
AVR1201: Using External Interrupts for tinyAVR Devices

APPLICATION NOTE

Introduction

This application note illustrates the functionality and provides steps to configure the external interrupts available on the Atmel® tinyAVR® 8-bit Microcontroller Family of Atmel AVR® microcontrollers. The application note also describes the points to be considered while using a GPIO pin as an external interrupt source pin.

The example codes have been implemented on ATtiny88 device with Atmel Studio 7 and tested on an Atmel AVR STK®600 starter kit for functionality.

Features

- Flexible pin configuration
- Synchronous and asynchronous interrupt sensing
- Asynchronous wake-up signal to wake up from sleep modes
- Driver source code included for Atmel ATtiny88
 - Basic external interrupt usage on ATtiny88
 - Nested external interrupt usage on ATtiny88
 - Generating software interrupt with external interrupt pin configured as output on ATtiny88
 - Using external interrupt to wake up device on ATtiny88

Table of Contents

Introduction.....	1
Features.....	1
1. External Interrupt – Overview.....	3
1.1. External Interrupt Vectors.....	3
1.2. External Interrupt Sensing.....	4
1.2.1. Asynchronous Sensing in ATtiny88.....	5
1.2.2. Synchronous Sensing in ATtiny88.....	5
1.3. Interrupt Response Time.....	5
1.4. Interrupt Priority.....	5
1.5. Important Points to be Noted when using External Interrupts.....	6
2. Getting Started.....	7
2.1. Task 1: Basic External Interrupt Usage.....	7
2.2. Task 2: Nested External Interrupt Usage.....	8
2.3. Task 3: External Interrupt Based on Signal Change on Pin.....	8
2.4. Task 4: External Interrupt for Device Wake-up.....	9
3. Driver Implementation.....	10
4. Revision History.....	11

1. External Interrupt – Overview

Interrupts are signals provided to the CPU of the microcontroller unit, either from internal peripheral modules or from external pins of the MCU. It alters the regular flow of the program execution by enabling the CPU to make a jump to execute instruction routines in some pre-defined location based on the interrupt that occurred. When the CPU completes the routine, it gets back to the location from where it had made a jump.

These pre-defined locations are called as the interrupt vector addresses or interrupt vectors.

An interrupt causes the device to save its state of execution and start executing the interrupt handler. For Atmel AVR Microcontrollers the PC register is the only register that will be saved and the stack pointer register is updated in the event of an interrupt. It is up to the user to save other registers like the status register, 32 general purpose registers (register file), on an event of an interrupt, if there is such a requirement in the application.

The interrupts are commonly used to save more time (such as multitasking) than the conventional polling method (waiting for the event to occur indefinitely).

1.1. External Interrupt Vectors

The Atmel tinyAVR supports several interrupt sources out of which external interrupts are significant. The external interrupts can be triggered using two sets of pins. INTn pins (ordinary external interrupt pins) and PCINTn pins (pin change external interrupt pins). The 'n' varies from device to device and signifies the number like INT0. Refer to the respective device datasheet for the specific values of n.

For Atmel ATtiny88 the numbers on external interrupt pins are as given below.

- INT0 and INT1 Pins (with various input sense configurations)
- Pin change interrupt pins (PCINT27:0)

For each interrupt source, there is an interrupt vector to which the program execution control jumps, to execute the corresponding service routine. The interrupt vectors for external interrupts on ATtiny88 are shown in the following table. For device specific interrupt vectors, refer to the respective datasheet.

Table 1-1. External Interrupt Vector Address

Vector no.	Program address	Source	Port pins in ATmega2560	Interrupt definitions
1	\$000	RESET	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$001	INT0	PD2	External Interrupt Request 0
3	\$002	INT1	PD3	External Interrupt Request 1
4	\$003	PCINT0	PCINT0:7 - PB0:7	Pin Change Interrupt Request 0
5	\$004	INT3	PCINT8:15 - PC0:7	Pin Change Interrupt Request 1
6	\$005	INT4	PCINT16:23 - PD0:7	Pin Change Interrupt Request 2
7	\$006	INT5	PCINT24:27 - PA0:4 ⁽¹⁾	Pin Change Interrupt Request 3

Note: 1. Available in only 32-pin packages

The two external interrupt sources (INT0 & INT1) have dedicated interrupt vectors where as group of pin change interrupts share the same interrupt vector as listed in table above.

Any signal level change in any of the eight pins PCINT0:7 (if enabled) will trigger the interrupt PCINT0. This means that, if an interrupt is triggered by either the pin PCINT0 or PCINT4, the CPU will jump to the same vector address \$002. Similarly, any signal level change in any of the eight pins PCINT8:15 (if enabled) will trigger the interrupt PCINT1 and any signal level change in any of the eight pins PCINT16:23 (if enabled) will trigger the interrupt PCINT2 and any signal level change in any of the four pins PCINT24:27 (if enabled) will trigger the interrupt PCINT3.

For pin change interrupt each of PCINT0 to PCINT7 is OR'ed together and synchronized. It is up to the application code to solve the handling by keeping track of previous pin values and then in the interrupt routine scan the present pin values to check which pin has changed. The same is applicable for PCINT8:15 and PCINT16:23 and PCINT24:27

1.2. External Interrupt Sensing

External interrupts can be sensed and registered either synchronously or asynchronously. Synchronous sensing requires I/O clock whereas asynchronous sensing does not requires I/O clock. This implies that the interrupts that are detected asynchronously can be used for waking the device from sleep modes other than idle mode because the I/O clock is halted in all sleep modes except idle mode.

The sense configuration for external interrupts and pin change interrupts for Atmel ATtiny88 is given in table below. For device specific sense configuration, refer to the respective datasheet.

Table 1-2. External Interrupts Sense Configuration

Program address	Interrupt source	Sensing
\$001	INT0	Asynchronous (level)
		Synchronous (edges)
\$002	INT1	Asynchronous (level)
		Synchronous (edges)
\$003	PCINT0	Asynchronous
\$004	PCINT1	Asynchronous
\$005	PCINT2	Asynchronous
\$006	PCINT3	Asynchronous

From the table above all the pin change interrupts are detected asynchronously. Other interrupts (INT0 and INT1) can be triggered by sensing the rising or falling edges or low level on the corresponding interrupt pins. The type of sensing (edge or level) for each of the INTn (n = 0 and 1 for ATtiny88) interrupts is software configurable using two Interrupt Sense Control (ISC) bits per interrupt. This is provided in the following table.

Table 1-3. External Interrupts Individual Sense Configuration

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any edge of INTn generates an interrupt request
1	0	The falling edge of INTn generates an interrupt request
1	1	The rising edge of INTn generates an interrupt request

Note: PCINT27:0 does not have sense configuration options. This means that the interrupt will be generated whenever there is a logic change in the pin, that is, from high to low transition and low to high transition.

1.2.1. Asynchronous Sensing in ATtiny88

From table 'External interrupts sense configuration', level interrupts in INT0 and INT1 and all the PCINT27:0 are registered asynchronously. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

1.2.2. Synchronous Sensing in ATtiny88

From table 'External interrupts sense configuration', edges of interrupts INT0 and INT1 are registered synchronously. The value on the INT0 and INT1 pins are sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt.

1.3. Interrupt Response Time

The interrupt execution response for all the enabled AVR interrupts is four/five clock cycles minimum. This four/five clock cycles depends on the program counter width. If the program counter width is not more than two bytes, then the interrupt response time will be four clock cycles minimum and if the program counter width is more than two bytes, then the interrupt response time will be minimum five clock cycles.

These four/five clock cycles include:

- Two/Three cycles for pushing the Program Counter (PC) value into the stack
- One cycle for updating the stack pointer
- One cycle for clearing the Global interrupt enable (I) bit

If an interrupt occurs when the MCU is in sleep mode, the interrupt execution response time is increased by five clock cycles. This increase comes in addition to the start-up time from the selected sleep mode. This start-up time is the time it will take to start the clock source.

1.4. Interrupt Priority

Priority for the interrupts is determined by the interrupt vector address. An interrupt with lowest interrupt vector address has the highest priority. So reset has the highest priority followed by INT0, then INT1 and so on. If two interrupts occurs simultaneously, then the interrupt with higher priority is served first.

1.5. Important Points to be Noted when using External Interrupts

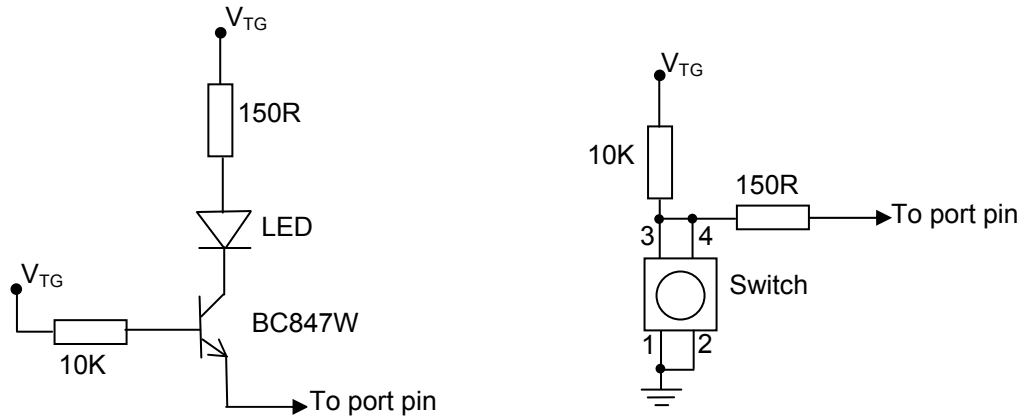
1. If a level triggered interrupt is used for waking up the device from Power-down, the required level must be held long enough for the MCU to complete the wake-up, to trigger the level interrupt. If the level disappears before the end of the start-up time, the MCU will still wake up, but no interrupt will be generated.
2. Both INT0 and INT1 should be configured to sense level interrupt to wake up the device from sleep mode other than idle mode.
3. If enabled, a level triggered interrupt will generate an interrupt request as long as the pin is held low.
4. When changing the ISCN bit, an interrupt can occur. Therefore, it is recommended to first disable INTn by clearing its Interrupt Enable bit in the EIMSK Register.
5. Before enabling an interrupt, it is recommended to clear the flag bit of the corresponding interrupt because when the flag bit is set, the interrupt will be triggered the moment we enable the interrupt.
6. If enabled, interrupts will be triggered even when the pins are configured as outputs. This provides a way of generating a software interrupt.
7. Most of the Atmel tinyAVR devices will have the reset pin multiplexed with some other functionality like an interrupt source or an ADC input channel. To use an interrupt pin (multiplexed with reset pin) the RSTDISBL (reset disable) fuse has to be programmed, otherwise the device will reset whenever the interrupt pin goes low. Once the RSTDISBL fuse has been programmed, the ISP interface becomes non-functional until the fuse is unprogrammed. So programming the AVR through ISP interface is not possible. High voltage programming should be used to unprogram the fuse.
8. Once the CPU enters the ISR, the global interrupt enable bit (I-bit) in SREG will be cleared so that all other interrupts are disabled. In order to use nested interrupts, the I-bit has to be set by software when the CPU enters an ISR.

2. Getting Started

This chapter walks you through the basic steps for getting started and running with the external interrupts on Atmel megaAVR[®] devices. The tasks given below simply use the switches and LEDs available on Atmel STK600. Without using STK600, these tasks can be verified by connecting a switch circuit and LED circuit to the port pins directly as shown in the figure below

It is to be noted that the first three tasks uses some delay loops inside the interrupt service routine to demonstrate the use of interrupts. Generally it is never recommended to use a delay routine inside the ISR.

Figure 2-1. Basic LED and Switch Circuit



2.1. Task 1: Basic External Interrupt Usage

Task: Enable INT0 interrupt and pin change interrupt PCINT0 to light up one LED for each, in their respective interrupt service routines.

1. Configure PORTC as output. To use the switches in the STK600 as interrupt source, enable pull-up on PORTB (for PCINT0) and PORTD (for INT0).
2. Configure INT0 to sense rising edge. Enable the interrupts INT0 and PCINT0, and set the global interrupt enable bit.
3. Inside the ISR of INT0, turn on LED0 and then turn off LED0 after some delay. Similarly for PCINT0, turn on and off LED1 with some delay in between.

Hardware Setup:

1. Connect PORTC header to the LED header on the STK600 using a ten-wire ribbon cable.
2. Connect two wires; one between pins SW0 and PD2, and the other between SW1 and PB0.

Without using STK600, connect two LED circuits as shown in [Figure 2-1](#) ; one at PC0 and another at PC1, and two switch circuits; one at PD2 and another at PB0.

While running the example code when switch SW0 is pressed, due to pull-up the interrupt INT0 will be triggered when SW0 is released and LED0 blinks once. When switch SW1 is pressed the interrupt PCINT0 is triggered and LED1 blinks once, and when SW1 is released the interrupt PCINT0 is triggered once again and LED1 blinks once again. So a single switch action (press and release) on SW1 produces two blinks on LED1.

2.2. Task 2: Nested External Interrupt Usage

As mentioned, once the CPU enters the ISR, the global interrupt enable bit (I-bit) in SREG will be cleared so that all other interrupts are disabled. In order to use nested interrupts, the I-bit is set by software when the CPU enters an ISR.

Task: Enable INT0 and INT1 interrupts. Within the ISR of INT0 set the I-bit so that INT1 interrupt will be sensed and executed (by jumping to ISR of INT1) while the CPU is inside ISR of INT0.

1. Configure PORTB0 and PORTC0 as outputs to turn ON LED0 and LED1 respectively. To use the switches in the Atmel STK600 as interrupt source, enable pull-up on PORTD (since PD2 and PD3 are INT0 and INT1 respectively).
2. Configure INT0 and INT1 to sense rising edge. Enable the interrupts INT0 and INT1 and set the global interrupt enable bit.
3. Inside the ISR of INT0, set the I-bit (using sei() instruction), turn on LED0 and then turn off LED0 after some delay. Similarly for INT1, turn on and off LED1 with some delay in between.

Hardware Setup:

1. Connect PB0 pin to LED0 and PC0 pin to LED1 using wires, on the STK600.
2. Connect two wires; one between pins SW0 and PD2 (for INT0), and the other between SW1 and PD3 (for INT1).

Without using STK600, connect two LED circuits as shown in [Figure 2-1](#); one at PB0 and another at PC0 and two switch circuits; one at PD0 and another at PD1.

While running the example code when switch SW0 is pressed, due to pull-up the interrupt INT0 will be triggered only when SW0 is released and LED0 blinks for some time. During the time while LED0 is glowing, a switch action on SW1 will trigger INT1 and hence the LED1 blinks for a moment and goes off. This is because, inside the ISR of INT0, the I-bit is set so that all other interrupts are activated. So even when the CPU is inside the INT0 routine it senses the interrupt INT1 and jumps to the ISR of INT1 and executes the routine and then jumps back to ISR of INT0.

Comment the line 'sei();' in the ISR of INT0 and observe the output.

2.3. Task 3: External Interrupt Based on Signal Change on Pin

As mentioned before, once an interrupt is enabled it will be triggered even when the corresponding pin is configured as output.

Task: Enable 16-bit Timer 1 in CTC mode with OC1A pin (PB1 pin - configured as output) (also PCINT1 pin) toggling on compare match. Enable the interrupt PCINT1 with the ISR containing a routine that turns on and off the LED0 connected to PORTC. Pin Change Interrupt PC10 triggers if a pin in PCINT[7:0] is toggled while enabled. Here PCINT1 is enabled.

1. Configure PORTC0 as output to drive LED0.
2. Enable pin change interrupt PCINT1. Also set the global interrupt enable bit.
3. Configure Timer1 to operate in CTC mode (OCR1A as TOP) with OC1A pin toggling on compare match. Load OCR1A with some value.
4. Configure OC1A pin (PCINT1 pin/PB1 pin) as output and start the timer with some prescaler value (in the example code it is divide by 64).
5. Within the ISR turn on LED0 connected to PORTC0 and turn off LED0 after some delay.

Hardware Setup:

1. Connect a wire between pins PC0 and LED0.
2. Connect a wire between pins PB1 and LED1 to view the OC1A output.

Without using STK600, connect two LED circuits as shown in [Figure 2-1](#); one at PC0 and another at PB1.

When running the example code, LED1 (connected to OC1A pin) toggles because of timer action. Whenever LED1 switches OFF (means a transition from low to high – a rising edge) or switches ON (means a transition from high to low – a falling edge), PCINT1 interrupt is triggered and so LED0 blinks once.

2.4. Task 4: External Interrupt for Device Wake-up

Task: Enable INT0 and set the device in sleep mode. Use INT0 to wake up the device and turn ON the LED to indicate that the device is in active mode.

1. Configure PORTB0 as output to drive LED0.
2. Enable pull-up on PORTC0 and PORTD2 to connect to switches SW0 and SW1 respectively.
3. Configure INT0 (on PORTD2) to sense level and enable INT0. Also, set the global interrupt enable bit.
4. Set sleep mode to power-down mode and turn ON LED0.
5. Wait until switch SW0 is pressed. Once it is pressed, turn OFF LED0 (to indicate that the device enters sleep mode) and enter into sleep mode.
6. Inside the ISR (after wake-up) turn ON LED0 to indicate that the device is in active mode now. Repeat steps 5 and 6.

Hardware Setup:

1. Connect a wire between pins PB0 and LED0.
2. Connect a wire between pins PC0 and SW0 (this connection is to make the device enter sleep mode).
3. Connect a wire between pins PD2 and SW1 (External Interrupt INT0 connection).

Without using STK600, connect one LED circuit as shown in [Figure 2-1](#) at PB0 and two switch circuits; one at PC0 and another at PD2.

By executing the example code LED0 will be turned ON. Once SW0 is pressed LED0 is turned OFF and the device enters sleep mode. Now a switch action on SW1 wakes up the device and turns ON LED0.

3. Driver Implementation

The example code is written for Atmel START. It can be downloaded from the "BROWSE EXAMPLES" entry of Atmel START for both Atmel Studio 7 and IAR™ IDE. Double click the downloaded .atzip file and project will be imported to Atmel Studio 7.

To import the project in IAR, refer "[Atmel START in IAR](#)", select Atmel Start Output in External Tools -> IAR.

Note: This external interrupt driver is not intended for use with high-performance code. It is designed as a library to get started with the external interrupts.

The example codes included are:

- Basic external interrupt usage on ATtiny88
- Nested external interrupt usage on ATtiny88
- Generating software interrupt with external interrupt pin configured as output in ATtiny88
- External interrupt for device wake-up of ATtiny88

4. Revision History

Doc. Rev.	Date	Comments
8469B	09/2016	Renamed AVR Studio 5 to Atmel Studio. 2.1 External interrupt vectors: Updated Table 1-1. "External Interrupts Sense Configuration".
8469A	01/2011	Initial document release

Atmel®, Atmel logo and combinations thereof, Enabling Unlimited Possibilities®, AVR®, megaAVR®, tinyAVR®, STK® and others are registered trademarks or trademarks of Atmel Corporation in U.S. and other countries. Other terms and product names may be trademarks of others.

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 THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, 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 AND 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 products 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 products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death ("Safety-Critical Applications") without an Atmel officer's specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.