
AVR1313: Using the XMEGA IO Pins and External Interrupts

Features

- Flexible pin configuration
- Synchronous and/or asynchronous input sensing
- Asynchronous wake-up signaling
- Configurable output driver – and pull settings:
 - Totem-pole
 - Wired-AND
 - Wired-OR
 - Pull-up/-down
 - Bus keeper
 - Inverted I/O
- Slew rate control
- Configuration of multiple pins in a single operation
- Virtual set/clear/toggle registers for output and direction registers
- Mapping of ports into virtual ports in I/O space for faster access

1 Introduction

This application note gives an introduction to the usage of the highly configurable XMEGA™ I/O pins and external interrupts.



8-bit **AVR**[®]
Microcontrollers

Application Note

Rev. 8050A-AVR-02/08





2 Module Overview

This chapter gives an overview of the I/O port module and describes the various configuration options available.

2.1 Naming

I/O pins on the XMEGA are grouped into I/O ports of 8 pins. The I/O ports are named PORT x , where x is a letter, e.g. PORTA, PORTB etc. The individual pins within a port are named P xn , where x is the port letter and n identifies the pin number, e.g. PA0, PA1 etc.

Some registers affect all pins in a port. In these registers, bit n corresponds to pin n . In other words, the least significant bit corresponds to pin 0, while the most significant bit corresponds to pin 7.

2.2 Basic Configuration and Usage

This section describes the configuration options and registers needed to use the basic functionality of the I/O port pins.

2.2.1 Setting the Direction of Port Pins

The direction of port pins is controlled through the DIR register. A port pin n is configured as output if the DIR n bit in the DIR register is '1'. If the DIR n bit is '0', the port pin is configured as an input.

The DIR register can be accessed directly, or manipulated through the strobe registers DIRSET, DIRCLR and DIRTGL. Writing a bit mask to DIRSET will cause that bit mask to be set in the DIR register. Writing a bit mask to DIRCLR will cause that bit mask to be cleared in the DIR register. Writing a bit mask to DIRTGL will cause the bits in the mask to be toggled in the DIR register.

2.2.2 Controlling the Output Value of Port Pins

The output value of a port pin is controlled through the OUT register. The direction of the port pin must be set to output for the corresponding OUT register bit to have any effect. Port pin n will be driven high when the OUT n register is '1'. When OUT n is '0', port pin n will be driven low. This assumes that the output configuration has not been changed and that the pin inversion bit has not been set.

The OUT register can either be accessed directly, or manipulated through the strobe registers OUTSET, OUTCLR and OUTTGL. Writing a bit mask to OUTSET will cause that bit mask to be set in the OUT register. Writing a bit mask to OUTCLR will cause that bit mask to be cleared in the OUT register. Writing a bit mask to OUTTGL will cause the bits in the mask to be toggled in the OUT register.

2.2.3 Reading the Logic State of Port Pins

The logic state of a port can be obtained by reading the IN registers. The current logic state of port pin n can be read through the IN n bit, Regardless of direction setting.

2.3 Pin Configuration

Each IO pin has its own configuration register named PIN n CTRL, where n signifies the pin number. Through these registers a number of parameters can be configured on a pin-by-pin basis. These parameters are explained in this section.

2.3.1 Output/Pull Configuration

The output/pull configuration bits are used to configure the output driver behavior and the pull configuration. Table 2-1 shows a parametric overview of the possible configurations.

Table 2-1. Output/pull configurations

Symbol	Output Configuration	Pull Configuration
PORT_OPC_TOTEM_gc	Totempole	(N/A)
PORT_OPC_BUSKEEPER_gc	Totempole	Bus keeper on input and output
PORT_OPC_PULLDOWN_gc	Totempole	Pull-down on input
PORT_OPC_PULLUP_gc	Totempole	Pull-up on input
PORT_OPC_WIREDOR_gc	Wired-OR	(N/A)
PORT_OPC_WIREDAND_gc	Wired-AND	(N/A)
PORT_OPC_WIREDORPULL_gc	Wired-OR	Pull-down
PORT_OPC_WIREDANDPULL_gc	Wired-AND	Pull-up

Output configurations:

- In the “Totempole” configuration, the output is driven hard to either VCC or GND as commanded by the corresponding bit in the OUT register.
- In the “Wired-OR” configuration, writing a ‘1’ to OUT n causes pin n to be driven hard to VCC. Writing ‘0’ to OUT n releases the pin, allowing the pin to be pulled to GND with an internal or external pull-down resistor.
- In the “Wired-AND” configuration, writing a ‘0’ to OUT n causes pin n to be driven hard to GND. Writing ‘1’ to OUT n releases the pin, allowing the pin to be pulled to VCC with an internal or external pull-up resistor.

Pull configurations:

- The “Buskeeper on input and output” configuration provides a weak bus keeper that will keep the pin at the same logic level when the pin is not driven to any logic state.
- The “Pull-down” configuration enables the internal pull-down resistor for the pin.
- The “Pull-up” configuration enables the internal pull-up resistor for the pin.

2.3.2 Input/Sense Configuration

The input/sense configuration bits controls the input sensing and digital input buffer of the I/O pin. The possible input/sense configurations are listed in Table 2-2.

Four sense configurations are available. These affect how interrupts and events are generated from the pin. The pin can sense on both edges, rising edge, falling edge or low level. Note that sensing on high level can be achieved by setting the “Inverted I/O” bit in the PIN n CTRL register. If the pin is used to generate events, and the sense





setting is set to low level, the pin is transparent to the event system, meaning that the level on the pin will be directly connected to the event line.

Setting the input/sense field to “Digital input buffer disabled” disables the digital input buffer on the pin. This can be used to reduce power consumption when the pin is unused, or used only for analog functions. The corresponding bit in the IN register will always read ‘0’ when the digital input buffer is disabled.

Table 2-2. Input/sense configuration

Symbol	Configuration
PORT_ISC_BOTHEDGES_gc	Sense both edges
PORT_ISC_RISING_gc	Sense rising edge
PORT_ISC_FALLING_gc	Sense falling edge
PORT_ISC_LEVEL_gc	Sense low level (transparent for events)
PORT_ISC_INPUT_DISABLE_gc	Digital input buffer disabled

2.3.3 Inversion

The “Inverted I/O” bit in the PIN n CTRL registers control the polarity of the pin. If this bit is written to zero, all input and output logic is inverted with respect to the descriptions in this application note.

Setting the “Inverted I/O” bit for a pin will invert the input/output signal for any peripheral module overriding the pin. As an example, it is possible to invert a PWM output signal from a timer/counter module simply by setting the “Inverted I/O” bit. This can be useful for easy switching between active high and active low driving.

2.3.4 Slew-Rate Control

Writing a ‘1’ to the “Slew-rate control” bit in the PIN n CTRL register enables slew-rate limiting for that I/O pin. This can be used to reduce EMC issues caused by switching of logic levels on the port pins. For information about the characteristics of the slew-rate limiter, please refer to the device data sheet.

2.4 Configuring and Using Port Interrupts

I/O port interrupts can be used to generate interrupt on pin change or pin level, and for waking the device from sleep modes. This section gives an overview of the I/O port interrupt system and how it is used.

2.4.1 Configuration of Port Interrupts

Each I/O port on the XMEGA has two interrupts. It is possible to map each interrupt to be triggered by an arbitrary combination of the pins in the I/O port.

Setting up the pin interrupts is done in 3 steps, in this example interrupt 0:

1. Configure the input/sense part of the PIN n CTRL register for each pin that can trigger the interrupt.
2. Write the bit mask corresponding to the pins that can trigger the interrupt to the INTOMASK register.
3. Select the interrupt priority level by setting the INT0LVL part of the INTLVL register.

Note that the selected interrupt level must be enabled in the Programmable Multi-level Interrupt Controller (PMIC) and the global interrupt enable flag must be set for

the interrupt handler to be executed. See application note AVR1305 for more information on interrupts on the XMEGA.

2.4.2 Asynchronous Sense

There are two levels of asynchronous support on the XMEGA. Pin 2 on every port has full asynchronous support, while the other pins have limited asynchronous support. A summary of the full and limited asynchronous sense modes is listed in Table 2-3 and Table 2-4.

Table 2-3. Full asynchronous sense support. (Pin 2)

Sense setting	Supported	Interrupt After Wake-up
Rising edge	Yes	Always
Falling edge	Yes	Always
Both edges	Yes	Always
Low level	Yes	Pin level must be kept unchanged

Table 2-4. Limited asynchronous sense support. (All port pins except pin 2)

Sense setting	Supported	Interrupt After Wake-up
Rising edge	No	-
Falling edge	No	-
Both edges	Yes	Pin value must be kept unchanged
Low level	Yes	Pin level must be kept unchanged

2.4.3 Using Port Interrupts to Wake up From Sleep Modes

Any port pin can be used to wake up the XMEGA from sleep modes. However, the asynchronous sense support level on the pin used for wake-up determines which sense settings can be used and the state of interrupt-flags after wake-up.

Table 2-3 and Table 2-4 shows the supported asynchronous sense settings for limited and full asynchronous sense pins. The pin used for wake-up must be configured to sense on one of the settings that are supported by the asynchronous sense to be able to wake up the device.

Table 2-3 and Table 2-4 also shows the conditions that must be met for the interrupt flag to be set after a device wake-up.

2.5 Using I/O Ports Efficiently

Two features are available on the XMEGA series that can reduce code size and increase execution speed: multi-pin configuration and virtual ports.

2.5.1 Multi-pin Configuration

Having one configuration register for each port pin increases the flexibility, but configuring every pin, one at a time can require a lot of code. Several pins in one IO port might need the same configuration. The process of configuring several pins in one IO port to the same configuration is simplified through a multi-pin configuration process. First, a bit pattern matching the pins to be configured is written to the PINMASK register in the PORTCFG module. When a pin configuration is written to





one of the PINnCTRL registers of that port, that value is written to all the PINnCTRL registers of the pins matching the bit pattern in the PINMASK register. It is not necessary to write to one of the registers that are targeted by the PINMASK register. If the register that is written to is not targeted by the bit mask in PINMASK, it remains unchanged.

Note that it is important that the multi-pin configuration is not interrupted by a task that writes to a PINnCTRL register. If the PINMASK register is already written, an interrupt is executed, and the interrupt service routine (ISR) writes to any PINnCTRL register, pins belonging to a different I/O port than intended will be configured. The recommended solution is to store and then disable the global interrupt flag before doing a multi-pin configuration and restore the global interrupt flag after the configuration has been written.

See chapter 3 for an example on how to use the multi-pin configuration.

2.5.2 Virtual Ports

Some instructions in the AVR® instruction set can only operate on addresses that are within the AVR I/O space. Using these instructions instead of their data space equivalents is both faster and consumes less program memory. All I/O port registers on the XMEGA have addresses outside the I/O space.

The solution to this is to use the virtual ports. Up to four of the I/O ports can be mapped into virtual ports that have registers in the I/O space. The virtual ports make the DIR, OUT, IN and INTFLAGS registers of the desired I/O port available in I/O space. The other, less used I/O port registers are still available through the regular port module registers.

2.5.3 Differences Between I/O Space and Data Space Instructions

The execution time and code size of the special I/O space instructions along with related data space instructions are shown in Table 2-5.

Table 2-5. Execution Time and Code Size for I/O Space and Data Space Instructions

Instruction	Clock cycles	Size (words)	Comment
IN	1	1	Only I/O space
OUT	1	1	Only I/O space
CBI	1	1	Only I/O space address < 32
SBI	1	1	Only I/O space address < 32
SBIC	2/3/4	1	Only I/O space address < 32.
SBIS	2/3/4	1	Only I/O space address < 32
LD / LD+	1 (IO) / 2 (RAM)	1	Pointer must be initialized
LD - / LDD	2 (IO) / 3 (RAM)	1	Pointer must be initialized
LDS	2 (IO) / 3 (RAM)	2	
ST / ST +	1	1	Pointer must be initialized
ST - / STD	2	1	Pointer must be initialized
STS	2	2	

Note that data instructions operating in data space can be used on registers with I/O space addresses, since the I/O space is also mapped into the data space.

The differences in execution time and code size might not seem that large, but in a real-life example the differences can be huge. To illustrate, consider setting the PC0 pin high without changing the state of the other pins. The following two examples show the assembly code needed to perform this task when PORTC is mapped to PVIRT0 compared to accessing PORTC directly in data space.

Using PVIRT0 (I/O space):

```
sbi PVIRT0_OUT, 0
```

The size of this code is 1 word and execution time is 1 clock cycle.

Using PORTC directly (Data space):

```
sbr r16, 0x01  
sts PORTC_base + PORT_OUTSET_offset, r16
```

The size of this code is 3 words and execution time is 3 clock cycles.

The above example shows that it is possible to save a significant amount of clock cycles and program memory by mapping a port to a virtual port. It is recommended to use virtual port mapping when timing requirements are tight and when the port registers are accessed frequently in the application.

Note that even though the CPU completes the “sbi” instruction in one clock cycle, it takes two clock cycles before the effect can be seen on the I/O port.

3 Getting Started

This section walks you through the basic steps for getting up and running with the XMEGA I/O pins and port interrupts.

3.1 Basic Digital I/O

Task: Set up PORTC to read input from 8 switches and output the pin state to 8 LEDs connected to PORTD.

1. Configure all 8 pins on PORTD to output by setting the PORTD.DIR register to 0xFF.
2. Read state of PORTC from the PORTC.IN register.
3. Store the value from step 1 to PORTD.OUT.
4. Repeat from step 1.

3.2 Configuring Several Pins in one Operation

Task: Set up pins 0-3 on PORTC for Wired-AND with pull-up operation.

1. Write 0x0f to PORTCFG.PINMASK to select that pins 0-3 are going to be configured.
2. Write PORT_OPC_WIREDANDPULL_gc to PORTC.PIN0CFG to trigger a write to the PINnCFG registers for pin0-3 on PORTC.

3.3 Mapping Real Ports to Virtual Ports

Task: Map PORTC to Virtual port 0 and PORTD to Virtual port 1 and perform the same task as in section 3.1 using the virtual ports.



1. Write `PORTCFG_VIRTMAP0_PORTC_gc` to `PORTCFG.VMAP0` to map PORTC to PVIRT0.
2. Write `PORTCFG_VIRTMAP0_PORTD_gc` to `PORTCFG.VMAP1` to map PORTD to PVIRT1.
3. Configure all 8 pins on PORTC to output by setting the `PVIRT0.DIR` register to `0xff`.
4. Read state of PORTC from the `PVIRT0.IN` register.
5. Store the value from step 4 to `PVIRT1.OUT`.
6. Repeat from step 4.

3.4 Configuring an I/O Pin for Interrupt Generation

Task: Set up PORTC interrupt 0 as a medium level interrupt, triggered by the rising edge of PC0. Use the interrupt service routine to toggle the output on PORTD.

1. Configure input/sense on pin0 to rising edge.
2. Write `0x01` to `PORTC.INT0MASK` to select PC0 as source for interrupt 0.
3. Set `IN0LVL` part of `PORTC.INTCTRL` to `PORT_INT0LVL_MED_gc` to enable interrupt 0 at medium level.
4. Enable medium level interrupts in the PMIC.
5. Enable the global interrupt flag.

4 Driver Implementation

The included driver has functions that control all the major features of the I/O port modules. Most functions take a pointer to an I/O port module as its first argument, so the same functions can be reused for all port modules on one XMEGA.

The driver is written in ANSI® C, and should compile on all compilers with XMEGA support. Note that this driver is **not** written with high performance in mind. It is designed as a library to get started with the XMEGA I/O ports and an easy-to-use framework for rapid prototyping. For time and code space critical application development, consider replacing function calls with macros or direct access to registers.

4.1 Files

The driver package consists of the following files:

- `port_driver.c` – I/O port driver source file.
- `port_driver.h` – I/O port driver header file.
- `port_example.c` – Examples using the I/O port driver.

4.2 Doxygen Documentation

All source code is prepared for automatic documentation generation using Doxygen. Doxygen is a tool for generating documentation from source code by analyzing the source code and using special keywords. For more details about Doxygen please visit <http://www.doxygen.org>. Precompiled Doxygen documentation is also supplied with the source code accompanying this application note, available from the `readme.html` file in the source code folder.



Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© 2008 Atmel Corporation. All rights reserved. Atmel®, logo and combinations thereof, AVR® and others, are the registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.