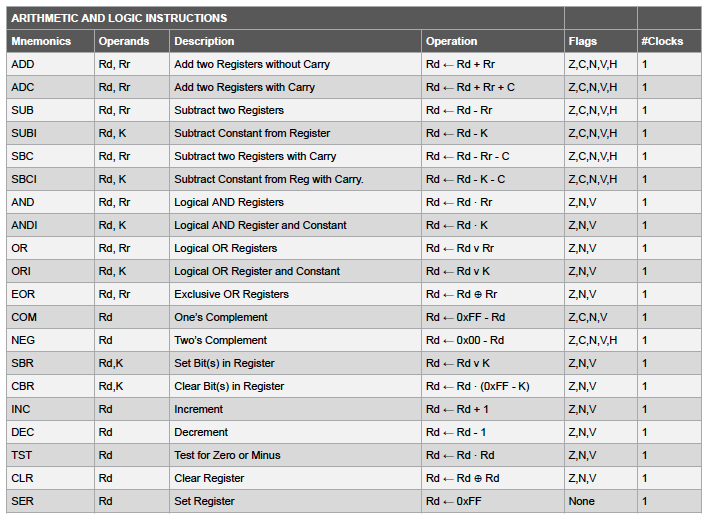
8-Bit AVR® Arithmetic Logic Unit

Last modified by Microchip on 2023/11/10 11:09

The high-performance AVR® Arithmetic Logic Unit (ALU) operates in direct connection with all the 32 [General Purpose Working registers](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/structure/gpr/). Within a single clock cycle, arithmetic operations between General Purpose registers, or between a register and an immediate, are executed. The ALU operations are divided into three main categories: arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format.

ALU Instruction Set



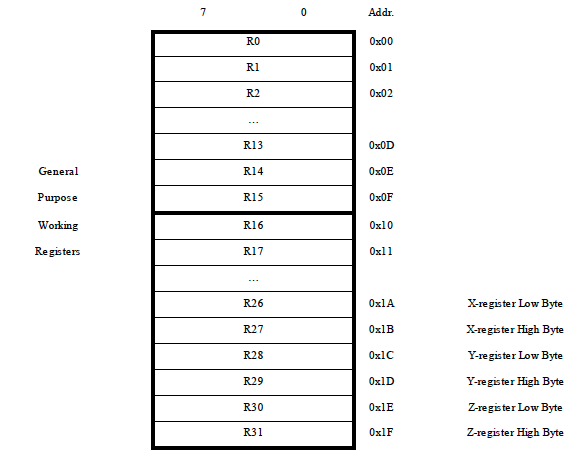
# 8-Bit AVR® General Purpose Registers

Last modified by Microchip on 2023/11/09 09:02

The AVR® register file structure is optimized for the AVR Enhanced Reduced Instruction Set Computer (RISC) instruction set. In order to achieve the required performance and flexibility, the following I/O schemes are supported by the register file:

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

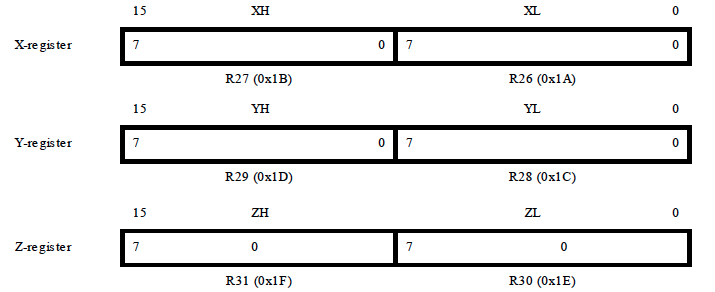
## **AVR CPU General Purpose Working Registers**



Most of the instructions operating on the register file have direct access to all registers and most of them are single-cycle instructions. Each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user data space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access to the registers, as the X-, Y-, and Z-pointer registers can be set to index any register in the file.

## **The X-register, Y-register, and Z-register**

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

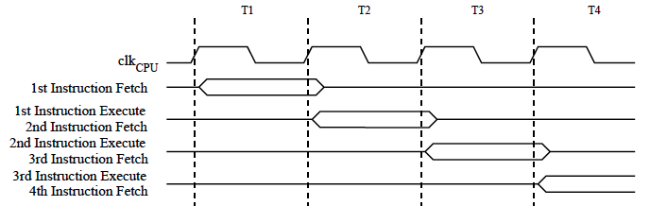


# 8-bit AVR® Instruction Timing

Last modified by Microchip on 2023/11/09 09:02

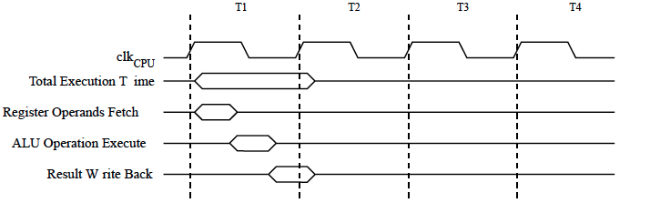
The AVR® Central Processing Unit (CPU) is driven by the CPU clock clkCPU, directly generated from the selected clock source for the chip. No internal clock division is used. Parallel instruction fetches and instruction executions are enabled by the Harvard architecture and the fast-access register file concept. This is he basic pipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

## **The Parallel Instruction Fetches and Instruction Executions**



In a single clock cycle, an Arithmetic Logic Unit (ALU) operation using two register operands is executed and the result is stored back to the destination register.

## **Single Cycle ALU Operation**



# 8-bit AVR® Microcontrollers Fuses

Last modified by Microchip on 2023/12/04 15:03

AVR® microcontroller (MCU) fuses are the locations in non-volatile memory that define the hardware configuration of an AVR device.  
  
Fuses are placed in a select section of memory and consist of a few registers. Each bit of the register represents a different fuse setting. Detailed information on which fuses are available in the different programming modes and their functions can be found in the device datasheet. Instruction clock speed, watchdog timer, and debug mode are just a few of the fuse settings available on most devices.  
  
Fuses are changed at programming time with a connected programmer and MPLAB X IDE. They are latched in place after programming and also at the power-up of the device.  
  
Unless careful consideration is given to which Fuse bits are programmed, it is easy for the novice programmer to [brick their device](https://developerhelp.microchip.com/xwiki/bin/view/software-tools/mcu-dev-boards/unbrick-avr/) (i.e., render the MCU un-programmable/un-debuggable in his/her application circuit).

<https://youtu.be/YG5QypvlMLQ>

## **AVR Fuses Summary**

The device has three Fuse bytes. The following tables describe briefly the functionality of all the fuses and how they are mapped into the Fuse bytes. Note that the fuses are read as logical zero, “0”, if they are programmed.

## **Extended Fuse Byte for ATmega328PB**

|  |  |  |  |
| --- | --- | --- | --- |
| **Extended Fuse Byte** | **Bit No.** | **Description** | **Default Value** |
| - | 7 | - | 1 |
| - | 6 | - | 1 |
| - | 5 | - | 1 |
| - | 4 | - | 1 |
| CFD | 3 | Disable Clock Failure Detection | 0 (programmed, CFD disable) |
| BODLEVEL2(1) | 2 | Brown-out Detector trigger level | 1 (unprogrammed) |
| BODLEVEL1(1) | 1 | Brown-out Detector trigger level | 1 (unprogrammed) |
| BODLEVEL0(1) | 0 | Brown-out Detector trigger level | 1 (unprogrammed) |

**Note:**

1. Please refer to Table BODLEVEL Fuse Coding in System and Reset Characteristics for BODLEVEL Fuse decoding in the device datasheet.

## **Fuse High Byte for ATmega328PB**

|  |  |  |  |
| --- | --- | --- | --- |
| **High Fuse Byte** | **Bit No.** | **Description** | **Default Value** |
| RSTDISBL(1) | 7 | External Reset Disable | 1 (unprogrammed) |
| DWEN | 6 | debugWIRE Enable | 1 (unprogrammed) |
| SPIEN(2) | 5 | Enable Serial Program and Data Downloading | 0 (programmed, SPI programming enabled) |
| WDTON(3) | 4 | Watchdog Timer Always On | 1 (unprogrammed) |
| EESAVE | 3 | EEPROM memory is preserved through the Chip Erase | 1 (unprogrammed), EEPROM not reserved |
| BODLEVEL2(4) | 2 | Brown-out Detector trigger level | 1 (unprogrammed) |
| BODLEVEL1(4) | 1 | Brown-out Detector trigger level | 1 (unprogrammed) |
| BODLEVEL0(4) | 0 | Brown-out Detector trigger level | 1 (unprogrammed) |

**Note:**

1. Please refer to Alternate Functions of Port C in I/O-Ports chapter for description of RSTDISBL Fuse.  
2. The SPIEN Fuse is not accessible in serial programming mode.  
3. Please refer to WDTCSR – Watchdog Timer Control Register for details.  
4. Please refer to Table. BODLEVEL Fuse Coding in System and Reset Characteristics for BODLEVEL Fuse decoding.

## **Fuse Low Byte for ATmega328PB**

|  |  |  |  |
| --- | --- | --- | --- |
| **High Fuse Byte** | **Bit No.** | **Description** | **Default Value** |
| CKDIV8(4) | 7 | Divide clock by 8 | 0 (programmed) |
| CKOUT(3) | 6 | Clock output | 1 (unprogrammed) |
| SUT1 | 5 | Select start-up time | 1 (unprogrammed)(1) |
| SUT0 | 4 | Select start-up time | 0 (programmed)(1) |
| CKSEL3 | 3 | Select Clock source | 0 (programmed)(2) |
| CKSEL2 | 2 | Select Clock source | 0 (programmed)(2) |
| CKSEL1 | 1 | Select Clock source | 1 (unprogrammed)(2) |
| CKSEL0 | 0 | Select Clock source | 0 (programmed)(2) |

**Note:**

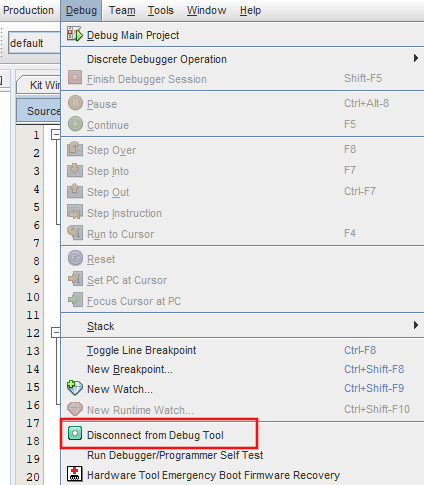
1. The default value of SUT[1:0] results in maximum start-up time for the default clock source. See the Fuse Low Byte for ATmega328PB table. Start-up times for the internal calibrated RC Oscillator clock selection in Calibrated Internal RC Oscillator of System Clock and Clock Options chapter for details.  
2. The default setting of CKSEL[3:0] results in internal RC Oscillator @ 8MHz. See the Fuse Low Byte for ATmega328PB table. Internal Calibrated RC Oscillator Operating Modes in Calibrated Internal RC Oscillator of the System Clock and Clock Options chapter for details.  
3. The CKOUT Fuse allows the system clock to be output on PORTB0. Please refer to Clock Output Buffer section in the System Clock and Clock Options chapter for details.  
4. Please refer to System Clock Prescaler section in the System Clock and Clock Options chapter for details.

The status of the Fuse bits is not affected by Chip Erase. Note that the Fuse bits are locked if Lock bit1 (LB1) is programmed. Program the Fuse bits before programming the Lock bits.

## **AVR Fuse Programming in MPLAB® X**

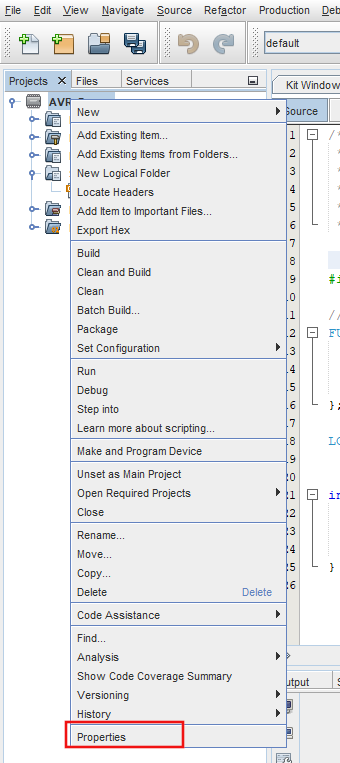
##### **Exit Debug Mode**

The fuses cannot be programmed while debugging a project.  The debug tool also needs to be disconnected.  Select**Debug > Disconnect from Debug Too**l as shown.



##### **Switch communication from debugWIRE to ISP**

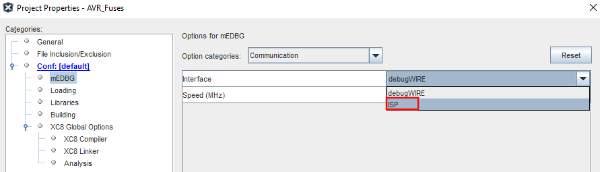
Right-click on the **Project Name** in the file explorer window and select **Properties** and then select **OK**at the bottom.



If a dialog appears asking if you would like to switch to ISP, select **YES**.

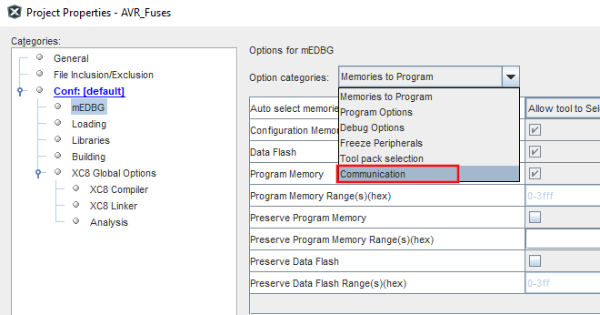
##### **Switch to ISP**

In the next dialog, select the pull down next to **Interface** and select **ISP** if not already selected.



##### **Select mEDBG and Communication**

In the next dialog box, select **mEDBG** and then scroll to **Communication** in the **Option Categories**pull-down menu, and then select **OK**.

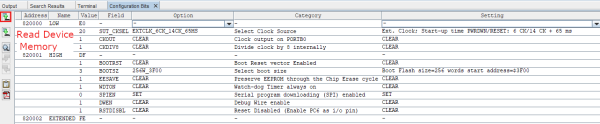
****

##### **Disconnect and Reconnect the Device**

The power needs to be cycled on the device to restart this communication in ISP mode.  Briefly unplug the device from your PC and reconnect.  You will now be able to read the current status of the configuration fuses.

##### **Read the Fuses**

Select **Window > Target Memory Views > Configuration Bits**.A new window will appear showing the status of each of the fuses.  They will appear RED until the device is read.  Now select the **Read Device Icon** in the upper left of the new window.  If successful, the readings will turn to black text.



##### **Making a Change to the Fuses**

Some changes to the fuses can result in bricking of your device.  See the "[Bricking a Device](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/structure/fuses/#brick)" section of this page for more details.

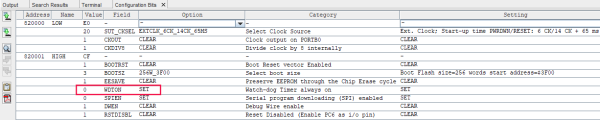
Currently, the fuses are configured to turn off the Watch Dog Timer.  We know this because in the **WDTON** row, the setting indicates **CLEAR**. We will turn it in on in the following example by changing it to **SET**.

The **WDTON** option is set by the High Fuse setting of bit 4.  The High Fuse currently has a hex value of 0xDF.  If we change bit 4 to a 0, the hex value becomes 0xCF.

Create a new project or add the following code toward the top of an existing project (just below the #include statements)...

*//Set Fuses to configure the device*FUSES = {  
 .low = 0xE0, *// LOW {SUT\_CKSEL=EXTCLK\_6CK\_14CK\_65MS, CKOUT=CLEAR, CKDIV8=CLEAR}* .high = 0xCF, *// HIGH {BOOTRST=CLEAR, BOOTSZ=256W\_3F00, EESAVE=CLEAR, WDTON=SET, SPIEN=SET, DWEN=CLEAR, RSTDISBL=CLEAR}* .extended = 0xFE, *// EXTENDED {BODLEVEL=1V8, CFD=CFD\_ENABLED}*};

Press the Make and Program device button and your new Configuration bits should reflect that the **WDTON**is now set.



Other fuses can be assigned **SET** or **CLEAR** in the same manner.

## **AVR Fuse Programming in Atmel Studio**

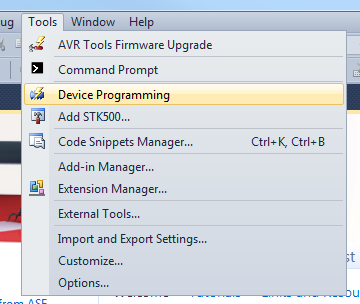
The **Device Programming** window (also known as the programming dialog), gives you the most low-level control over the debugging and programming tools. With it, you can program the device's different memories, fuses and lock bits, erase memories, and write user signatures. It can also adjust some of the starter kit properties, such as voltage and clock generators.

The programming dialog is accessible from the **Device Programming** icon on the standard toolbar or in the **Tools > Device Programming** drop-down menu selection.

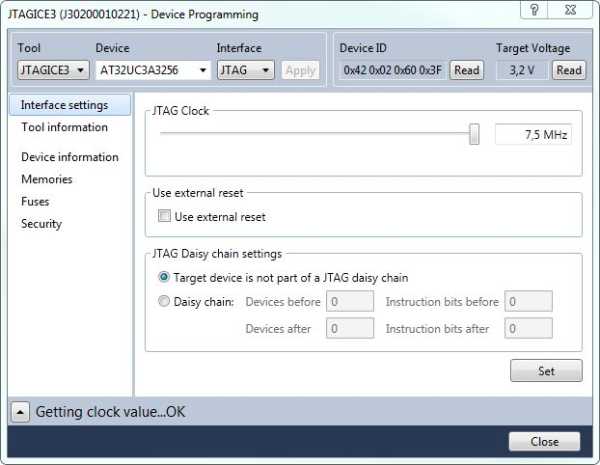
##### **Device Programming Icon**



##### **Menu Selection**

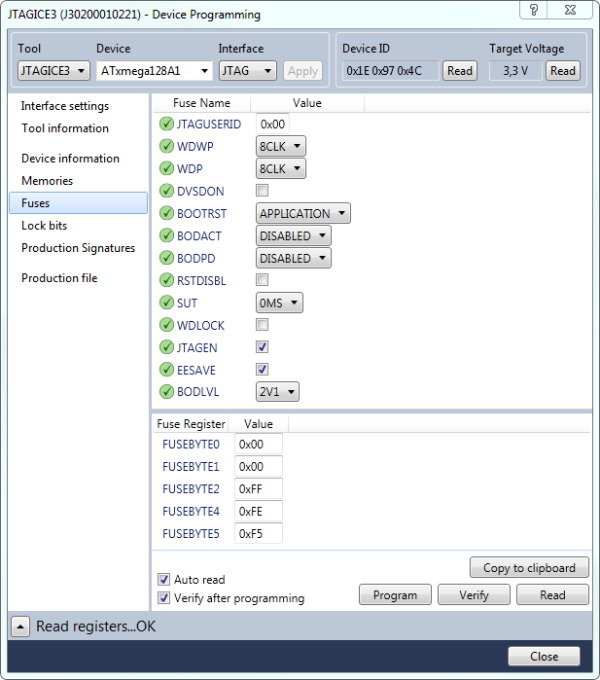


The **Device Programming** window will appear with various programming options listed on the left.



##### **Select the Fuses option**

The Fuses pane will appear showing the fuses of the selected device. Fuse settings are presented as checkboxes or as drop-down lists. The Fuse Register settings also appear in the lower pane as hexadecimal values.



##### **Read**

Press the **Read** button, in the lower right corner, to read the current value of the fuses. If the Auto read box is checked, the fuse settings will be read from the device each time you enter the fuse page.

##### **Change**

The fuse settings can be changed via the dropdown boxes. Some selections require a box to be checked or unchecked.

A fuse bit that is selected is set to 0 in the register.

##### **Program**

After any changes are made to the settings, press the **Program** button to write the current fuse setting to the device. If the Verify after programming box is checked, the settings will be verified after a programming operation is completed.

##### **Latch**

The fuse values are latched when the device enters programming mode, and changes in the fuse values will have no effect until the part leaves the Programming mode. This does not apply to the EESAVE Fuse, which will take effect once it is programmed. The fuses are also latched on Power-Up in Normal mode.

## **Adding Fuse Settings to .elf Production File**

The fuse API allows a user to specify the fuse settings for the specific AVR device for which they are compiling. These fuse settings will be placed in a special section in the ELF output file after linking.

Programming tools can take advantage of the fuse information embedded in the ELF file, by extracting this information and determining if the fuses need to be programmed before programming the Flash and EEPROM memories. This also allows a single ELF file to contain all the information needed to program an AVR MCU.

### Converting Studio .elf format to .hex for Programming with MPLAB® X Tools

If you look at the build output in Studio IDE, you’ll see how the .hex file is made from the .elf file, e.g.:

"E:\Program Files (x86)\Atmel\Studio\7.0\toolchain\avr8\avr8-gnu-toolchain\bin\avr-objcopy.exe" -O ihex -R .eeprom -R .fuse -R .lock -R .signature -R .user\_signatures "ATtiny416.elf" "ATtiny416.hex"

This means: make the .hex but discard .eeprom, .fuse, .lock, .signature, and .user\_signature files.

If you want a HEX file with those sections left over for use with MPLAB X IDE programming tools, make a post-build step, e.g.:

"$(ToolchainDir)\avr-objcopy.exe" -O ihex -R .eeprom -R .lock -R .signature -R user\_signatures "$(OutputDirectory)\$(OutDir)\$(OutputFileName)$(OutputFileExtension)" "$(OutputDirectory)\$(OutputFileName).with-fuse.hex"

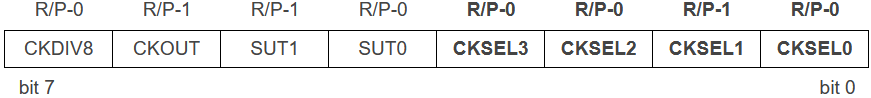
​

The reason for the post-build is that Atmel Studio considers .elf as the production file, as does the 'classic AVR environment,' while the MPLAB X environment uses .hex files.

## **Bricking a Device**

The following fuse bytes (and default fuse bit settings) are shown for the [ATmega328PB](http://www.microchip.com/wwwproducts/en/ATmega328PB). Key fuse bit settings which could brick the device are highlighted:

### FUSE LOW Byte

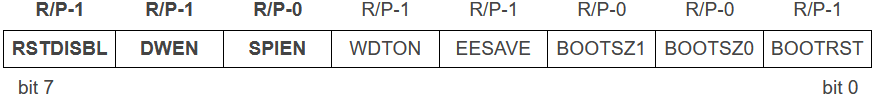


* **CKSELn [n=3:0]** – Selects clock source for the system clock.
* SUTn [n=1:0] - Selects the delay period from when the External Reset is released (not active anymore) until the Internal Reset is released.
* CKOUT – Enables clock output on PB0.
* CKDIV8 – Configures the CPU clock to be pre-scaled by 8.

​

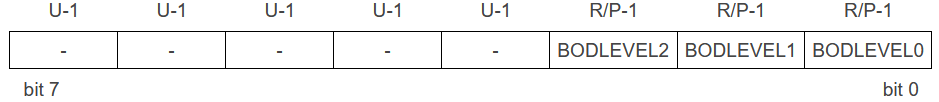
* Fuse bit = 1 is un-programmed (inactive/disabled).
* Fuse bit = 0 is programmed (active/enabled).

### FUSE HIGH Byte



* BOOTRST - If you use a bootloader to flash MCU, this bit must be enabled.
* BOOTSZn [n=1:0] - These bits select the bootloader section size.
* EESAVE – Exclude the EEPROM during a chip erase procedure.
* WDTON – Enable Watchdog Timer by HW.
* **SPIEN** – Enable/Disable In-Circuit Serial Programming (ISP) mode.
* **DWEN** – Enable/Disable DebugWire debug interface.
* **RSTDISBL** – Enable/Disable nRESET pin usage as IO.

### FUSE EXTENDED Byte



* BODLEVELn [n=2:0] - They select the brown-out voltage level when the VDD supply is no longer suitable for operation and the MCU is reset.

## **Learn More**

* [Un-Brick an AVR® Xplained Board](https://developerhelp.microchip.com/xwiki/bin/view/software-tools/mcu-dev-boards/unbrick-avr/)

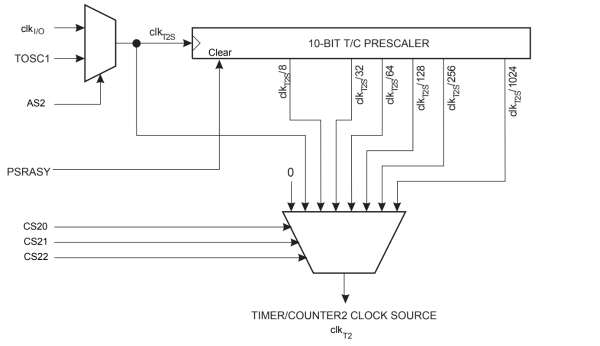
# 8-Bit AVR® Real Time Counter (RTC)

Last modified by Microchip on 2023/11/09 09:02

## **Real Time Counter (RTC)**

AVR® devices have a Timer/Counter Type 2 (TC2) general-purpose, dual-channel, 8-bit timer/counter module. This timer/counter allows clocking from an external 32 kHz watch crystal, independent of the I/O. This allows the timer to run as a relatively accurate Real Time Counter (RTC).

<https://youtu.be/-8Qk3eDpr6Q>



### Clock Source

The clock source for TC2 is named clkT2S. It is by default connected to the main system I/O clock, clkI/O. By writing a 1 to the Asynchronous TC2 bit in the Asynchronous Status Register (ASSR.AS2), TC2 is asynchronously clocked from the TOSC1 pin. This enables the use of TC2 as an RTC.

When AS2 is set, pins TOSC1 and TOSC2 are disconnected from the I/O Port. A crystal can then be connected between the TOSC1 and TOSC2 pins to serve as an independent clock source for TC2. The oscillator is optimized for use with a 32.768 kHz crystal.

### Prescaler

For TC2, the possible prescaled selections are clkT2S/8, clkT2S/32, clkT2S/64, clkT2S/128, clkT2S/256, and clkT2S/1024. Additionally, clkT2S, as well as 0 (stop) may be selected. The prescaler is reset by writing a 1 to the Prescaler Reset TC2 bit in the General TC2 Control Register (GTCCR.PSRASY). This allows the user to operate with a defined prescaler.

## **Additional Information**

* [Application Note 1259: Real Time Clock (RTC) Using the Asynchronous Timer](https://www.microchip.com/wwwAppNotes/AppNotes.aspx?appnote=en591232)

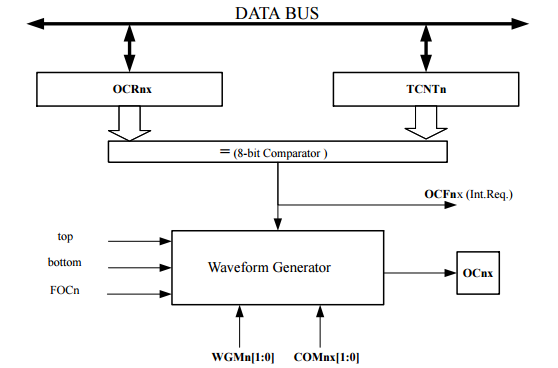
# AVR Timer Compare Register Double Buffer

Last modified by Microchip on 2023/11/09 09:02

AVR® devices have a general-purpose 8-bit Timer/Counter module, with two independent Output Compare Units, and Pulse Width Modulation (PWM) support. It allows accurate program execution timing (event management) and wave generation.

<https://youtu.be/XCPwsDx6oyo>

A simplified block diagram of the 8-bit Timer/Counter Output Compare Unit is shown.



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

Adjustments to the PWM period register value or compare register value can be random in the application. If an update occurs at the same time as a timer compare match occurs, the match may be missed. This can cause the PWM control to hit a max duty cycle of 100% which can cause issues in the control application. For this reason, AVR timers have optional double buffers.

## **Double Buffer Summary**

### Download

* [Double Buffer Example Project the Video](https://microchiptechnology.sharepoint.com/:u:/s/DeveloperHelp/EQnaGDx3PiNEqqYngi2JmZIBiznJrt5MWK6nlVtFVENq7A?e=uoJgTY)
* [ATmega324PB datasheet double buffer reference](http://www.atmel.com/Images/Atmel-42546-8-bit%20AVR%20ATmega324PB_Datasheet.pdf#page=139)

## **Double Buffer Operation**

The Output Compare Registers (OCR0x) are double-buffered when using any of the PWM modes. When double buffering is enabled, the CPU has access to the OCR0x Buffer Register. The double buffering synchronizes the update of the OCR0x Compare Registers to either top or bottom of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The double buffered OCR0x are compared with the Timer/Counter value at all times. The result of the comparison can be used by the Waveform Generator to generate a PWM or variable frequency output on the Output Compare pins (OC0A and OC0B). The compare match event will also set the Compare Flag (OCF0A or OCF0B) which can be used to generate an Output Compare interrupt request.

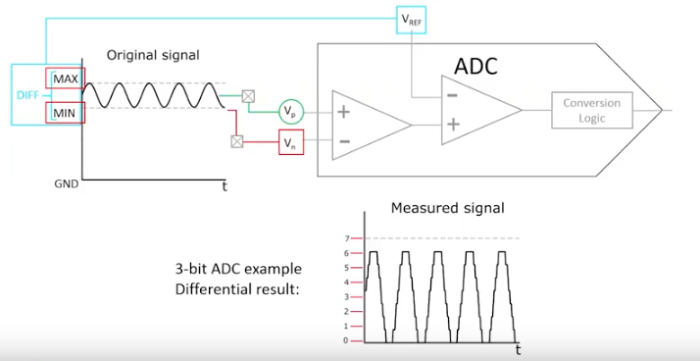
# AVR® ADC Differential Mode

Last modified by Microchip on 2023/11/09 09:02

A differential Analog-to-Digital Converter (ADC) measures the voltage difference between two signals. An ADC typically measures the voltage between the signal and ground but in differential mode, the ground pin is actually connected to another part of the circuit so the ADC can measure the difference between the two signals. This is often used to measure a small signal with a large offset. Using a differential input allows the ADC to measure a smaller portion of the signal.  
 <https://youtu.be/rh_8gBkiQk4>

## **Differential Setup**

The differential inputs are run through a gain amplifier to increase the signal size to the converter. Some AVR**®** devices have pins with adjustable gain. When using differential gain channels, certain aspects of the conversion need to be taken into consideration. Differential channels should not be used with an Analog Reference voltage (AREF) less than 2V. On AVR devices, differential conversions are synchronized to the internal clock CKADC2 equal to half the ADC clock. This synchronization is done automatically by the ADC interface in such a way that the sample-and-hold occurs at a specific phase of the CKADC2 clock.



## **Timing Critical Measurements**

On a typical AVR device with differential mode, a conversion you initiate (that is, all single conversions, and the first free-running conversion) when the CKADC2 clock signal is low will take the same amount of time as a single-ended conversion (13 ADC clock cycles from the next prescaled clock cycle). A conversion you initiate when the CKADC2 clock signal is high will take 14 ADC clock cycles due to the synchronization mechanism.

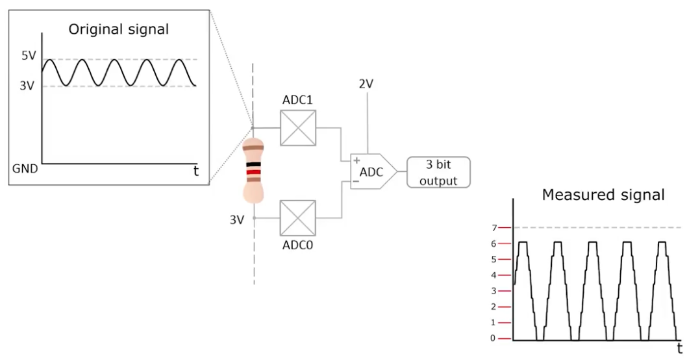
In free-running mode, a new conversion is initiated immediately after the previous conversion completes, and since CKADC2 is high at this time, all automatically started (that is, all but the first) free-running conversions will take 14 ADC clock cycles.

If differential gain channels are used and conversions are started by auto triggering, the ADC must be switched off between conversions. When auto triggering is used, the ADC prescaler is reset before the conversion is started. Since the gain stage is dependent on a stable ADC clock prior to the conversion, this conversion will not be valid. By disabling and then re-enabling the ADC between each conversion (setting the ADEN bit in ADCSRA to 0 and then to 1), only extended conversions are performed. The result from the extended conversions will be valid.

## **Typical Application**

Reading the voltage across a resistor with a steady three-volt offset is a simple example where a differential input can measure the signal above the DC offset.

**Note:** The Differential Schematic at 1:32 in the video above has the ADC pins reversed. Refer to the schematic below for proper connections.



## **Additional Information**

* [AN\_8456 - Understanding ADC Parameters](http://www.microchip.com/wwwAppNotes/AppNotes.aspx?appnote=en590903)
* [AN\_2559 - Characterization and Calibration of the ADC on an AVR](http://www.microchip.com/wwwAppNotes/AppNotes.aspx?appnote=en591791)
* [Example Project using the ATmega324PB](https://microchiptechnology.sharepoint.com/:u:/s/DeveloperHelp/EVt_rq4aFspEll6R4NJq8joBdAmyrgQe8kFfCWjsB0-Acg?e=8LfRgn)

# AVR® ADC Noise Reduction Mode

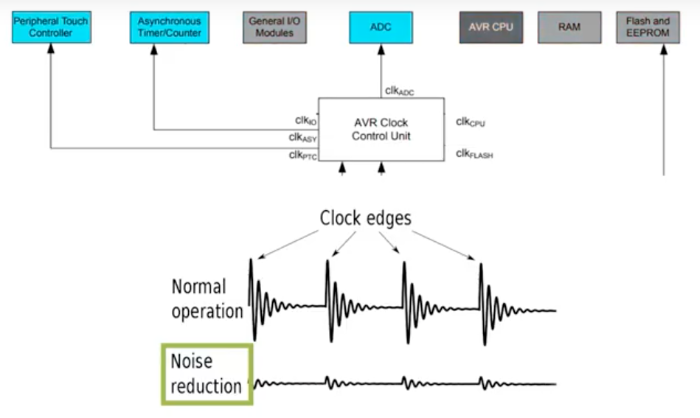
Last modified by Microchip on 2023/11/09 09:02

AVR® devices have an Analog to Digital Converter (ADC) Noise Reduction mode, that stops the CPU and all I/O modules except asynchronous timer, PTC, and ADC, to minimize switching noise during ADC conversions. It's used when a high-resolution ADC measurement is required. ADC measurements are then implemented when the core is put to sleep.

<https://youtu.be/VeodIyVU5Ic>

## **Entering/Exiting ADC Noise Reduction**

The SLEEP instruction makes the MCU enter ADC Noise Reduction mode, stopping the CPU but allowing the ADC, the external interrupts, the two-wire serial Interface address watch, Timer1, and the Watchdog to continue operating (if enabled). This sleep mode basically halts clkI/O, clkCPU, and clkFLASH, while allowing the other clocks to run. This improves the noise environment for the ADC, enabling higher-resolution measurements. If the ADC is enabled, a conversion starts automatically when this mode is entered.



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

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

​

**Note:** Timer/Counter only keeps running in Asynchronous mode.

## **Additional Information**

* [Example Project using the ATmega324PB](https://microchiptechnology.sharepoint.com/:u:/s/DeveloperHelp/EY68XJjWqCdIuOobM6xXFV0BzVMSkNl-gP8BUBcHEVAAlA?e=amCSSZ)

# AVR® ADC Operating Modes

Last modified by Microchip on 2023/11/09 09:02

TheAVR® Analog to Digital Conversion (ADC) Peripheral Operation Modes are just some of the features this peripheral offers. Refer to the ADC section of the datasheet for the device you've selected for more detail. <https://youtu.be/YPtjZmXxK1M>

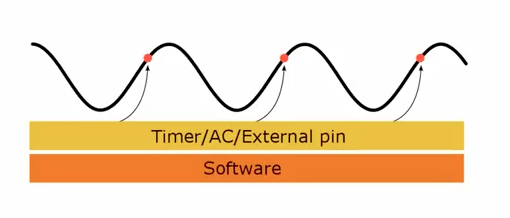
### Single Conversion Mode

A single conversion is started by writing a to the Power Reduction ADC bit in the Power Reduction Register PRR0.PRADC, and writing a 1 to the ADC Start Conversion bit in the ADC Control and Status Register A ADCSRA.ADSC. The ADCS bit will stay high as long as the conversion is in progress and will be cleared by hardware when the conversion is completed. If a different data channel is selected while a conversion is in progress, the ADC will finish the current conversion before performing the channel change.

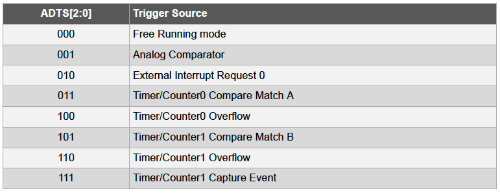
# Single Conversion Mode

### Auto Triggering Mode

A conversion can be triggered automatically by various sources. Auto Triggering is enabled by setting the ADC Auto Trigger Enable bit ADCSRA.ADATE.



The trigger source is selected by setting the ADC Trigger Select bits in the ADC Control and Status Register B ADCSRB.ADTS. See the description of the ADCSRB.ADTS for a list of available trigger sources.



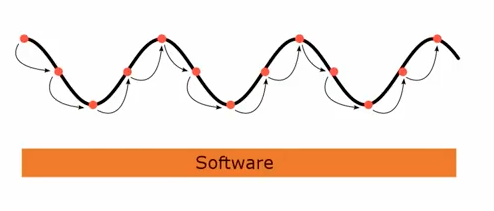
When a positive edge occurs on the selected trigger signal, the ADC prescaler is reset and a conversion is started. This provides a method of starting conversions at fixed intervals. If the trigger signal is still set when the conversion completes, a new conversion will not be started. If another positive edge occurs on the trigger signal during conversion, the edge will be ignored.

​

An interrupt flag will be set even if the specific interrupt is disabled or the Global Interrupt Enable bit in the AVR Status Register SREG.I is cleared. A conversion can thus be triggered without causing an interrupt. However, the Interrupt Flag must be cleared in order to trigger a new conversion at the next interrupt event.

### Free Running Mode

When Free Running Mode is selected the ADC Interrupt Flag is used as a trigger source and makes the ADC start a new conversion as soon as the ongoing conversion has finished. This Free Running mode, constantly samples and updates the ADC Data Register. The first conversion must be started by writing a 1 to ADCSRA.ADSC. In this mode, the ADC will perform successive conversions independently of whether the ADC Interrupt Flag ADIF is cleared or not.



## **Additional Information**

* [Example Project Using the ATmega324PB](https://microchiptechnology.sharepoint.com/:u:/s/DeveloperHelp/EWQhZvdzVcZChm_ZdgUwM9YBeCELPSFkXOXyKJcoP1M_qQ?e=I7n4lV)
* [AVR126: ADC of megaAVR® in Single-Ended Mode](http://ww1.microchip.com/downloads/en/AppNotes/00002538A.pdf)

# AVR® ADC Voltage Reference

Last modified by Microchip on 2023/11/09 09:02

The Analog Reference (AREF) is the reference voltage to the on-chip Analog to Digital Converter (ADC) on AVR® devices. The reference voltage for the ADC, VREF, indicates the voltage range of the ADC conversion. Single-ended channels that exceed VREF results in a maximum conversion value result. VREF can be measured at the AREF pin with a high-impedance voltmeter. <https://youtu.be/fN3J6wxijrE>

## **VREF Options**

VREF can be selected as either AVCC, internal 1.1 V reference, or external AREF pin.

* AVCC is the voltage connected to the AVCC pin which is internally connected to the ADC through a passive switch.
* The internal 1.1 V reference is generated from the internal bandgap reference (VBG) through an internal amplifier.
* The external AREF pin is directly connected to the ADC, and the reference voltage can be made more immune to noise by connecting a capacitor between the AREF pin and ground.

If a fixed voltage source is connected to the external AREF pin, the application may not use the other reference voltage options in the application, as they will be shorted to the external voltage.

If no external voltage is applied to the AREF pin, you may switch between AVCC and 1.1 V as the reference selection. The first ADC conversion result after switching the reference voltage source may be inaccurate, and you are advised to discard this result.

​

If differential channels are used, the selected reference should not be closer to AVCC than indicated in the "ADC Characteristics of Electrical Characteristics" in the device datasheet.

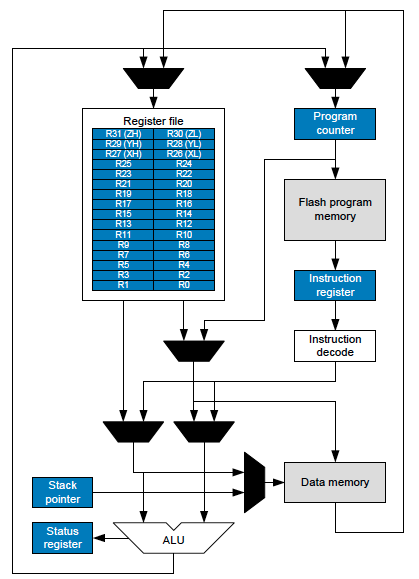
### Additional Information

* [AVR353: Voltage Reference Calibration and Voltage ADC Usage](http://www.microchip.com/wwwAppNotes/AppNotes.aspx?appnote=en591688)
* [Example Project using the ATmega324PB](https://microchiptechnology.sharepoint.com/:u:/s/DeveloperHelp/EYnqfALOsXtJuqs5Xg5W3lUBWjg0agaKtJaBUKvK5xB4dA?e=71D6ze)

# AVR® CPU Core

Last modified by Microchip on 2023/11/10 11:09

The main function of the AVR® Central Processing Unit (CPU) core is to ensure correct program execution. The CPU must, therefore, be able to access memories, perform calculations, control peripherals, and handle interrupts.



## **AVR Core**

In order to maximize performance and parallelism, the AVR uses a Harvard architecture with separate memories and buses for program and data. Instructions in the program memory are executed with single-level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables [instructions to be executed in every clock cycle](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/structure/instruction-timing/). The program memory is In-System Reprogrammable Flash memory.

## **Registers**

The fast-access Register file contains 32 x 8-bit [General Purpose Working registers](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/structure/gpr/) with a single clock cycle access time. Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing, enabling efficient address calculations. One of these address pointers can also be used as an address pointer for lookup tables in Flash program memory. These added function registers are the 16-bit X-, Y-, and Z-registers.

## **Arithmetic Logic Unit (ALU)**

The [ALU](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/structure/alu/) supports arithmetic and logic operations between registers or between a constant and a register. Single clock cycle access time allows single-cycle ALU operations. In a typical ALU operation, two operands are output from the Register file, the operation is executed and the result is stored back in the Register file in one clock cycle. Single register operations can also be executed in the ALU. After an arithmetic operation, the Status register is updated to reflect information about the result of the operation. Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction.

## **Memory**

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

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

During interrupts and subroutine calls, the return address Program Counter (PC) is stored on the [stack](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/structure/stack/). The stack is effectively allocated in the general data SRAM and consequently, the stack size is only limited by the total SRAM size and the usage of the SRAM.

All user programs must initialize the Stack Pointer (SP) in the Reset routine (before subroutines or interrupts are executed). The SP is read/write  
accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture.

The I/O memory space contains 64 addresses for CPU peripheral functions as Control registers, Serial Peripheral Interface (SPI), and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the Register file, 0x20 - 0x5F. In addition, this device has extended I/O space from 0x60 - 0xFF in SRAM.

## **Interrupts**

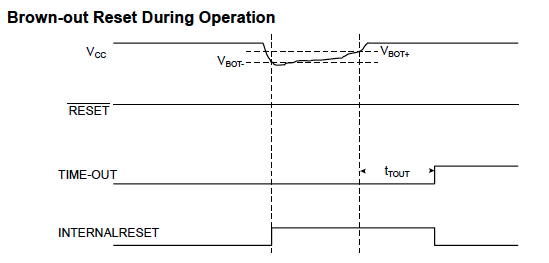
A flexible [Interrupt module](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/structure/interrupts/) has its Control registers in the I/O space with an additional Global Interrupt Enable bit in the [Status register](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/structure/status/). All interrupts have a separate interrupt vector in the interrupt vector table. The interrupts have priority in accordance with their interrupt vector position. The lower the interrupt vector address, the higher the priority.

# AVR® Device Brown-out Detection (BOD)

Last modified by Microchip on 2023/11/09 09:02

Many AVR® devices have an on-chip Brown-out Detection (BOD) circuit for monitoring the Operating Voltage (VCC) level during operation. By comparing the VCC to a fixed trigger level it can determine if the device needs to be put into reset mode to prevent erratic operation. <https://youtu.be/G90Xd2ssxog>

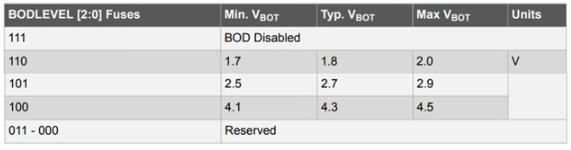
## **BOD Operation**



The trigger level has a hysteresis to ensure spike-free BOD. When the BOD is enabled, and VCC decreases to a value below the trigger level (VBOT-), the Brown-out Reset is immediately activated. When VCC increases above the trigger level (VBOT+), the delay counter starts the MCU after the Time-out period tTOUT has expired. The BOD circuit will only detect a drop in VCC if the voltage stays below the trigger level for longer than the minimum pulse width Brown-out Detection (tBOD) value specified in the device's data sheet.

## **BOD Fuse Settings**

The trigger level for the BOD can be selected by the BODLEVEL Fuses when the device is programmed. This setting cannot be changed while running the application software.



## **BOD Disable**

When the Brown-out Detector (BOD) is enabled by BODLEVEL fuses, the BOD actively monitors the power supply voltage even during a sleep period. To save power, it is possible to disable the BOD by software for some of the sleep modes. The sleep mode power consumption will then be at the same level as when BOD is globally disabled by fuses.

If BOD is disabled in the software, the BOD function is turned off immediately after entering the sleep mode. Upon wake-up from sleep, BOD is automatically enabled again. This ensures safe operation in case the VCC level has dropped during the sleep period.

When the BOD has been disabled, the wake-up time from sleep mode will be approximately 60 μs to ensure that the BOD is working correctly before the MCU continues executing code.

BOD disable is controlled by the BOD Sleep bit in the MCU Control Register MCUCR.BODS. Setting this bit to '1' turns off the BOD in relevant sleep modes, while a '0' in this bit keeps BOD active. The default setting, MCUCR.BODS = 0, keeps BOD active.

## **Additional Information**

[AN1051 - AVR180: External Brown-Out Protection](https://www.microchip.com/en-us/application-notes?rv=1234a514) (Application Note)

# AVR® Device Peripheral Power Reduction Register

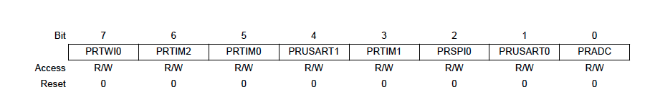
Last modified by Microchip on 2023/11/09 09:02

AVR® 8-bit microcontrollers include several sleep modes to save power. The AVR device can also lower power consumption by shutting down the clock, for select peripherals, via a register setting. That register is called the Power Reduction Register (PRR).

<https://youtu.be/S82BSPbYoVA>

### Example: Register from ATmega328PB.

Writing a logic one to a bit shuts down the peripheral clock.



* PRTWI0: Power Reduction TWI0
* PRTIM2: Power Reduction Timer/Counter2
* PRTIM0: Power Reduction Timer/Counter0
* PRUSART1: Power Reduction USART1
* PRTIM1: Power Reduction Timer/Counter1
* PRSPI0: Power Reduction Serial Peripheral Interface 0
* PRUSART0: Power Reduction USART0
* PRADC: Power Reduction ADC

The PRR provides a run time method to stop the clock to select individual peripherals. The current state of the peripheral is frozen and the I/O registers can not be read or written. Resources used by the peripheral when stopping the clock will remain committed, hence the peripheral should, in most cases, be disabled before stopping the clock. Waking up a module, which is done by clearing the bit in PRR, puts the module into the same state as before shutdown.

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

### Brief AVR Power Reduction Register Summary

Project Download: [Sample Project Mentioned in Video](https://microchiptechnology.sharepoint.com/:u:/s/DeveloperHelp/EV4p1L7UAhFJhMqUIU6iUJ8BFz_aIePRtfnulnJ1QjoNGw?e=bQmgMn)

## **Minimizing Power Consumption**

There are several possibilities to consider when trying to minimize power consumption in an AVR-controlled system. In general, sleep modes should be used as much as possible, and the sleep mode should be selected so that as few as possible of the device’s functions are operating. All functions that are not needed should be disabled. In particular, the following modules may need special consideration when trying to achieve the lowest possible power consumption:

### Analog to Digital Converter (ADC)

If enabled, the Analog-to-Digital Converter(ADC) will be enabled in all sleep modes. To save power, the ADC should be disabled before entering any sleep mode. When the ADC is turned off and on again, the next conversion will be an extended conversion.

### Analog Comparator

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

### Brown-Out Detector

If the Brown-Out Detector (BOD) is not needed by the application, this module should be turned off. If the BOD is enabled by the BODLEVEL Fuses, it will be enabled in all sleep modes, and consequently will always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption.

### Internal Voltage Reference

The Internal Voltage Reference will be enabled when needed by the Brown-Out Detector, the Analog Comparator, or the Analog-to-Digital Converter. If these modules are disabled as described in the sections above, the internal voltage reference will be disabled and it will not be consuming power. When turned on again, the user must allow the reference to start up before the output is used. If the reference is kept on in sleep mode, the output can be used immediately.

### Watchdog Timer

If the Watchdog Timer is not needed in the application, the module should be turned off. If the Watchdog Timer is enabled, it will be enabled in all sleep modes and hence always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption.

### Port Pins

When entering a sleep mode, all port pins should be configured to use minimum power. The most important consideration is to ensure that no pins drive resistive loads. In sleep modes where both the I/O clock (clkI/O) and the ADC clock (clkADC) are stopped, the input buffers of the device will be disabled. This ensures that no power is consumed by the input logic when not needed. In some cases, the input logic is needed for detecting wake-up conditions, and it will then be enabled. If the input buffer is enabled and the input signal is left floating or has an analog signal level close to VCC/2, the input buffer will use excessive power.

For analog input pins, the digital input buffer should be disabled at all times. An analog signal level close to VCC/2 on an input pin can cause significant current even in active mode. Digital input buffers can be disabled by writing to the Digital Input Disable Registers (DIDR0 for ADC, DIDR1 for AC).

### On-chip Debug System

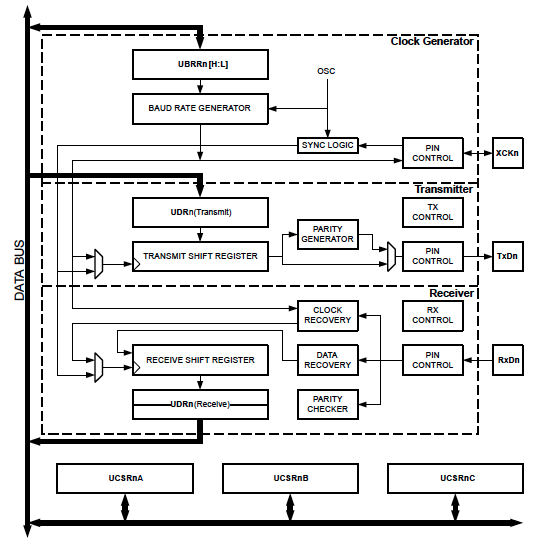
If the on-chip debug system is enabled by the DWEN Fuse and the chip enters sleep mode, the main clock source is enabled and always consumes power. In the deeper sleep modes, this will contribute significantly to the total current consumption.

# AVR® Device USART Introduction

Last modified by Microchip on 2023/11/09 09:02

AVR® devices include at least one and sometimes more Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) blocks. In the block diagram of the USART, the Central Processing Unit (CPU) accessible I/O Registers and I/O pins are shown. <https://youtu.be/DRqzr0dru8M>

## **USART Block Diagram**



The dashed boxes in the block diagram separate the three main parts of the USART:

* Clock Generator
* Transmitter
* Receiver

The Clock Generation logic consists of synchronization logic for the external clock input used by synchronous slave operation and the baud rate generator. The XCKn (Transfer Clock) pin is only used by Synchronous Transfer mode.

The Transmitter consists of a single write buffer, a serial Shift Register, Parity Generator, and Control logic for handling different serial frame formats. The write buffer allows a continuous transfer of data without any delay between frames.

The Receiver is the most complex part of the USART module due to its clock and data recovery units. The recovery units are used for asynchronous data reception. In addition to the recovery units, the Receiver includes a Parity Checker, Control logic, a Shift Register, and a two level receive buffer (UDRn). The Receiver supports the same frame formats as the Transmitter and can detect Frame Errors, Data OverRun, and Parity Errors.

## **Brief USART Introduction**

## **USART Features**

* Full Duplex Operation (Independent Serial Receive and Transmit Registers)
* Asynchronous or Synchronous Operation
* Master or Slave Clocked Synchronous Operation
* High Resolution Baud Rate Generator
* Supports Serial Frames with 5, 6, 7, 8, or 9 data bits and 1 or 2 stop bits
* Odd or Even Parity Generation and Parity Check Supported by Hardware
* Data OverRun Detection
* Framing Error Detection
* Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter
* Three Separate Interrupts on TX Complete, TX Data Register Empty, and RX Complete
* Multi-processor Communication mode
* Double Speed Asynchronous Communication mode
* Start Frame Detection

## **Serial Communication**

Serial Communication is a way to send data between two electronic devices using just two wires. The USART transmits and receives data using the standard non-return-to-zero (NRZ) format. NRZ is implemented with two levels: a Voltage Output High (VOH) or mark state which represents a ‘1’ data bit, and a Voltage Output Low (VOL) or space state which represents a ‘0’ data bit. NRZ refers to the fact that consecutively transmitted data bits of the same state stay at the same output level without returning to a zero or neutral level between each bit transmission. The Idle state puts the output pin at a mark (i.e. high state). Transmission can occur in a Synchronous or Asynchronous fashion.

## **Synchronous Clock Operation**

Synchronous serial communications are typically used in systems with a single master and one or more slaves. The master device contains the necessary circuitry for baud rate generation and supplies the clock for all devices in the system. Slave devices can take advantage of the master clock by eliminating the internal clock generation circuitry. There are two signal lines in Synchronous mode:

* Bidirectional data line
* Clock line

Slaves use the external clock supplied by the master to shift the serial data into and out of their respective receive and transmit registers. Since the data line is bidirectional, synchronous operation is half-duplex only. Half-duplex refers to the fact that master and slave devices can receive and transmit data but not both simultaneously. The USART can operate as either a master or slave device. Typically Start and Stop bits are not required for synchronous transmissions

When Synchronous mode is used (UMSEL = 1), the XCKn pin will be used as either clock input (Slave) or clock output (Master). The dependency between the clock edges and data sampling or data change is the same. The basic principle is that data input (RxDn pin) is sampled at the opposite XCKn clock edge of the edge the data output (TxDn pin) is changed.

## **Asynchronous Data Reception**

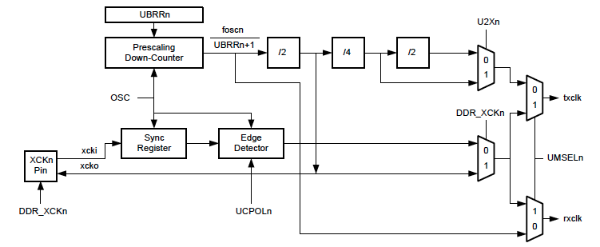
In Asynchronous communications, two pins are used. One is the Transmit (TxDn) and one is the Receive (RxDn). The TxDn of one device is connected to the RxDn of the second device. Each character transmission consists of one Start bit followed by five to nine data bits and is always terminated by one or more Stop bits. The Start bit is always a space and the Stop bits are always marks. Each transmitted bit persists for a period of 1/(Baud Rate). An on-chip dedicated Baud Rate Generator is used to derive standard baud rate frequencies from the system oscillator.

## **Clock Generation**

The Clock Generation logic generates the base clock for the Transmitter and Receiver. The USART supports four modes of clock operation:

* Normal asynchronous
* Double Speed asynchronous
* Master synchronous
* Slave synchronous

## **Clock Block Diagram**



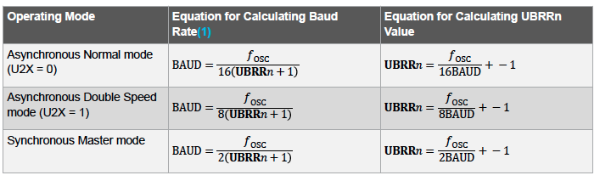
Signal description:

* txclk Transmitter clock (internal signal).
* rxclk Receiver base clock (internal signal).
* xcki Input from XCK pin (internal signal). Used for synchronous slave operation.
* xcko Clock output to XCK pin (internal signal). Used for synchronous master operation.
* OSC System clock frequency.

## **Baud Rate Generator**

Internal clock generation is used for the asynchronous and Synchronous Master modes of operation. The USART Baud Rate Register (UBRRn) and the down-counter connected to it, function as a programmable prescaler or baud rate generator. The down counter, running at system clock (fosc), is loaded with the UBRRn value each time the counter has counted down to zero or when the UBRRnL Register is written. A clock is generated each time the counter reaches zero. This clock is the baud rate generator clock output (= fosc/(UBRRn+1)).

The table below contains equations for calculating the baud rate (in bits per second) and for calculating the UBRRn value for each mode of operation using an internally generated clock source.



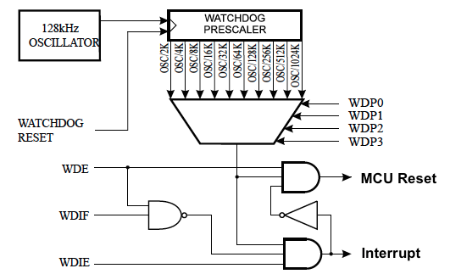
## **More Information**

[Example Project using the ATmega324PB](https://microchiptechnology.sharepoint.com/:u:/s/DeveloperHelp/EWFOCa52qP5MmkRot2kWnVwBFoLGa3c8KYEjhyaLikHqtg?e=Iqa6eV)  
[AN1451-1 - AVR306: Using the AVR UART in C on tinyAVR and megaAVR devices](https://www.microchip.com/en-us/application-notes?rv=1234a518) (Application Note)

# AVR® Device Watchdog Timer (WDT)

Last modified by Microchip on 2023/11/09 09:02

AVR® devices have an Enhanced Watchdog Timer (WDT) that runs on a separate oscillator from the main instruction clock. The WDT is essentially a counter that increments based on the clock cycles of an on-chip 128 kHz oscillator. The WDT forces an interrupt or a system reset when the counter reaches a given time-out value. In normal operation mode, the application code needs to issue a Watchdog Timer Reset (WDR) instruction to restart the counter before the time-out value is reached. If the system doesn't restart the counter, an interrupt or system reset will be issued.  
<https://youtu.be/E4m6nhWZNzA>



## **Brief WDT Summary**

### WDT Features

* Clocked from separate On-chip Oscillator
* Three operating modes:
  + Interrupt
  + System Reset
  + Interrupt and System Reset
* Selectable Time-out period from 16 milliseconds to 8 seconds
* Optional Hardware fuse Watchdog always on (WDTON) for Fail-Safe mode

### Interrupt Mode

In Interrupt mode, the WDT forces an interrupt when the timer expires. This interrupt can be used to wake the device from any of the sleep modes, and also as a general system timer. One example is to limit the maximum time allowed for certain operations, forcing an interrupt when the operation has run longer than expected. This is enabled by setting the Interrupt mode bit (WDIE) in the Watchdog Timer Control Register (WDTCSR).

### System Reset

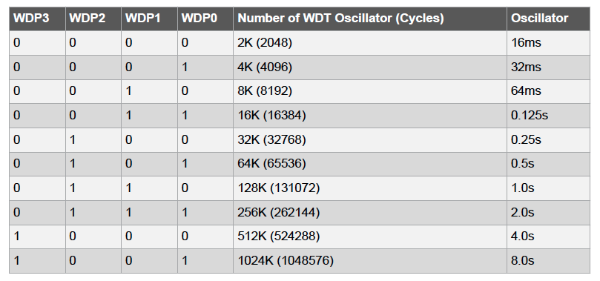
In System Reset mode, the WDT forces a reset when the timer expires. This is typically used to prevent system hang-ups in the case of runaway code. This is enabled by setting the System Reset mode bit (WDE) in the Watchdog Timer Control Register (WDTCSR).

### Interrupt and System Reset Mode

Interrupt and System Reset mode combines the other two modes by first forcing an interrupt and then switching to the System Reset mode. This mode will offer a safe shutdown by allowing time to save critical parameters before a system reset. This is enabled when both the WDTIE and WDTE are set.

### WDT Time-out Period

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



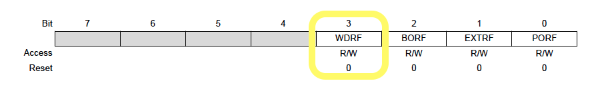
### WDTON

The WDTON fuse, if programmed, will force the WDT into System Reset mode. With the fuse programmed, the WDE bit and WDIE bit are locked to 1 and 0 respectively.

## **Watchdog System Reset Flag**

The Watchdog System Reset Flag (WDRF) bit in the MCU Status Register (MCUSR) is set if a Watchdog System Reset occurs. The WDRF bit is cleared by a Power-on Reset or by writing '0' to it. To identify a reset condition, the user should read and then reset the WDRF as early as possible in the program. If the register is cleared before another reset occurs, the source of the reset can be found by examining the Reset Flags.

### MCU Status Register



# AVR® External Interrupts

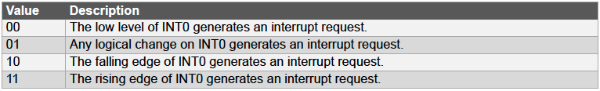
Last modified by Microchip on 2023/11/09 09:02

AVR® devices have external interrupts that can wake a device from sleep based on a rising or falling edge signal at an I/O pin or a change in digital voltage level at an I/O pin. The device can then process an application based on the interrupt source and then go back to sleep. The device has multiple interrupt pins for multiple interrupt sources. <https://youtu.be/121d4HVkADs>

## **External Interrupts**

External interrupts are triggered by the INT pin or any of the PCINT pins. If enabled, the interrupts trigger even if the INT or PCINT pins are configured as outputs. This feature provides a way of generating a software interrupt. The external interrupts can be triggered by a falling or rising edge or a low level. This is set up by the External Interrupt Control Register A EICRA. When the external interrupts are enabled and are configured as level-triggered, the interrupts trigger as long as the pin is held low.

The External Interrupt Control Register EICR controls how the external interrupts operate.



A low level interrupt on the INT pin is detected asynchronously. This implies that this interrupt can be used for waking the part also from sleep modes other than Idle mode. The I/O clock is halted in all sleep modes except Idle mode.

## **Pin Change Interrupt**

The Pin Change Interrupt Request triggers if any enabled PCINT pin changes state. There are multiple pin change interrupts, all tied to a set of pins or ports.

On an ATmega324PB device, for example, the Pin Change locations are:

* Pin Change Interrupt Request 4 (PCI4) triggers upon changes to pins PCINT[38:32]
* Pin Change Interrupt Request 3 (PCI3) triggers upon changes to pins PCINT[31:24]
* Pin Change Interrupt Request 2 (PCI2) triggers upon changes to pins PCINT[23:16]
* Pin Change Interrupt Request 1 (PCI1) triggers upon changes to pins PCINT[15:8]
* Pin Change Interrupt Request 0 (PCI0) triggers upon changes to pins PCINT[7:0]

The PCMSK4, PCMSK3, PCMSK2, PCMSK1and PCMSK0 registers control which pins contribute to the pin change interrupts. Pin change interrupts on PCINT are detected asynchronously. This implies that these interrupts can also be used for waking the part from sleep modes other than Idle mode.

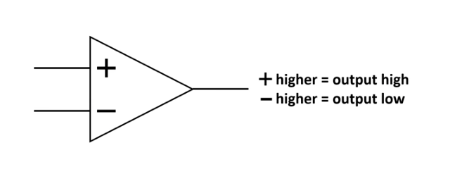
## **Additional Information**

* [Application Note AN\_8468 - Using External Interrupts for megaAVR® Devices](http://www.microchip.com/wwwAppNotes/AppNotes.aspx?appnote=en590935)

# AVR® Internal Comparator

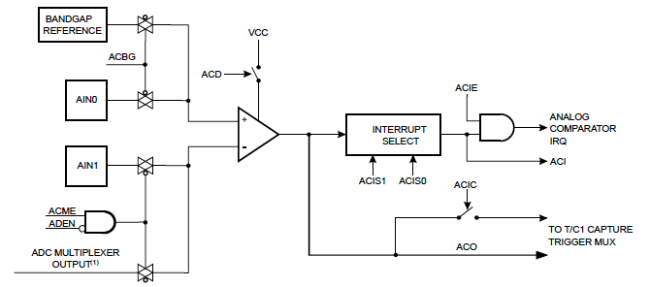
Last modified by Microchip on 2023/11/09 09:02

Many AVR® devices have an internal Analog Comparator (AC) peripheral that compares the input values on the positive pin AIN0 and negative pin AIN1. When the voltage on the positive pin AIN0 is higher than the voltage on the negative pin AIN1, the Analog Comparator Output, ACO, is set. <https://youtu.be/wB_4bGv9sYc>



## **Comparator Overview**

## **Comparator Setup**



## **Comparator Interrupts**

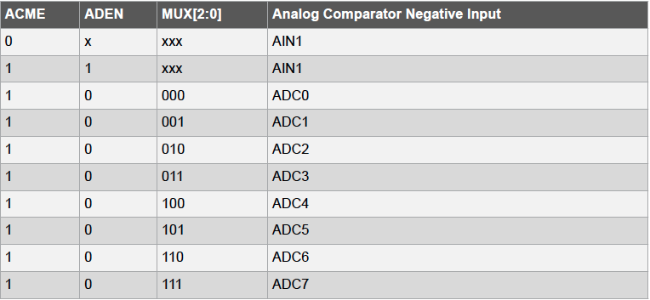
The comparator’s output can be set to trigger the Timer/Counter1 Input Capture function. In addition, the comparator can trigger a separate interrupt, exclusive to the analog comparator. The user can select interrupt triggering on comparator output rise, fall, or toggle.

## **Comparator Negative Input Pin Options**

It is possible to select any of the ADC[7:0] pins to replace the negative input to the analog comparator. The ADC multiplexer is used to select this input, and consequently, the ADC must be switched off to utilize this feature.

If the Analog Comparator Multiplexer Enable bit in the ADC Control and Status Register B ADCSRB.ACME is '1' and the ADC is switched off ADCSRA.ADEN=0, then the three least significant Analog Channel Selection bits in the ADC Multiplexer Selection register ADMUX.MUX[2:0] select the input pin to replace the negative input to the analog comparator.

When ADCSRB.ACME=0 or ADCSRA.ADEN=1, AIN1 is applied to the negative input of the analog comparator.



## **Additional Information**

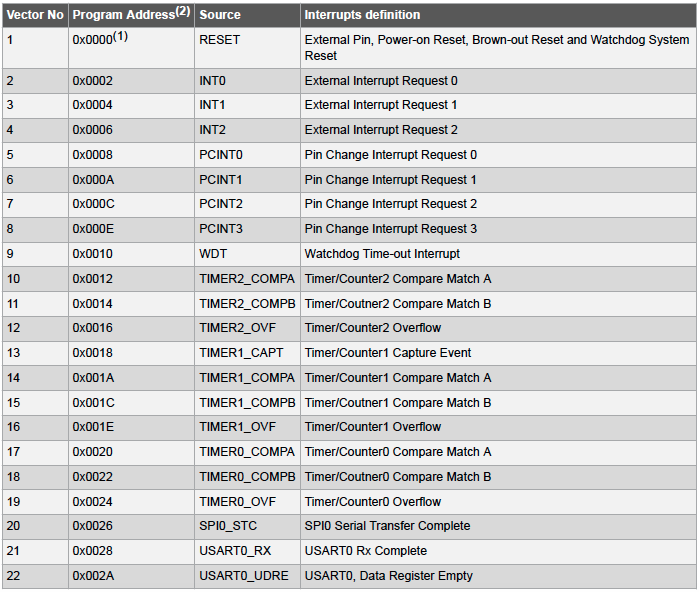
* [AN\_0934 - Setup and Use of the AVR Analog Comparator](http://www.microchip.com/wwwAppNotes/AppNotes.aspx?appnote=en591231)
* [Analog Comparator Voltage Reference Overview](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/peripherals/voltage-reference/analog-comparator-overview/)
* [Comparator Example Project](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/peripherals/voltage-reference/8-bit-avr-analog-comparator-voltage-reference/)

# AVR® Interrupts

Last modified by Microchip on 2023/11/09 09:02

AVR® devices provide several different interrupt sources including internal and [external interrupts](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/structure/extint/). Interrupts can stop the main program from executing to perform a separate interrupt service routine (ISR). When the ISR is completed, program control is returned to the main program at the instruction that was interrupted.

These interrupts each have a separate program vector in the program memory space. All interrupts are assigned individual enable bits which must be written in one logic together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt. The lowest addresses in the program memory space are by default defined as the Reset and Interrupt Vectors. They have determined priority levels; the lower the address the higher the priority level. RESET has the highest priority, and next is the External Interrupt Request 0 (INT0). <https://youtu.be/onfpksEIXzg>



Interrupt Vector Table for ATmega324PB

The Interrupt Vectors can be moved to the start of the Boot Flash section by setting the IVSEL bit in the MCU Control Register MCUCR. The Reset Vector can also be moved to the start of the Boot Flash section by programming the BOOTRST Fuse.

## **How it Works**

When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled. The interrupt vector directs program control to the proper ISR or execution. That ISR can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. When the ISR is completed and the return (RETI) command is executed from the ISR, the Global I-bit is automatically set to 'ON' and the program execution returns to the main program at the instruction that was interrupted.

## **Interrupt Response Time**

The interrupt execution response for all the enabled AVR interrupts is four clock cycles minimum. After four clock cycles, the program vector address for the actual interrupt handling routine is executed. During this four clock cycle period, the Program Counter is pushed onto the stack. The vector is normally a jump to the interrupt routine, and this jump takes three clock cycles. If an interrupt occurs during the execution of a multi-cycle instruction, this instruction is completed before the interrupt is served.

If an interrupt occurs when the MCU is in sleep mode, the interrupt execution response time is increased by four clock cycles. This increase comes in addition to the start-up time from the selected sleep mode. A return from an interrupt handling routine takes four clock cycles. During these four clock cycles, the Program Counter (two bytes) is popped back from the stack, the stack pointer is incremented by two, and the I-bit in SREG is set.

## **Additional Information**

* [Example Project File Using the Watchdog Timer as an Interrupt on ATmega324PB](https://microchiptechnology.sharepoint.com/:u:/s/DeveloperHelp/EXb8iTpCqgxHrXsg1eoY1-8Bbwvi6e1ngmuSeX21Xngt7g?e=J7hrYP)
* [AVR External Interrupts](https://microchiptechnology.sharepoint.com/:u:/s/DeveloperHelp/EXb8iTpCqgxHrXsg1eoY1-8Bbwvi6e1ngmuSeX21Xngt7g?e=J7hrYP)

# AVR® Low Power Sleep Modes

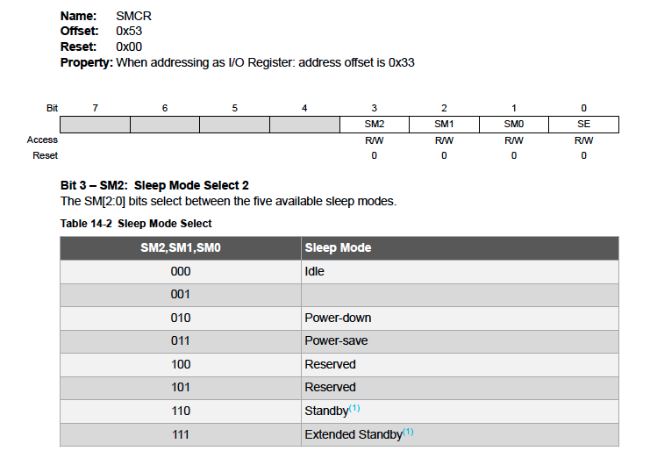
Last modified by Microchip on 2023/11/09 09:02

AVR® 8-bit microcontrollers include several sleep modes that enable the application to shut down unused modules in the MCU, thereby saving power. The AVR device provides various sleep modes allowing you to tailor the power consumption to the application requirements. <https://youtu.be/Fyspr40KcMU>

There are five sleep modes to select from:

* Idle Mode
* Power Down
* Power Save
* Standby
* Extended Standby

To enter any of the sleep modes, the Sleep Enable bit in the Sleep Mode Control Register (SMCR.SE) must be written to '1' and a SLEEP instruction must be executed. Sleep Mode Select bits (SMCR.SM[2:0]) select which sleep mode (Idle, Power-down, Power-save, Standby, or Extended Standby) will be activated by the SLEEP instruction. Sleep options shown are from ATmega328PB.



## **Interrupts**

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

## **Reset**

If a reset occurs during sleep mode, the MCU wakes up and executes from the Reset Vector.

## **Brief AVR Sleep Mode Summary**

Project Download: [Sample Project in Video](https://microchiptechnology.sharepoint.com/:u:/s/DeveloperHelp/EWtjQ0WKuI5LryCcPDJydSIBCed1BT1izkjcnnuw5EAxQQ?e=NTjnlx)

## **Idle Mode**

When the SM[2:0] bits are written to '000', the SLEEP instruction makes the MCU enter Idle mode, stopping the CPU but allowing the Serial Peripheral Interface (SPI), Universal Synchronous/Asynchronous Receiver/Transmitter (USART), Analog Comparator, 2-wire Serial Interface, Timer/Counters, Watchdog, and the interrupt system to continue operating. This sleep mode basically halts the CPU clock (clkCPU) and Flash clock (clkFLASH), while allowing the other clocks to run.

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

## **Power-Down Mode**

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

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

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

Note: If a level triggered interrupt is used for wake up from power down, the required level must be held long enough for the MCU to complete the wake-up to trigger the level interrupt. If the level disappears before the end of the Start-up Time, the MCU will still wake up, but no interrupt will be generated. The start-up time is defined by the Start Up Timer (SUT) and Clock Select (CKSEL) Fuses.

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

## **Power-save Mode**

When the SM[2:0] bits are written to 011, the SLEEP instruction makes the MCU enter Power-save mode. This mode is identical to Power-down, with one exception: If Timer/Counter2 is enabled, it will keep running during sleep.

The device can wake up from either Timer Overflow or Output Compare event from Timer/Counter2 if the corresponding Timer/Counter2 interrupt enable bits are set in Timer Interrupt Mask Register (TIMSK2), and the Global Interrupt Enable bit in Status Register (SREG) is set.

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

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

## **Standby Mode**

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

## **Extended Standby Mode**

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

## **Tips and Tricks to Minimize Power Consumption**

There are several possibilities to consider when trying to minimize power consumption in an AVR-controlled system. In general, sleep modes should be used as much as possible, and the sleep mode should be selected so that as few as possible of the device’s functions are operating. All functions not needed should be disabled.

In particular, the following modules may need special consideration when trying to achieve the lowest possible power consumption.

### ****Analog to Digital Converter****

If enabled, the ADC will be enabled in all sleep modes. To save power, the ADC should be disabled before entering any sleep mode. When the ADC is turned off and on again, the next conversion will be an extended conversion.

### ****Analog Comparator****

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

### ****Brown-Out Detector****

If the Brown-Out Detector (BOD) is not needed by the application, this module should be turned off. If the BOD is enabled by the BODLEVEL Fuses, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption.

### ****Internal Voltage Reference****

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

### ****Watchdog Timer****

If the Watchdog Timer is not needed in the application, the module should be turned off. If the Watchdog Timer is enabled, it will be enabled in all sleep modes and hence always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption.

### ****Port Pins****

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

For analog input pins, the digital input buffer should be disabled at all times. An analog signal level close to VCC/2 on an input pin can cause significant current even in active mode. Digital input buffers can be disabled by writing to the Digital Input Disable Registers (DIDR0 for ADC, DIDR1 for AC).

### ****On-chip Debug System****

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

# AVR® MCU Digital I/O Ports

Last modified by Microchip on 2023/11/09 09:02

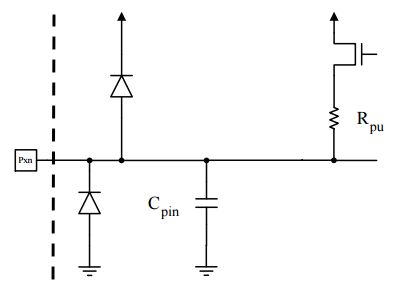
## **Digital Input/Output Ports on AVR**

Eight-bit AVR® microcontrollers (MCUs) control applications through their digital Input and Output (I/O) pins. These pins can monitor any voltage present as a high-impedance input and supply or sink current as a high or low-voltage digital output. These pins are usually organized in groups of eight and referred to as a port. The AVR uses the alphabet to name these ports, for example, PortA, PortB, etc. The pins of PortA are referred to as PA0 - PA7. <https://youtu.be/bDPdrWS-YUc>

## **Overview**

All AVR ports have true Read-Modify-Write functionality when used as general digital I/O ports. This means that the direction of one port pin can be changed without unintentionally changing the direction of any other. The same applies when changing the drive value (if configured as an output) or enabling/disabling pull-up resistors (if configured as an input). Each output buffer has symmetrical drive characteristics with both high sink and source capability.

The pin driver is robust enough to drive LED displays directly. All port pins have individually selectable pull-up resistors with a supply-voltage invariant resistance. All I/O pins have protection diodes to both VCC and Ground as indicated in the figure.



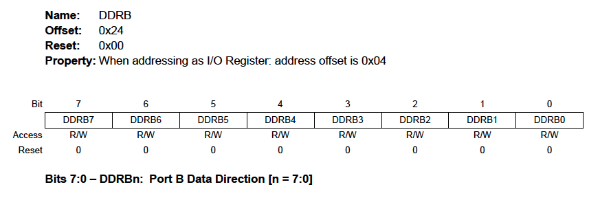
### Brief AVR I/O Port Summary

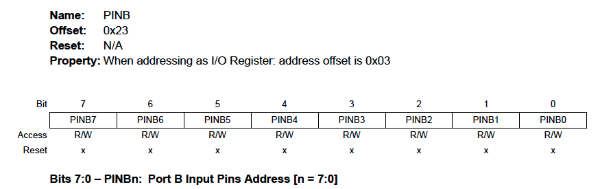
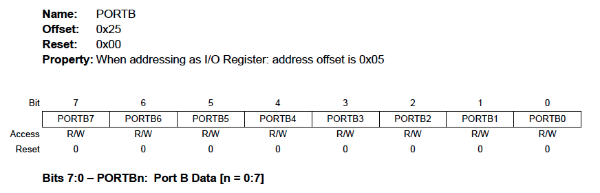
### Configuring the I/O Digital Pins

Each port consists of three registers:

* **DDRx** – Data Direction Register
* **PORTx** – Pin Output Register
* **PINx** – Pin Input Register

where x = Port Name (A, B, C or D)





These registers determine the setup of the digital inputs and outputs. I/O pins can also be shared with internal peripherals. For example, the Analog to Digital (ADC) converter can be connected to the I/O pin instead of being a digital pin. In this case, the I/O pin registers set it up as a tri-state high-impedance input.

### Register Bits

* DDxn bits are accessed at the DDRx I/O address
* PORTxn bits at the PORTx I/O address
* PINxn bits at the PINx I/O address

Where n = pin bit number in the Port Register

#### **DDxn**

The DDxn bits in the DDRx Register select the direction of this pin. If DDxn is written to '1', Pxn is configured as an output pin. If DDxn is written to '0', Pxn is configured as an input pin.

#### **PORTxn**

The PORTxn bits in the PORTx register have two functions. They can control the output state of a pin and the setup of an input pin.

As an Output:  
If a '1' is written to the bit when the pin is configured as an output pin, the port pin is driven high. If a ‘0’ is written to the bit when the pin is configured as an output pin, the port pin is driven low.

As an Input:  
If a '1' is written to the bit when the pin is configured as an input pin, the pull-up resistor is activated. If a ‘0’ is written to the bit when the pin is configured as an input pin, the port pin is tri-stated.

#### **PINxn**

The PINxn bits in the PINx register are used to read data from port pin. When the pin is configured as a digital input (in the DDRx register), and the pull-up is enabled (in the PORTx register) the bit will indicate the state of the signal at the pin (high or low).

Note: If a port is made an output, then reading the PINx register will give you data that has been written to the port pins.

As a Tri-State Input:  
When the PORTx register disables the pull-up resistor the input will be tri-stated, leaving the pin left floating. When left in this state, even a small static charge present on surrounding objects can change the logic state of the pin. If you try to read the corresponding bit in the pin register, its state cannot be predicted.

### Examples

All PORTA pins set as inputs with pull-ups enabled and then read data from PORTA:

DDRA = 0x00;        //make PORTA all inputs  
PORTA = 0xFF;        //enable all pull-ups  
data = PINA;        //read PORTA pins into variable data

PORTB set to tri-state inputs:

DDRB  = 0x00;        //make PORTB all inputs  
PORTB = 0x00;        //disable pull-ups and make all pins tri-state

PORTA lower nybble set as outputs, higher nybble as inputs with pull-ups enabled:

DDRA  = 0x0F;        //lower pins output, higher pins input  
PORTA = 0xF0;        //output pins set to 0, input pins enable pull-ups

### Example Project

[Video Referenced I/O Example](https://microchiptechnology.sharepoint.com/:u:/s/DeveloperHelp/EZ-ZCQJt6dNItJa2e5ZyoS0BPRC0Zgz3CqYeel_NLOIEpQ?e=ZOOuRD)

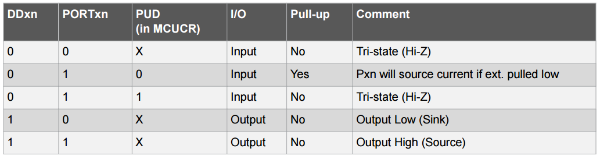
### Tips and Tricks

#### **Switching Between Input and Output**

When switching between tri-state ({DDxn, PORTxn} = 0b00) and output high ({DDxn, PORTxn} = 0b11), an intermediate state with either pull-up enabled {DDxn, PORTxn} = 0b01) or output low ({DDxn, PORTxn} = 0b10) must occur.

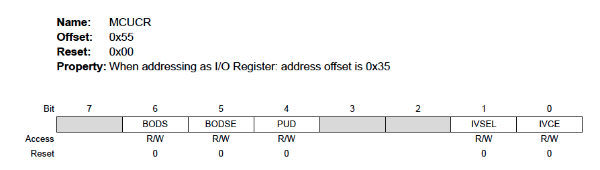
Normally, the pull-up enabled state is fully acceptable, as a high impedance environment will not notice the difference between a strong high driver and a pull-up. If this is not the case, the PUD bit in the MCUCR Register can be set to disable all pull-ups in all ports.

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



### Disable Pull-Ups Over-ride

The PUD Pull-up Disable bit in the MCUCR register can override the DDRx and PORTx pull-up settings.



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

### Toggling an I/O Pin

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

### Unconnected Pins

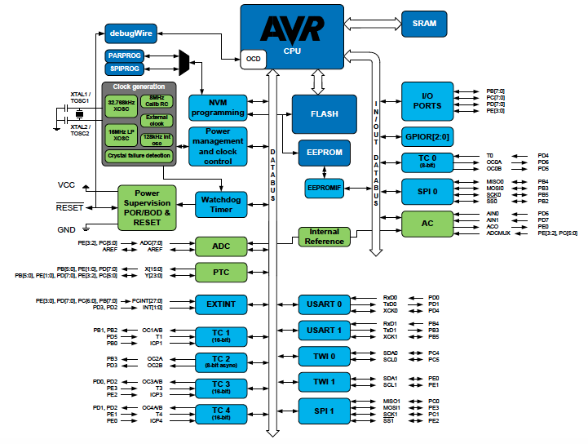
If some pins are unused, we recommend that you ensure that these pins have a defined level, even though most of the digital inputs are disabled in the deep sleep modes. Floating inputs should be avoided to reduce current consumption in all other modes where the digital inputs are enabled (Reset, Active mode and Idle mode).

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

# AVR® Memory

Last modified by Microchip on 2023/11/09 09:02

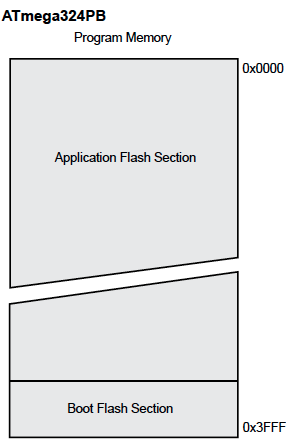
AVR® microcontrollers are built using a modified Harvard Architecture. This means the Flash Program Memory space is on a separate address bus than the Static Random Access Memory (SRAM). There are two data bus, one that can access all data and the In/Out data bus with limited access to a small section of memory. <https://youtu.be/ZeY6BKqIZGk>



## **Brief AVR Memory Summary**

## **Program Memory**

The AVR microcontrollers contain 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 32K x 16. For software security, the Flash Program memory space is divided into two sections - Boot Loader Section and Application Program Section in the device. The Flash memory has a typical endurance of at least 10,000 write/erase cycles. Constant tables can be allocated within the entire program memory address space, using the Load Program Memory (LPM) instruction. There is also a library of functions to make this easier [AVR Libc Library](https://onlinedocs.microchip.com/pr/GUID-317042D4-BCCE-4065-BB05-AC4312DBC2C4-en-US-2/index.html).



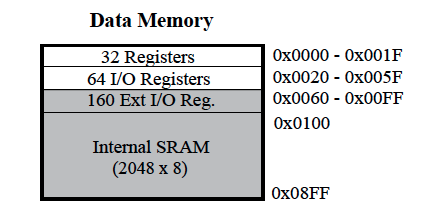
## **EEPROM Data Memory**

Data electrically erasable programmable read-only memory (EEPROM) is organized as a separate data space, in which single bytes can be read and written. Access from the CPU to EEPROM is done through the EEPROM Address Registers, the EEPROM Data Register, and the EEPROM Control Register. The EEPROM has an endurance of at least 100,000 write/erase cycles.

## **SRAM Data Memory**

Data can be accessed through the standard data bus. There is a secondary In/Out bus for rapid direct access to select locations.

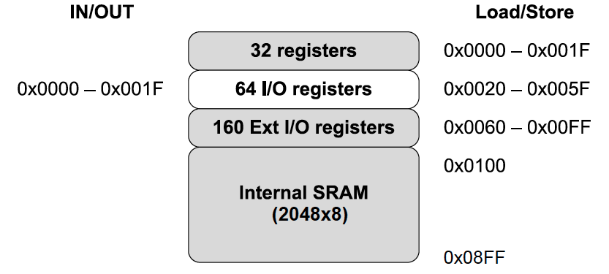
|  |  |
| --- | --- |
| Data memory consists of : | |
| Register space | Consists of 32 general-purpose working 8-bit registers (R0-R31). |
| I/O Memory | Contains addressable space for peripheral functions, such as control registers and other I/O functions. |
| Extended I/O Memory (Device dependent) | Some AVR microcontrollers with more peripherals need more space than the I/O memory can address so some of the SRAM is used as Extended I/O memory to handle the extra peripherals control registers and other I/O functions. |
| Internal SRAM (Data Memory) | Is used for temporarily storing intermediate results and variables within a software application. |



|  |  |
| --- | --- |
| There are five different data bus (not In/Out) addressing modes for the data memory: | |
| Direct | The direct addressing reaches the entire data space. |
| Indirect | In the Register File, registers R26 to R31 feature the indirect addressing pointer registers. |
| Indirect with Displacement | The Indirect with Displacement mode reaches 63 address locations from the base address given by the Y- or Z-register. |
| Indirect with Pre-decrement | The address registers X, Y, and Z are decremented. |
| Indirect with Post-increment | The address registers X, Y, and Z are incremented. |

In/Out Data bus

This data bus has direct access to the 64-byte I/O Memory section (not Extended) using a 0x00 to 0x1F address. This memory can also be accessed by the standard data bus using a 0x20 address offset in the access command.



## **I/O Memory**

All I/O locations (I/O Memory and Extended I/O Memory) can be accessed by the LD/LDS/LDD and ST/STS/STD assembly instructions using the standard data bus. Data is transferred between the 32 general-purpose working registers and the I/O space.

I/O Registers within the In/Out data bus address range 0x00-0x1F (I/O Memory) 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

## **General Purpose I/O Registers**

Three General Purpose I/O Registers, General Purpose I/O Register 0/1/2 (GPIOR 0/1/2) are at the top of the I/O memory (0x020- 0x022). These registers can be used for storing any information, and they are particularly useful for storing global variables and Status Flags. These registers are directly bit-accessible using the SBI, CBI, SBIS, and SBIC instructions through the In/Out data bus. The remaining I/O registers begin after the General Purpose I/O Registers.

## **Accessing 16-bit Registers**

The AVR data bus is 8 bits wide 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 high byte of the 16-bit register must be written before the low byte. The high byte is then written into the temporary register. When the low byte of the 16-bit register is written, the temporary register is copied into the high 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.

# AVR® Peripheral Touch Controller (PTC)

Last modified by Microchip on 2023/11/09 09:02

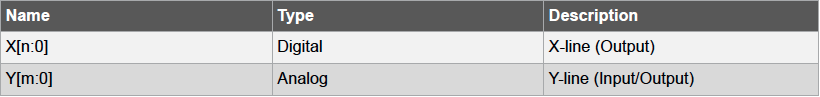
## **Capacitive Touch on AVR® Using the PTC**

The Peripheral Touch Controller (PTC), in some AVR® devices, is used for capacitive touch applications. The PTC acquires signals in order to detect a touch on capacitive sensors. The external capacitive touch sensor is typically formed on a Printed Circuit Board (PCB), and the sensor electrodes are connected to the analog front end of the PTC through the I/O pins in the device. The PTC supports both self- and mutual-capacitance sensors.

<https://youtu.be/ZLl7yDypew8>

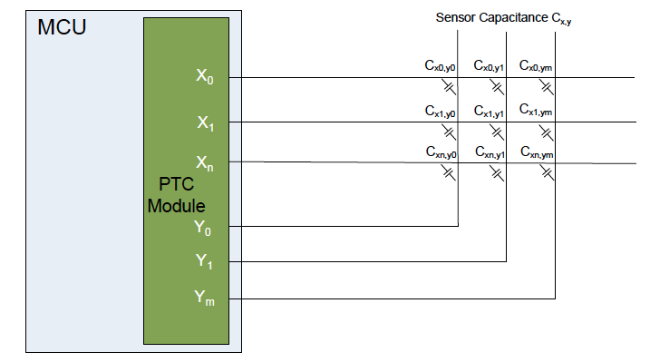
## **I/O Connections**

The I/O lines used for analog X-lines and Y-lines must be connected to external capacitive touch sensor electrodes. External components are not required for normal operation. However, to improve the Electromagnetic Compatibility (EMC) performance, a series resistor of 1k Ω or more can be used on X-lines and Y-lines.



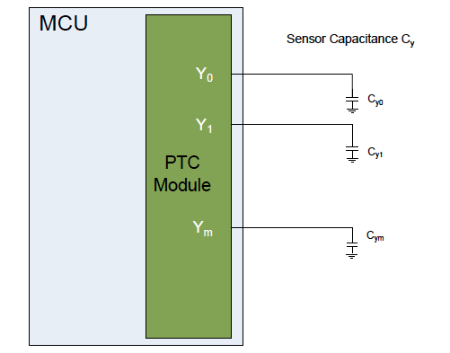
## **Mutual-Capacitance Mode**

In the mutual-capacitance mode, sensing is done using capacitive touch matrices in various X-Y configurations, including Indium Tin Oxide (ITO) sensor grids. The PTC requires one pin per X-line and one pin per Y-line. A mutual-capacitance sensor is formed between the two I/O lines - an X electrode for transmitting and Y electrode for sensing. The mutual capacitance between the X and Y electrode is measured by the PTC.



## **Self-Capacitance Mode**

In the self-capacitance mode, the PTC requires only one pin (Y-line) for each touch sensor. A self-capacitance sensor is connected to a single pin on the PTC through the Y electrode for sensing the signal. The sense electrode capacitance is measured by the PTC.



## **Additional Information**

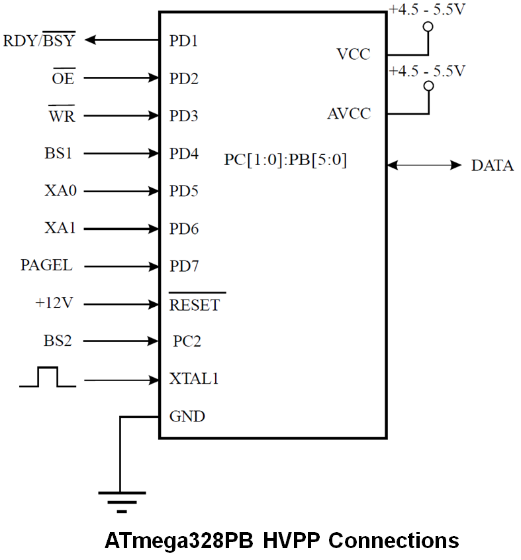
* [PTC Robustness Design Guide](http://ww1.microchip.com/downloads/en/AppNotes/atmel-42360-ptc-robustness-design-guide_applicationnote_at09363.pdf)

# AVR® Programming Interfaces

Last modified by Microchip on 2023/11/09 09:02

## **High Voltage Parallel Programming (HVPP)**

It’s first important to understand that all (see the "[Exceptions](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/structure/programming-interfaces/#exceptions)" section) AVR®-based Tiny and Mega devices include a High Voltage Parallel Programming (HVPP) or a High Voltage Serial Programming (HVSP) programming interface. Both require the application of a “high voltage” (12V) to the Reset pin and both require access to a large number of pins. The HVPP interface requires access to at least 16 pins while the HVSP interface requires access to at least eight pins. For those reasons, these interfaces are primarily used for production programming of the devices. The following schematic depicts the required HVPP connections for the [ATmega328PB](https://www.microchip.com/en-us/product/ATmega328PB) (see section 33.7 in the [datasheet](https://www.microchip.com/en-us/product/ATmega328PB#document-table)):



The good news…

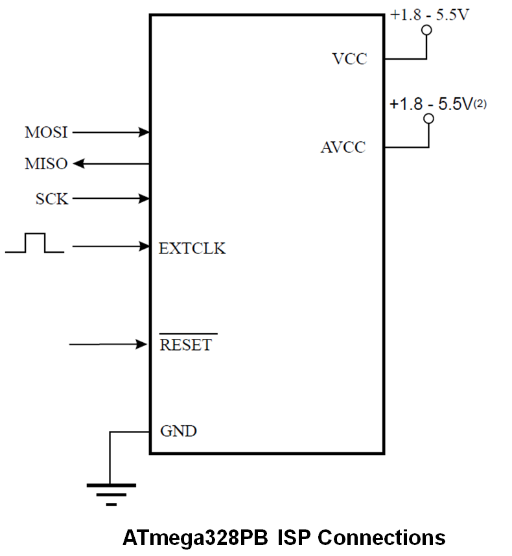
* The HVPP or HVSP interfaces are always enabled because they cannot be inadvertently disabled by a fuse setting or user action.

The bad news…

* They are almost never a realistic programming option once a Tiny or Mega is soldered onto a custom board because they simply require too many pins.

## **ISP/JTAG**

In addition to the HVPP or HVSP interfaces, all (see the "[Exceptions](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/structure/programming-interfaces/#exceptions)" section) Tiny and Mega devices also include one or two “standard” programming interfaces, ISP or JTAG.  
  
In-Circuit Serial Programming (ISP) allows the program memory to be reprogrammed In-System through a Serial Peripheral Interface (SPI). The following schematic depicts the required ISP connections for the [ATmega328PB](https://www.microchip.com/en-us/product/ATmega328PB) (see section 33.9 in the [datasheet](https://www.microchip.com/en-us/product/ATmega328PB#document-table)).



​

Depending on CKSEL Fuses, a valid clock must be present for ISP to function.

ISP is covered in detail in app-note [AVR910 - In-System Programming](http://ww1.microchip.com/downloads/en/AppNotes/Atmel-0943-In-System-Programming_ApplicationNote_AVR910.pdf).

The good news…

* These standard interfaces only require three or four pins.

The bad news…

* They can be easily disabled by the wrong fuse settings.

## **Combined Interfaces**

All (see the "[Exceptions](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/structure/programming-interfaces/#exceptions)" section) Tiny and Mega devices include one of the two following combinations of programming interfaces:

* HVPP (or HVSP)and ISP
* HVPP (or HVSP)and ISP and JTAG

The ISP and JTAG interfaces are the standard programming interfaces for the Tiny and Mega devices. It’s recommended to include a programming header for one of the two interfaces on any custom board to allow convenient reprogramming of the device if necessary.

If a device only has an ISP interface and it is disabled via a fuse setting the only recovery is through its HVPP or HVSP interface (which is most likely not physically possible).  
  
If a device has both an ISP and JTAG interface and one of those two is disabled by fuse settings, the other interface can be used to access the part if the pins required are accessible. The ISP interface requires three pins and the JTAG interface requires four pins so it’s more likely either of those interfaces would be easier to access than the HVSP or HVPP interfaces.

## **Exceptions**

* The Tiny4/5/9/10/20/40 devices do not have an HVPP or HVSP interface. They only have a Tiny Programming Interface (TPI). Newer devices like the Tiny417/817/1617 devices only have a Unified Program and Debug Interface (UPDI). As long as you have access to those interfaces there is no worry about “bricking” these devices.
* The XMEGA® devices include only a Programming and Debug Interface (PDI) or both a PDI and JTAG interface. You can’t disable the PDI interface so as long as you have access to the one PDI pin that can’t be used for any other function and the Reset pin you can’t “brick” these devices.

# AVR® Serial Peripheral Interface (SPI)

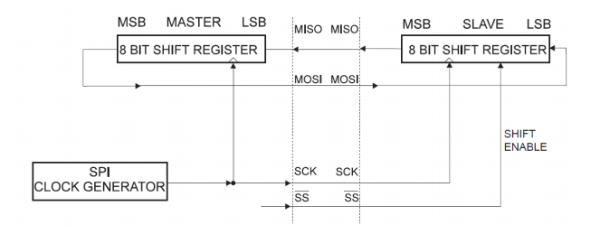
Last modified by Microchip on 2023/11/09 09:02

Serial Peripheral Interface (SPI) protocol on AVR**®** devices will enable your AVR microcontroller to communicate with multiple other devices at the same time. Use the SPI bus to communicate between a master device and one or multiple slave devices. SPI uses Master In Slave Out (MISO) and Master Out Slave In (MOSI) lines to communicate between devices, the Serial Clock (SCK) to maintain a consistent clock between devices, and Slave Select (SS) line to pick which peripheral device is communicating with the master device.

<https://youtu.be/9DLxM_GwT0A>

## **SPI System**

The system consists of two shift registers and a master clock generator. The SPI master initiates the communication cycle when pulling low the SS pin of the desired slave. Master and slave prepare the data to be sent in their respective shift registers, and the master generates the required clock pulses on the SCK line to interchange data. Data is always shifted from master to slave on the MOSI line, and from slave to master on the MISO line. After each data packet, the master will synchronize the slave by pulling the SS line high.



## **Master Mode**

When configured as a master, the SPI interface has no automatic control of the SS line. This must be handled by user software before communication can start. When this is done, writing a byte to the SPI Data Register starts the SPI clock generator, and the hardware shifts the eight bits into the slave. After shifting one byte, the SPI clock generator stops, setting the end of the Transmission Flag (SPIF). If the SPI Interrupt Enable (SPIE) bit in the SPCR Register is set, an interrupt is requested. The master may continue to shift the next byte by writing it into SPDR or signal the end of the packet by pulling high the Slave Select, SS line. The last incoming byte will be kept in the Buffer Register for later use.

## **Slave Mode**

When configured as a slave, the SPI interface will remain sleeping with MISO tri-stated as long as the SS pin is driven high. In this state, the software may update the contents of the SPI Data Register, SPDR, but the data will not be shifted out by incoming clock pulses on the SCK pin until the SS pin is driven low. As one byte has been completely shifted, the end of the Transmission Flag (SPIF) is set. If the SPI Interrupt enable bit (SPIE) in the SPCR Register is set, an interrupt is requested.

## **Additional Information**

* [Example Project using the ATmega324PB](https://microchiptechnology.sharepoint.com/:u:/s/DeveloperHelp/EegIg4fOnnFHuro_d3sdmEUBQfpyFy2PT9Q88RyWC6EhTA?e=Xyb7jL)
* "[*AVR319: Using the USI Module for SPI Communication on tinyAVR® and megaAVR® Devices*](http://ww1.microchip.com/downloads/en/AppNotes/Atmel-2582-Using-the-USI-Module-for-SPI-Communication-on-tinyAVR-and-megaAVR-Devices_ApplicationNote_AVR319.pdf)"

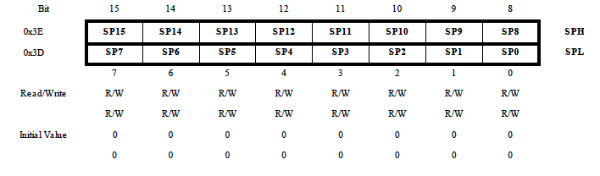
# AVR® Stack Register

Last modified by Microchip on 2023/11/09 09:02

## **8-bit AVR® Stack**

The stack is mainly used for storing temporary data, local variables, and return addresses after interrupts and subroutine calls. It is implemented as growing from higher to lower memory locations. The Stack Pointer register always points to the top of the stack; it points to the data SRAM stack area where the subroutine and interrupt stacks are located.

### Stack Pointer

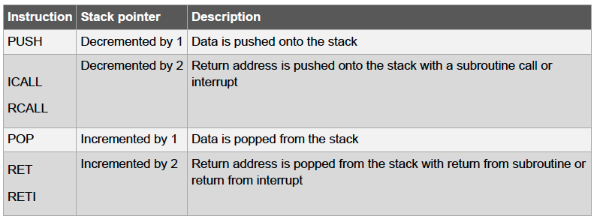


The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used depends on the implementation.

​

Data space in some implementations of the AVR architecture is so small that only the Stack Pointer Low (SPL) register is needed. In this case, the Stack Pointer High (SPH) register will not be present.

### Stack Instruction Set



A stack PUSH command will decrease the Stack Pointer. The stack in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. The initial Stack Pointer value equals the last address of the internal SRAM and the Stack Pointer must be set to point above the start of the SRAM.

# AVR® Status Register

Last modified by Microchip on 2023/11/09 09:02

The Status register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. The Status register is updated after all the Arithmetic Logical Unit (ALU) operations. This will, in many cases, remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

​

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

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

## **STATUS: Status Register**



|  |  |
| --- | --- |
| bit 7 | **I: Global Interrupt Enable:** The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable register is cleared, none of the interrupts are enabled independently of the individual interrupt enable settings. The I-bit (bit 7) is cleared by hardware after an interrupt has occurred and is set by the Return from Interrupt (RETI) instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the Set Global Interrupt Flag (SEI) and Clear Global Interrupt Flag (CLI) instructions, as described in the instruction set reference. |
| bit 6 | **T: Copy Storage:** The Bit Copy instructions Bit Load (BLD) and Bit Store (BST) use the T-bit as a source or destination for the operated bit. A bit from a register in the register file can be copied into T by the BST instruction and a bit in T can be copied into a bit in a register in the register file by the BLD instruction. |
| bit 5 | **H: Half Carry Flag:** The Half Carry Flag, H, indicates a half carry in some arithmetic operations. It is useful in Binary Code Decimal (BCD) arithmetic. |
| bit 4 | **S: Sign Flag, S = N xor V:** The S-bit is always an exclusive or between the Negative Flag and the Two’s Complement Overflow Flag. |
| bit 3 | **V: Two's Compliment 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. |

# Optimizing C Code on 8-Bit AVR®

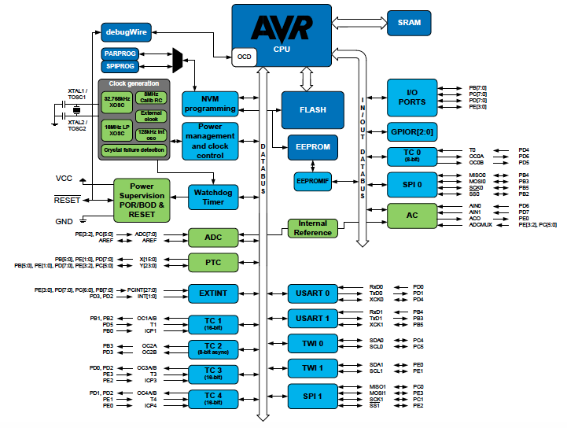
Last modified by Microchip on 2023/11/09 09:02

Before optimizing embedded systems software, it is necessary to have a good understanding of how the AVR**®** microcontroller (MCU) core is structured. <https://youtu.be/jzXy9B44FJ4>

## **Optimization Overview**

## **Atmel AVR® 8-bit Architecture**

AVR uses the Harvard architecture – with separate memories and buses for programs and data. It has a fast-access register file of 32 x 8 general purpose working registers with a single clock cycle access time. The 32 working registers are one of the keys to efficient C coding. These registers have the same function as the traditional accumulator, except that there are 32 of them. The AVR arithmetic and logical instructions work on these registers, hence they take up less instruction space. In one clock cycle, AVR can feed two arbitrary registers from the register file to the ALU, perform an operation, and write back the result to the register file.



Instructions in the program memory are executed with a single-level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16 or 32-bit instruction.

The C High-level Language (HLL) has become increasingly popular for programming microcontrollers. The advantages of using C compared to Assembler are numerous: Reduced development time, easier maintainability and portability, and easier-to-reuse. The penalty can be larger code size and as a result of that often reduced speed. To reduce these penalties the AVR architecture is tuned to efficiently decode and execute instructions that are typically generated by C compilers.

## **Additional Information**

* [App Note AVR035 - Efficient C Coding for AVR](http://ww1.microchip.com/downloads/en/AppNotes/doc1497.pdf)
* [App Note AVR4027 - Tips and Tricks to Optimize Your C Code for 8-bit AVR Microcontrollers](http://ww1.microchip.com/downloads/en/AppNotes/doc8453.pdf)

# Read While Write Flash Memory on AVR®

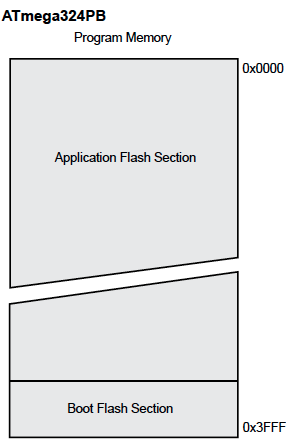
Last modified by Microchip on 2023/11/09 09:02

In many AVR® devices, the Boot Loader section of program memory supports real Read-While-Write Self-Programming. This feature allows flexible application software updates controlled by the MCU using a Flash-resident Boot Loader program. The Boot Loader program can use any available data interface and associated protocol to read code and write (program) that code into the Flash memory or read the code from the program memory. The program code within the Boot Loader section has the capability of writing into the entire Flash, including the Boot Loader memory. The Boot Loader can thus even modify itself, and it can also erase itself from the code if the feature is not needed anymore.

<https://youtu.be/YKFWUX9w4eY>

## **Flash Memory**

The Flash memory is organized into two main sections: the Application section and the Boot Loader section. The size of the different sections is configured by the BOOTSZ Fuses. These two sections can have different levels of protection since they have different sets of Lock bits.



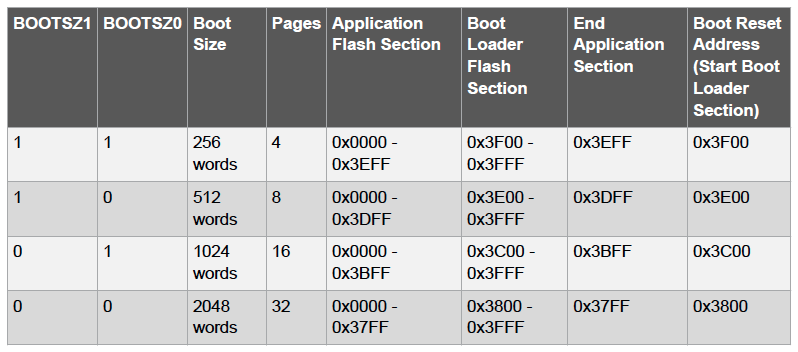
## **Brief AVR Read While Write Memory Summary**

## **Application Section**

The Application section is the section of the Flash that is used for storing the application code. The protection level for the Application section can be selected by the application Boot Lock bits (Boot Lock bits 0). The Application section can never store any Boot Loader code since the Store Program Memory (SPM) instruction is disabled when executed from the Application section.

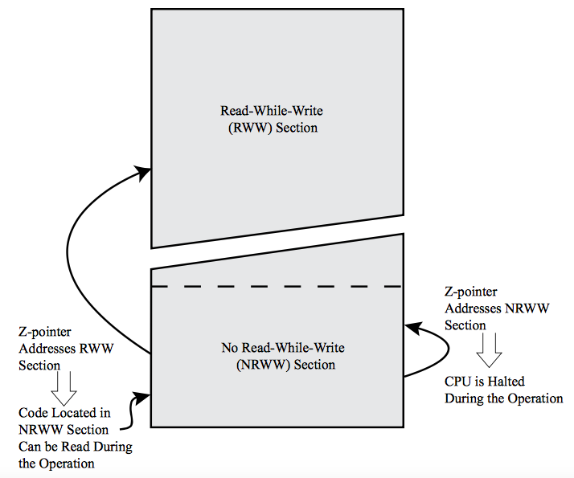
## **Boot Loader Section (BLS)**

While the Application section is used for storing the application code, the Boot Loader software must be located in the BLS since the SPM instruction can initiate a programming when executing from the BLS only. The SPM instruction can access the entire Flash, including the BLS itself. The protection level for the Boot Loader section can be selected by the Boot Loader Lock bits. The example shown is from the AVRmega328PB datasheet.



## **Read-While-Write (RWW) and No Read-While-Write (NRWW) Flash Sections**

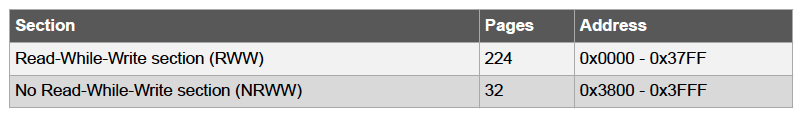
In addition to the two sections that are configurable by the BOOTSZ Fuses as described above, the Flash is also divided into two fixed sections, the Read-While-Write (RWW) section and the No Read-While-Write (NRWW) section.



The main difference between the two sections is:

* When erasing or writing a page located inside the RWW section, the NRWW section can be read during the operation.
* When erasing or writing a page located inside the NRWW section, the CPU is halted during the entire operation.

The limit between the RWW and NRWW sections is defined in the device datasheet. These are the settings for the AVRmega328PB.



Whether the AVR device supports RWW or if the AVR device is halted during a Boot Loader software update is dependent on which address is being programmed.

The user software can never read any code located inside the RWW section during a Boot Loader software operation. The syntax “Read-While-Write section” refers to which section is being programmed (erased or written), not which section is being read during a Boot Loader software update.