheral, refer to *TB3233 - Using ZCD to Implement Special Functions*.

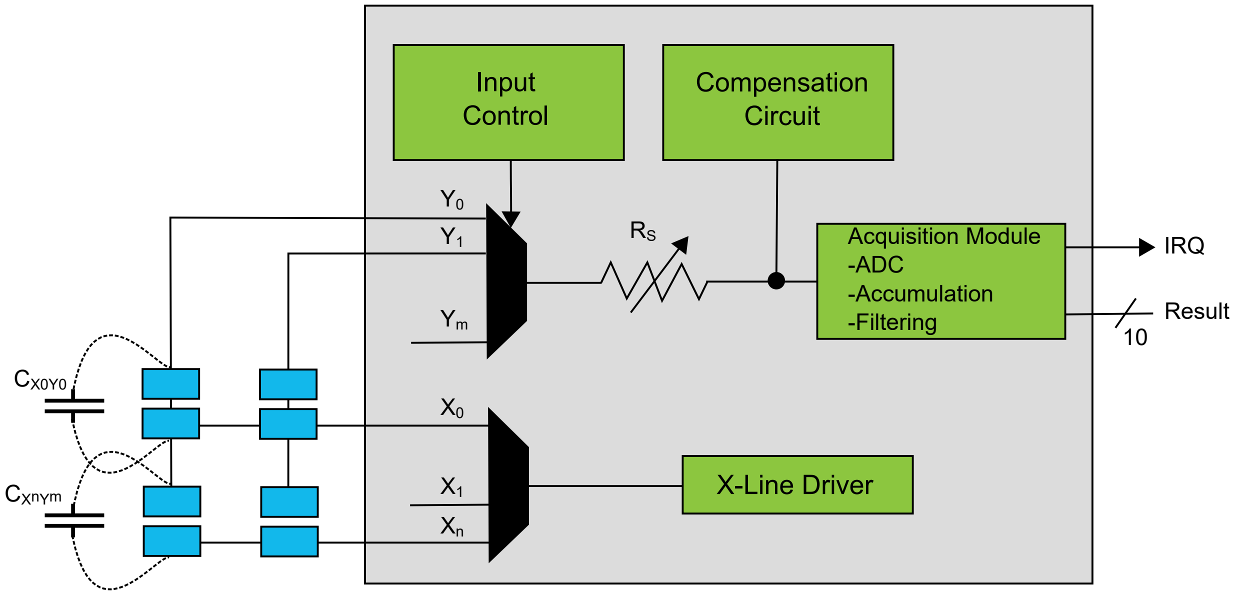
3.7 PTC - Peripheral Touch Controller

The Peripheral Touch Controller (PTC) is a dedicated peripheral to detect a touch on capacitive sensors. The external capacitive touch sensor is typically formed on a PCB or a transparent substrate with a transparent or translucent material such as indium tin oxide (ITO) or PEDOT. The PTC allows the design of robust touch solutions with low-power, high-sensitivity for a large variety of sensors (buttons, sliders, wheels or 2D surfaces) without using external components.

The sensor electrodes are connected directly to the analog front end of the PTC through the I/O pins in the device. The dedicated hardware and the QTouch® software library allow low CPU utilization and faster development of touch solutions.

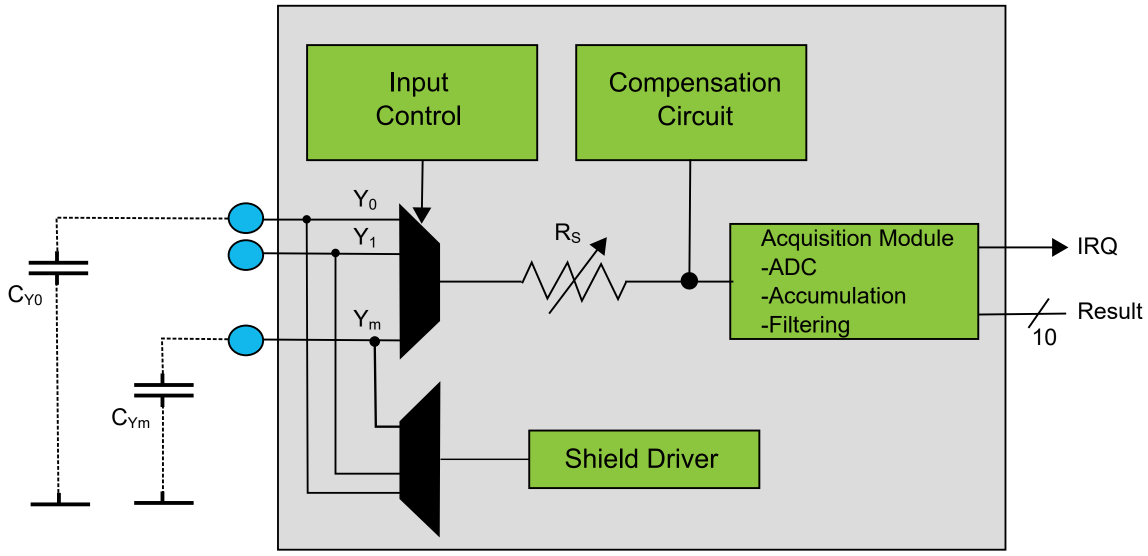
The PTC supports both mutual and self-capacitance sensors. In Mutual Capacitance mode, sensing is done using capacitive touch matrices in various X-Y configurations. The PTC requires one pin per X-line and one pin per Y-line:

Figure 3-3. AVR® Dx - PTC Block Diagram in Mutual Capacitance Mode



In Self-Capacitance mode, the PTC requires one pin (Y-line) for each touch sensor:

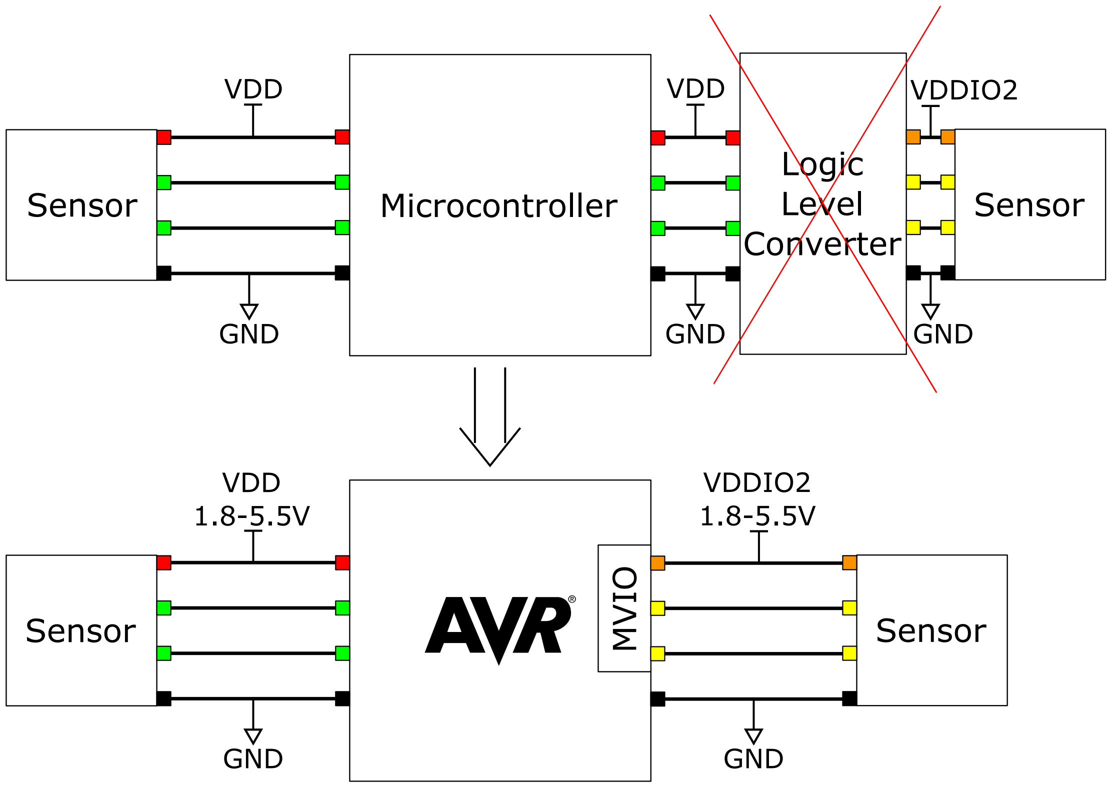
Figure 3-4. AVR® Dx - PTC Block Diagram in Self-Capacitance Mode



3.8 MVIO - Multi-Voltage I/O

The MVIO feature allows a subset of the I/O pins to be powered by a different I/O voltage domain than the rest of the I/O pins. This eliminates the need to have external level shifters for communication or control of external components running on a different voltage level. Eliminating external logic level shifters will, in turn, reduce the BOM and free up PCB space.

Figure 3-5. Multi-Voltage I/O Replacing External Logic Level Converters



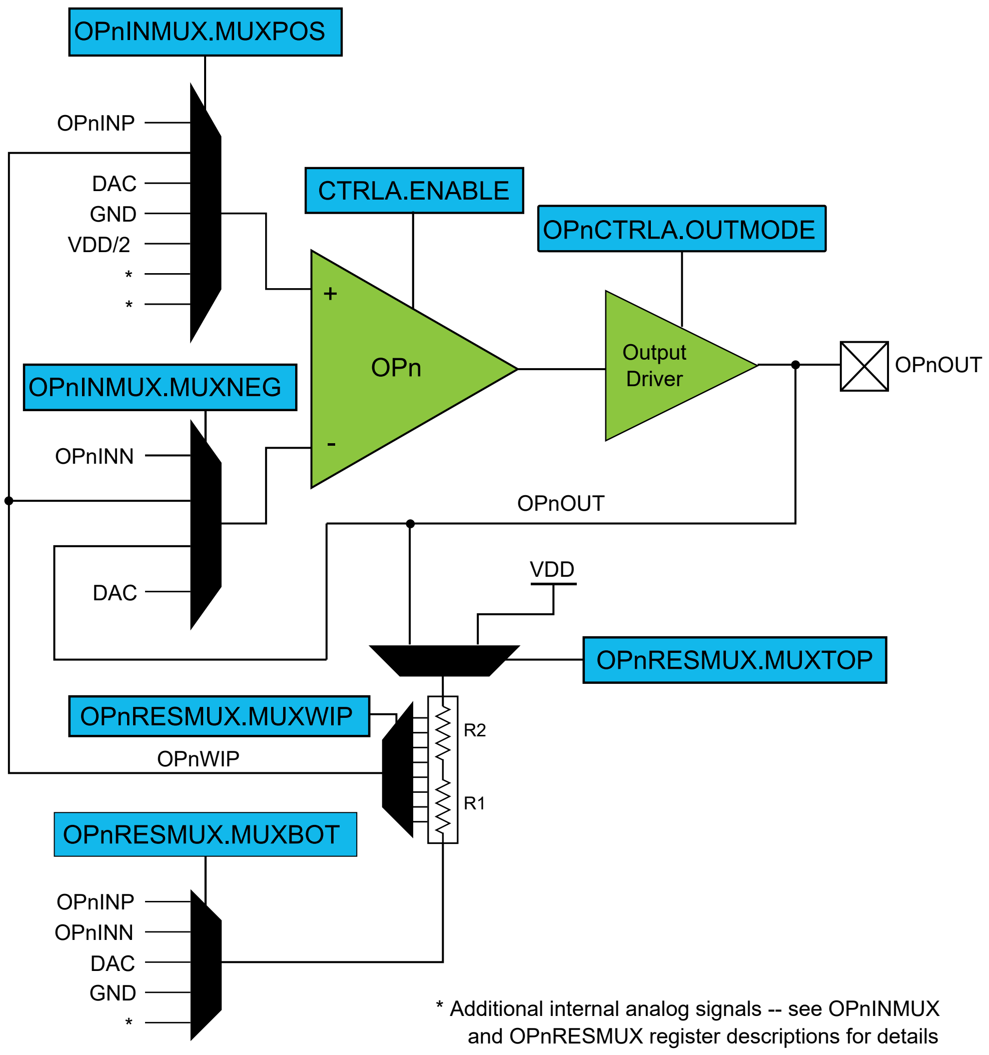
Note: For more details and code example, refer to *TB3287 - Getting Started with MVIO*.

3.9 OPAMP - Analog Signal Conditioning

The Analog Signal Conditioning (OPAMP) peripheral features up to three operational amplifiers (op amps). These op amps are implemented with a flexible connection scheme using analog multiplexers and resistor ladders. This allows a large number of analog signal conditioning configurations to be achieved, many of which require no external components.

A multiplexer at the non-inverting (+) input of each op amp allows the connection to either an external pin, a wiper position from a resistor ladder, a DAC output, ground, $V_{DD}/2$, or an output from another op amp. A second multiplexer at the inverting (-) input of each op amp allows connection to either an external pin, a wiper position from a resistor ladder, the output of the op amp, or DAC output. Three more multiplexers connected to each resistor ladder provide additional configuration flexibility. Two of these multiplexers select the top and bottom connections to the resistor ladder, and the third controls the wiper position.

Figure 3-6. Op Amp Block Diagram



Note: For code examples and details about using the OPAMP peripheral, refer to *TB3286 - Getting Started with Analog Signal Conditioning (OPAMP)*.

4. References

- ATmega128 Data Sheet
- AVR128DA64 Data Sheet
- AVR128DB64 Data Sheet

Additional documents can be found at www.microchip.com.

5. Revision History

Doc. Rev.	Date	Comments
A	11/2020	Initial document release

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **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 (FAQs), technical support requests, online discussion groups, Microchip design partner 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

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Legal Notice

Trademarks

ISBN: 978-1-5224-7079-3

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4485-5910
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-72884388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820