TB3229

Getting Started with General Purpose Input/Output (GPIO)

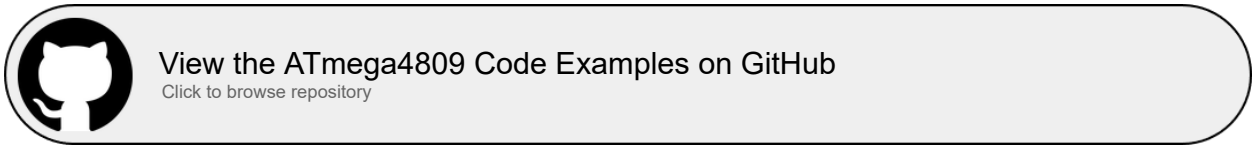
Introduction

Authors: Alexandru Niculae, Catalin Visan, Microchip Technology Inc.

The purpose of this document is to describe step-by-step how to use the microcontroller's pins as General Purpose Input/Output (GPIO) on Microchip tinyAVR® 0- and 1-series, megaAVR® 0-series and AVR® DA devices. The following use cases are presented:

- **Blink an LED:**
  Blink an LED using a delay, toggling the LED every 500 ms.
- **Long and Short Button Press:**
  Use a pin as input to distinguish between a long and short button press, defined by a delay threshold. An LED will toggle every 100 ms if a long press is detected, and every 500 ms if a short press is detected.
- **Wake-Up On Button Press:**
  Exit sleep on button press, turn on an LED, and go back to sleep. On button release - exit sleep, turn off the LED, and go back to sleep.

Note: For each use case described in this document, there are two code examples: One bare metal developed on ATmega4809, and one generated with MPLAB® Code Configurator (MCC) developed on AVR128DA48.



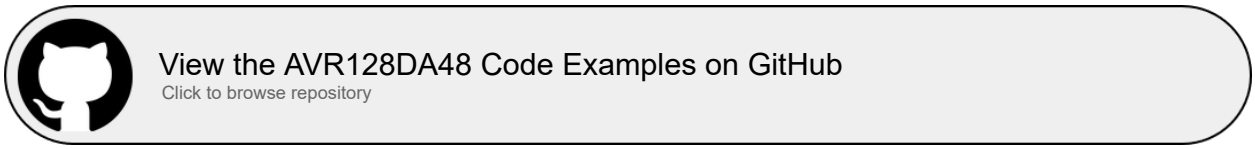


Table of Contents

1. Relevant Devices

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

- Vertical migration upwards is possible without code modification, as these devices are pin-compatible and provide the same or more features. Downward migration on tinyAVR® 1-series devices may require code modification due to fewer available instances of some peripherals
- Horizontal migration to the left reduces the pin count and, therefore, the available features
- Devices with different Flash memory sizes typically also have different SRAM and EEPROM

Figure 1-1. tinyAVR® 0-series Overview

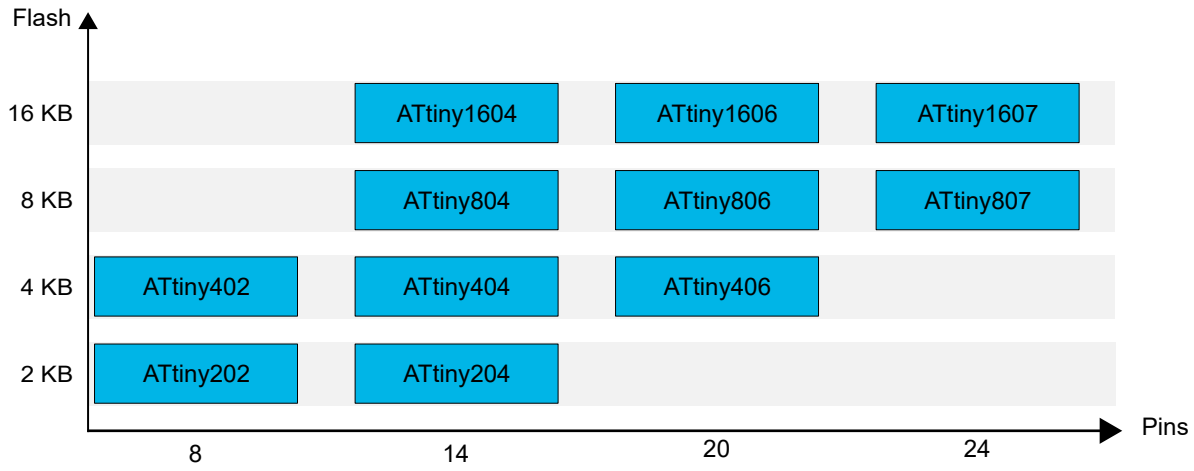


Figure 1-2. tinyAVR® 1-series Overview

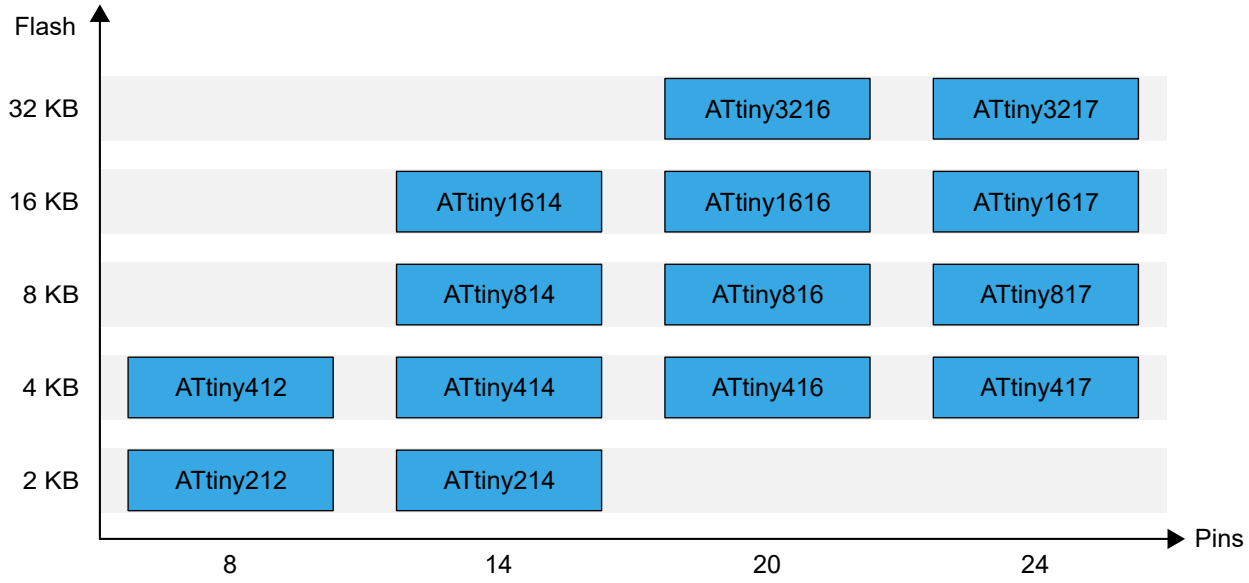


Figure 1-3. megaAVR® 0-series Overview

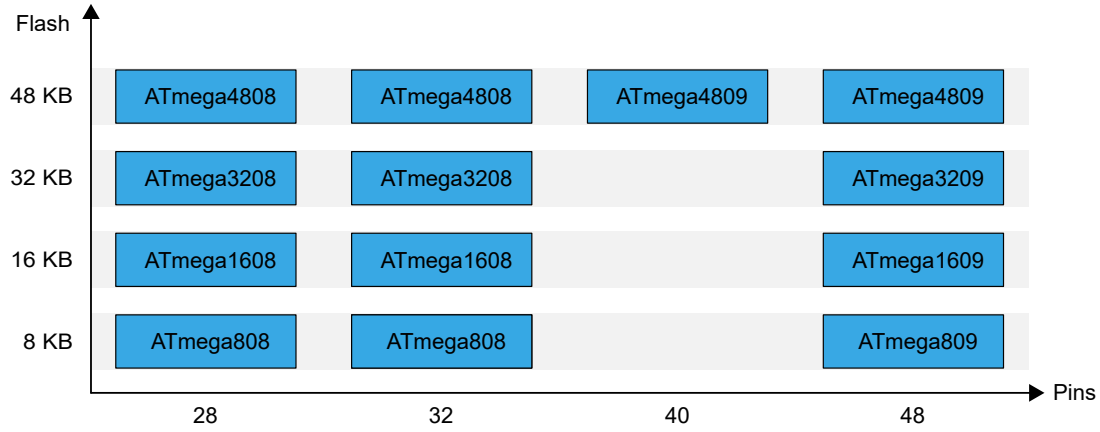
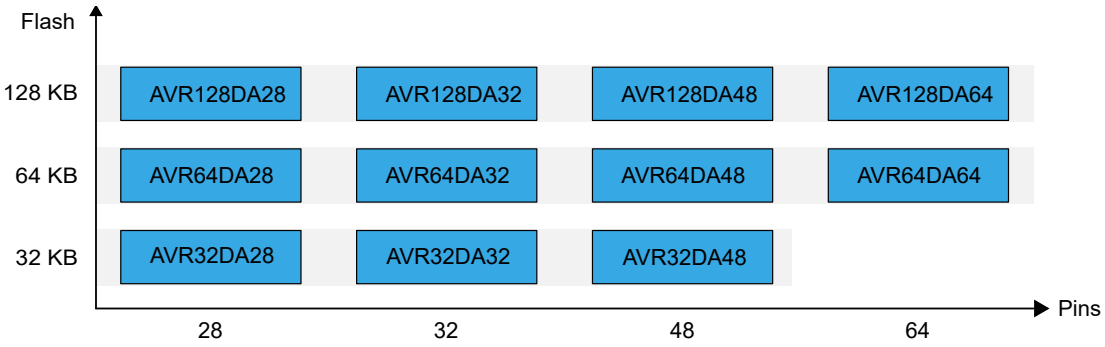


Figure 1-4. AVR® DA Family Overview

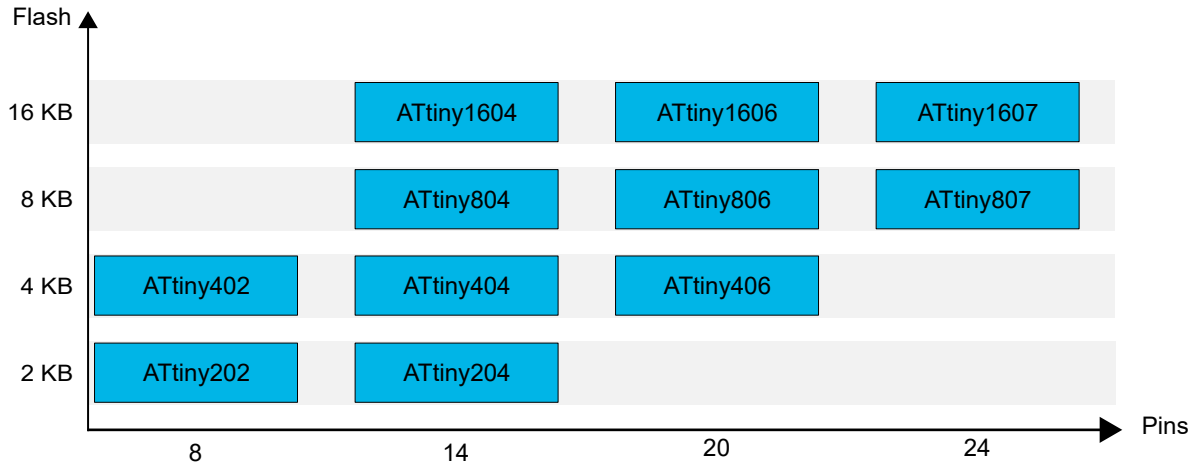


1.1 tinyAVR® 0-series

The figure below shows the tinyAVR® 0-series devices, laying out pin count variants and memory sizes:

- Vertical migration upwards is possible without code modification, as these devices are pin-compatible and provide the same or more features
- Horizontal migration to the left reduces the pin count and, therefore, the available features

Figure 1-5. tinyAVR® 0-series Overview



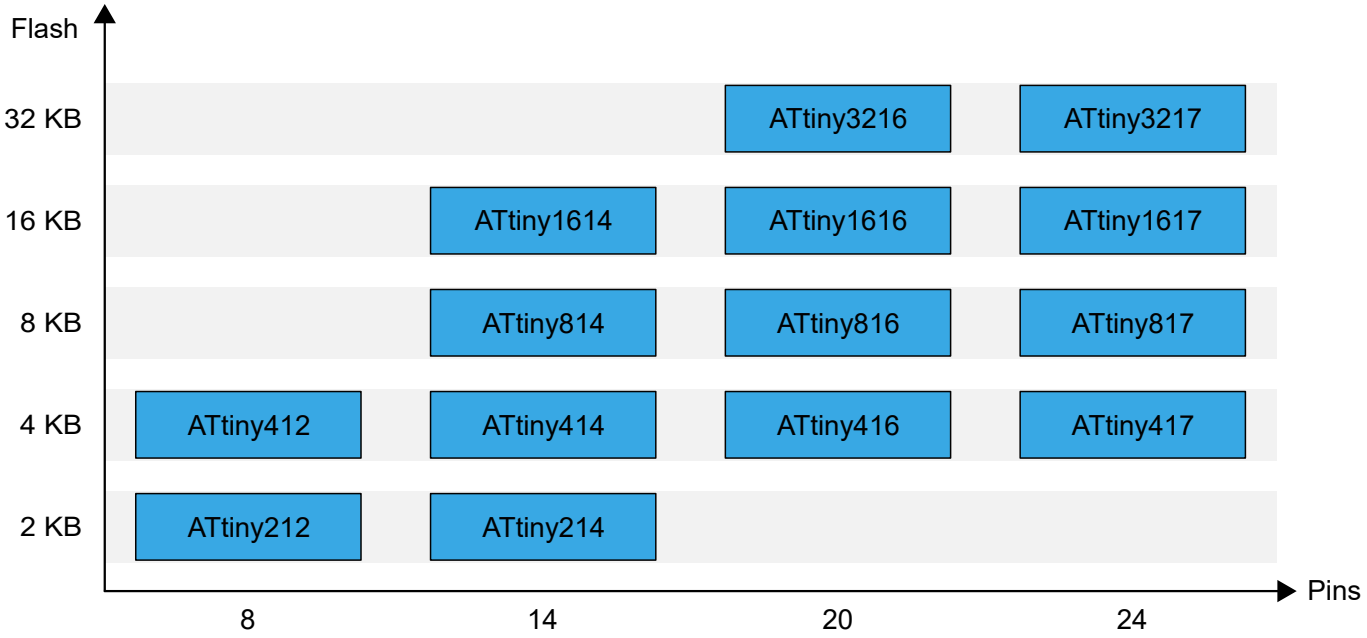
Devices with different Flash memory sizes typically also have different SRAM and EEPROM.

1.2 tinyAVR® 1-series

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

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

Figure 1-6. tinyAVR® 1-series Overview



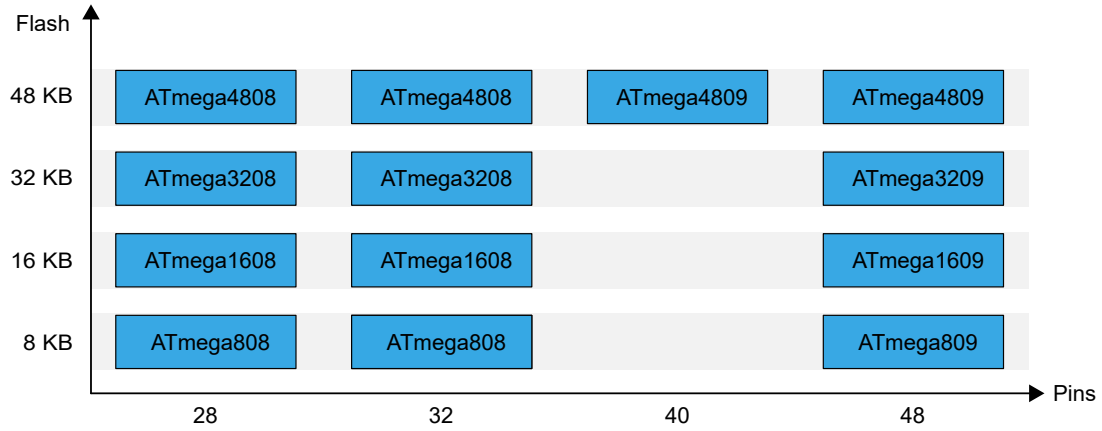
Devices with different Flash memory sizes typically also have different SRAM and EEPROM.

1.3 megaAVR® 0-series

The figure below shows the megaAVR® 0-series devices, laying out pin count variants and memory sizes:

- Vertical migration is possible without code modification, as these devices are fully pin and feature compatible
- Horizontal migration to the left reduces the pin count and, therefore, the available features

Figure 1-7. megaAVR® 0-series Overview



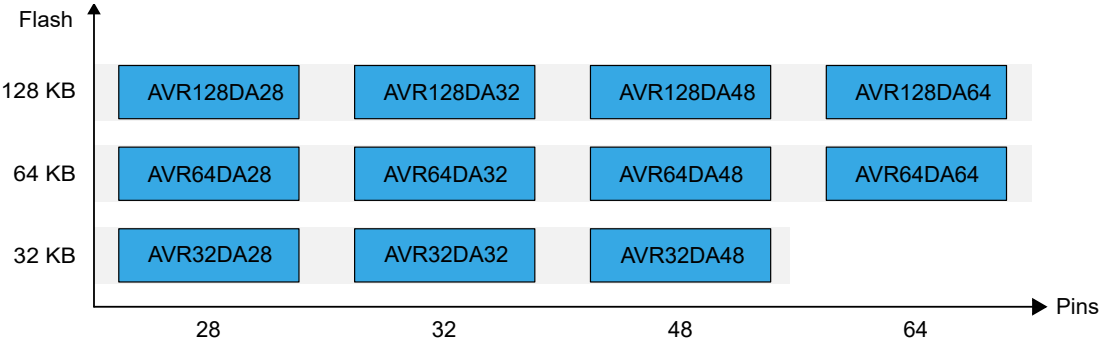
Devices with different Flash memory sizes typically also have different SRAM and EEPROM.

1.4 AVR® DA Family Overview

The figure below shows the AVR® DA devices, laying out pin count variants and memory sizes:

- Vertical migration is possible without code modification, as these devices are fully pin and feature compatible
- Horizontal migration to the left reduces the pin count, and therefore, the available features

Figure 1-8. AVR® DA Family Overview



Devices with different Flash memory sizes typically also have different SRAM.

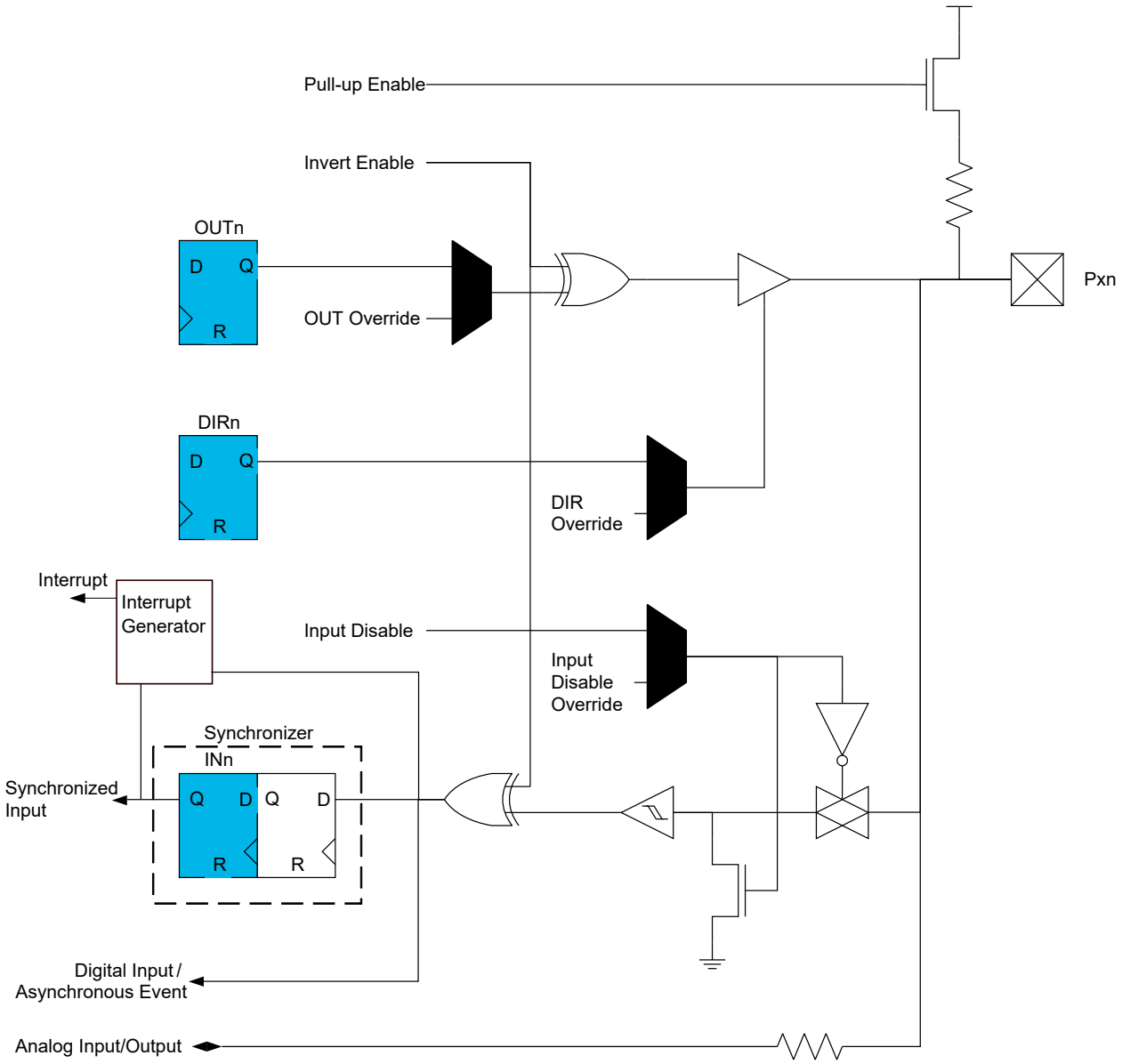
2. Overview

The instances of the PORT peripheral registers control the I/O pins of the device. Each port instance has up to eight I/O pins. The ports are named PORTA, PORTB, PORTC, etc. All pin functions are configurable individually per pin.

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

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

Figure 2-1. Port Block Diagram



3. Blink an LED

This section demonstrates how to turn an LED on and off alternatively.

Although this use case is very simple, it is widely used in real life applications, and it helps in understanding how to achieve some of the very basic functions of the microcontroller. These functions are:

- Setting a pin direction
- Setting the output value of a pin

Setting a Pin Direction

The directions of the pins are stored in the PORTn.DIR register. For example, the directions for the pins of PORTA are stored in PORTA.DIR. Each bit in the register controls the direction of the corresponding pin, so PORTA.DIR bit 0 controls pin A0 (also called PA0). For a bit value of '1', the pin is configured as output, while for '0', it is configured as input.

In most cases, it is only necessary to write one bit at a time. For this purpose, there are eight bitmasks (PINx_bm) defined in the <avr/io.h> header file, one for each bit. These macros represent bitmasks that have a '1' bit on x position and '0' bits in rest. For example, PIN5_bm is 0b00100000. Using these bitmasks in conjunction with logic operations makes it possible to only write the corresponding pin and make sure the others are unchanged.

Figure 3-1. Bit 5 Data Direction Register

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

To set (write to '1') pin x, use:

```
PORTn.DIR = PORTn.DIR | PINx_bm;
```

This because a logic OR operation with '1' will always result in '1', while logic OR with '0' will leave the value unchanged (1|1 = 1, 1|0 = 1, 0|0 = 0). Hence, the bitmask only has '1' on the position to be set.

To clear (write to '0') pin x, use:

```
PORTn.DIR = PORTn.DIR & ~PINx_bm;
```

This because a logic AND operation with '0' will always result in a '0', while logic AND operation with '1' will leave the value unchanged (1&1 = 1, 1&0 = 0, 0&0 = 0). Hence, the bitmask only has '0' on the position to clear.

Notes:

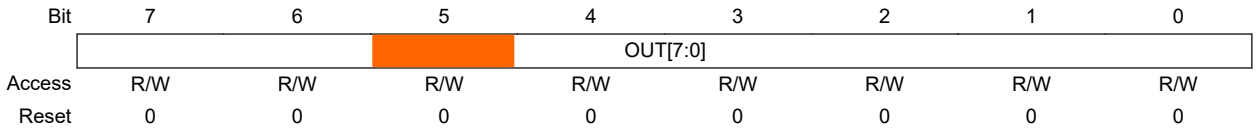
1. The same approach, to only modify a single bit, can be used for all other registers as described in previous paragraphs.
2. Logic operations with more bitmasks can be chained in an expression such as PORTA.DIR = PORTA.DIR | PIN0_bm | PIN1_bm.
3. PINx_bp are similar macros that define the position of the bit in the register. For example, PIN2_bp has the value 2. The bitmask macros can be obtained using these macros and the shift operation.

Setting the Output Value of a Pin

The PORTn.OUT register controls the output values, so that '0' means the pin will be pulled to GND and '1' means it will be pulled to V_{DD}.

Note: The meaning of these values, '0' and '1', can be inverted using the PORTn.PINxCTRL register. See the PORTn.PINxCTRL register section in the data sheet to understand its functionality.

Figure 3-2. Bit 5 Output Value Register



As previously described, the same approach to only modify a single bit can be applied to write to the PORTn.OUT register.

The following code toggles pin PB5 each 500 ms. An LED is attached to that pin on the ATmega 4809 Xplained Pro board.

```
#define F_CPU 3333333
#include <avr/io.h>
#include <avr/delay.h>

int main(void)
{

    PORTB.DIR |= PIN5_bm;

    while (1)
    {
        PORTB.OUT |= PIN5_bm;
        _delay_ms(500);
        PORTB.OUT &= ~PIN5_bm;
        _delay_ms(500);
    }
}
```

Notes:

1. To use the delay functions, F_CPU must be defined before including `avr/delay.h`. F_CPU must match the CPU frequency.
2. The PORTn.OUTTGL register can also be used to toggle a pin.
3. As an optimization, PORTn.OUTSET/PORTn.OUTCLR and PORTn.DIRSET/PORTn.DIRCLR registers can also be used to set the pin direction and output value. Their benefit is that instead of the read-modify-write operation, only a write operation is used.

View the ATmega4809 Code Example on GitHub
Click to browse repository

An MCC generated code example for AVR128DA48 with the same functionality as the one described in this section can be found here:

View the AVR128DA48 Code Example on GitHub
Click to browse repository

4. Long and Short Button Press

This section explains how to detect whether a button press is long or short, resulting in two different scenarios.

The GPIO interface can be used to sense external digital signals in order to make certain decisions. In this regard, the application range is vast, so the AVR controllers can be configured several of ways to match as many use cases as the user might need. In this section, the focus will be on the push of a button.

First, PB2 will be configured as input by writing a '1' on bit 2 of the Data Direction Clear (PORTB.DIRCLR) register.

Figure 4-1. Bit 2 Data Direction Clear Register

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

```
PORTB.DIRCLR = PIN2_bm;
```

For a button connected between the GPIO pin and GND to work correctly, connect a pull-up resistor between the GPIO pin and V_{DD}. This configuration will make sure the pin reads a default value of logic '1' when the button is not pressed. It is possible to activate an internal pull-up resistor from the Pin 2 Control (PORTB.PIN2CTRL) register.

Figure 4-2. Internal Pull-Up Resistor Pin 2 Control Register

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

```
PORTB.PIN2CTRL = PORT_PULLUPEN_bm;
```

For a better user experience, an LED-connected PB5 will blink with different frequency based on the type of press. Thus, PB5 will be configured as output by writing a '1' on bit 5 of the Data Direction Set (PORTB.DIRSET) register.

```
PORTB.DIRSET = PIN5_bm;
```

Then, in each iteration of the main loop, bit 2 of the Input Value (PORTB.IN) register will be checked. If the value is logic '0', the program will wait until the value is back to logic '1'. By doing this, the program will sense when a press-and-release action on the button was performed. While waiting, a counter will be incremented every few milliseconds.

If the value of the counter passes a certain threshold, the program will decide there was a long press. If the button is released before passing the threshold, the program will decide there was a short press. Depending on the press type, the LED will blink with different frequencies.

Figure 4-3. Bit 2 Input Value Register

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

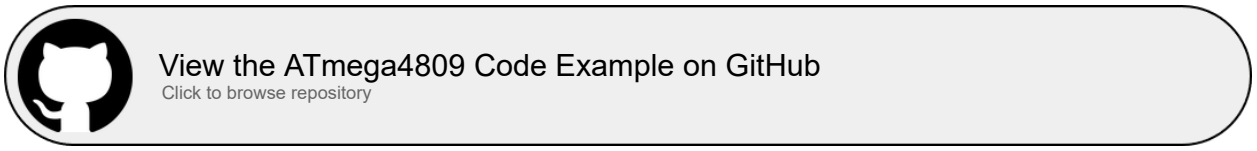
```
if(~PORTB.IN & PIN2_bm) /* check if PB2 is pulled to GND */
    {
        while(~PORTB.IN & PIN2_bm) /* wait until PB2 is pulled to VDD */
        {
            _delay_ms(STEP_DELAY);
            counter++;
            if(counter >= THRESHOLD)
            {
```

```
                LED_blink(LONG_DELAY);
                while (~PORTB.IN & PIN2_bm) /* wait until PB2 is pulled to VDD */
                }
                    ;
                }
                break;
            }
        }
        if(counter < THRESHOLD)
        {
            LED_blink(SHORT_DELAY);
        }
        counter = 0;
}
```

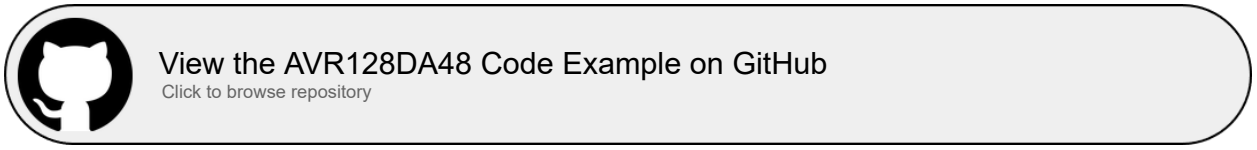
Note: The delay macro can be found in the `<util/delay.h>` header file.

The LED_blink function takes the blinking period in milliseconds divided by two as a parameter. The function must be declared “inline” because the `_delay_ms` macro must take a parameter that is already known at the compilation stage.

```
inline void LED_blink(uint32_t time_ms)
{
        for(uint8_t i = 0; i < NUMBER_OF_BLINKS ; i++)
        {
            PORTB.OUT |= PIN5_bm;
            _delay_ms(time_ms);
            PORTB.OUT &= ~PIN5_bm;
            _delay_ms(time_ms);
        }
}
```



An MCC generated code example for AVR128DA48 with the same functionality as the one described in this section can be found here:



5. Wake-Up On Button Press

This example demonstrates the usage of interrupts and sleep modes. In this use case, the microcontroller exits sleep on button press, turns on an LED, and goes back to sleep. On button release, it exits sleep, turns off the LED, and goes back to sleep. The LED is ON while the button is pressed, but the microcontroller can be in sleep mode.

Pins can detect transitions from '0' to '1' (rising edge) and from '1' to '0' (falling edge). An interrupt can be triggered on one or both transitions. This is configured using the ISC bit field in the PORTx.PINnCTRL register.

Figure 5-1. ISC Bit Field Control Register

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

These interrupts wake the device from all sleep modes. In this example, on button press/release, the microcontroller will wake from sleep and turn an LED on/off and then go back to sleep. In this way, the LED will stay on only while the button is pressed.

First, the B2 pin, where the button is attached, is configured as input, and its interrupt is activated.

```
PORTB.DIR &= ~ PIN2_bm;
PORTB.PIN2CTRL |= PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc;
```

Note: When pressed, the button will close the circuit to ground, and the pin will read '0'. Therefore, whenever the button is not pressed, it must be connected to V_{DD} using a pull-up resistor. The pull-up resistor is activated using the PULLUPEN bit field in the PORTx.PINnCTRL register.

Second, the sleep mode is selected using the macros defined in the <avr/sleep.h> header. The modes of sleep are Idle, Standby and Power-Down. In this example, the deepest sleep mode is used.

```
set_sleep_mode(sleep_MODE_PWR_DOWN);
```

This only selects which sleep mode to be used. The following macro makes the microcontroller go to sleep.

```
sleep_mode();
```

Note: The sleep_mode() macro also enables sleep before sending the CPU sleep instruction and disables sleep when the microcontroller wakes up.

When going to sleep, the global interrupts must be enabled. Otherwise, there is no way to wake up. Using the sei() macro defined in the <avr/interrupt.h> header enables global interrupts. In this example, this is done in the initialization code.

Sometimes, there are blocks of instructions that need to execute without interruption. They are called atomic blocks. This is achieved by disabling interrupts at the beginning of the block and re-enabling them at the end of the block. For this, the ATOMIC_BLOCK macro in <avr/atomic.h> is used.

Finally, inside the Interrupt Service Routine (ISR), a flag is set to indicate a transition (rising or falling edge) happened. Because it is a good practice to keep the ISR short, the transition type check and LED toggling are handled in the main-line code.

```
ISR(PORTB_PORT_vect)
{
    if(PB2_INTERRUPT)
    {
        pb2Ioc = 1;
        PB2_CLEAR_INTERRUPT_FLAG;
    }
}
```

Because the same interrupt is used for all the pins in a port, the ISR code must check what pin triggered the interrupt. This is done by checking its flag in PORTx.INTFLAGS. In this example, to check whether the pin B2 triggered the interrupt, the following macro is used.

```
#define PB2_INTERRUPT PORTB.INTFLAGS & PIN2_bm
```

Clearing the interrupt flag is done by writing a '1' to its location in PORTn.INTFLAGS.

```
#define PB2_CLEAR_INTERRUPT_FLAG PORTB.INTFLAGS &= PIN2_bm
```

Note: If the flag is not cleared, the interrupt will keep triggering, so the flag must always be cleared before exiting the ISR. Any algorithm that relies on not clearing the interrupt flag is highly discouraged because this effectively overloads the ISR responsibility. The resulting responsibilities of the ISR will be to handle the interrupt and to keep firing until the flag is cleared. This violates the Single Responsibility principle in software design and extreme care must be taken to avoid bugs.

The main-line code checks the pin value ten milliseconds after the interrupt to decide whether it was a falling edge or a rising edge. If the pin is low (value '0'), then it must have been a falling edge, otherwise it was a rising edge. The ten milliseconds delay is a denouncing method.

```
#define PB2_LOW !(PORTB.IN & PIN2_bm)
```

View the ATmega4809 Code Example on GitHub
Click to browse repository

An MCC generated code example for AVR128DA48 with the same functionality as the one described in this section can be found here:

View the AVR128DA48 Code Example on GitHub
Click to browse repository

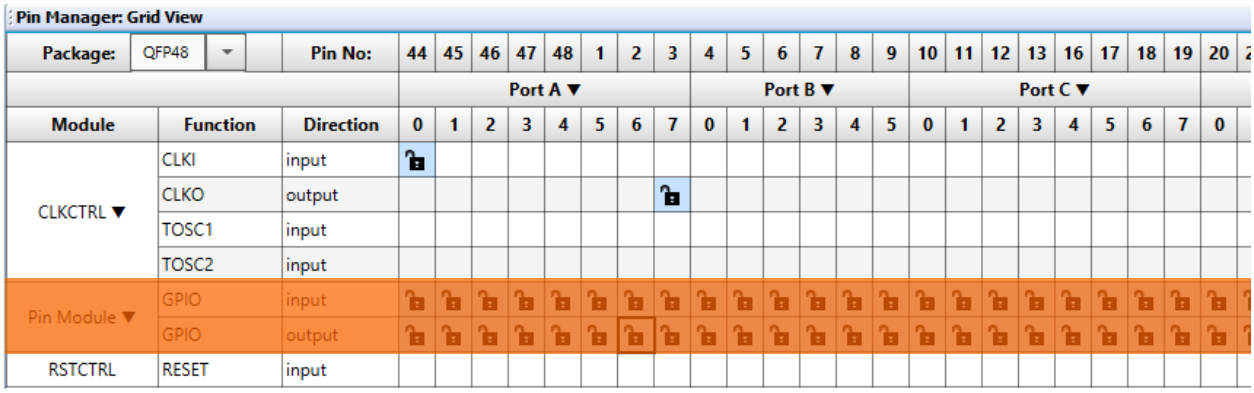
6. Using GPIO within MCC

The MCC is a user-friendly plug-in tool for MPLAB X and MPLAB Xpress IDEs, which generate drivers for controlling and driving peripherals of PIC® and AVR microcontrollers, based on the settings and selections made in the Graphical User Interface. The generated drivers can be used in any application program.

- For more details about MCC, see the MCC User's Guide.
- MPLAB Xpress is an online IDE that can be used in the browser.

In MCC, the GPIO pins can be configured in the Pin Manager: Grid View window.

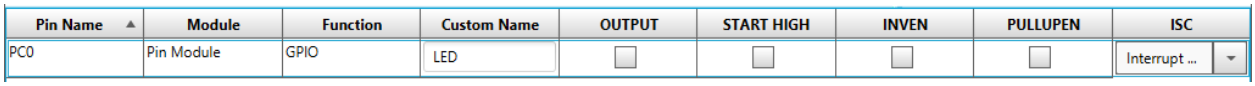
Figure 6-1. Pin Manager: Grid View Window



Clicking one of the locks will select the corresponding pin as either input or output, depending on the row clicked. A pin cannot act as input and output at the same time. If the run-time switch between the input and the output is needed, this will have to be manually implemented.

Once selected, the pins will also appear in the Pin Module window.

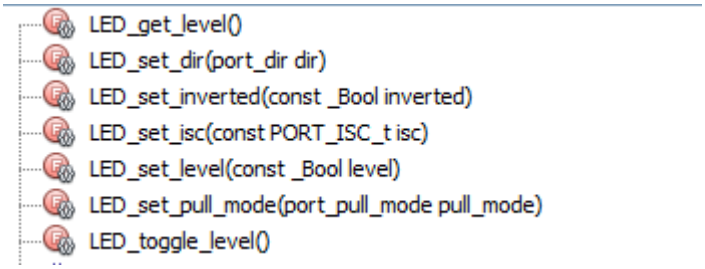
Figure 6-2. Pin Module Window



Various options can be configured here, most of which have already been discussed in this document. If the Output check box is unchecked, the pin will be input.

MCC generates macros for the pin usage. The prefix of the macros is the pin name, given in the Custom Name field, which is a helpful abstraction if, for example, one pin controls an LED and its macros can be prefixed with LED instead of the actual pin name.

Figure 6-3. Macros for the Pin Usage



7. References

1. AVR128DA48 Product Page: www.microchip.com/wwwproducts/en/AVR128DA28
2. AVR128DA48 Curiosity Nano Evaluation Kit webpage: www.microchip.com/Developmenttools/ProductDetails/DM164151
3. AVR128DA28/32/48/64 Data Sheet
4. Getting Started with the AVR® DA Family
5. ATmega4809 product page: www.microchip.com/wwwproducts/en/ATMEGA4809
6. megaAVR® 0-series Family Data Sheet (DS40002015)
7. ATmega809/1609/3209/4809 - 48-pin Data Sheet megaAVR® 0-series (DS40002016)
8. ATmega4809 Xplained Pro webpage: www.microchip.com/developmenttools/ProductDetails/atmega4809-xpro

8. Revision History

Document Revision	Date	Comments
B	01/2021	Updated the GitHub repository links. Added the *AVR® DA Family Overview*, *References* and *Revision History* sections. Added MCC versions for each use case, running on AVR128DA48. Other minor corrections.
A	10/2019	Initial document release.

The Microchip Website

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

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

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

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

Customer Support

Users of Microchip products can receive assistance through several channels:

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

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

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

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable." Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-7349-7

Quality Management System

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

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820