Atmel AVR1201: Using External Interrupts for tinyAVR Devices

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 in ATtiny88**
  - **Using external interrupt to wake up device on ATtiny88**

1 Introduction

This application note illustrates the functionality and configuration steps (usage) of the external interrupts available in Atmel tinyAVR® 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 AVR Studio® 5 and tested on Atmel STK®600 starter kit for functionality.

8-bit Atmel Microcontrollers

Application Note

Rev. 8469A-AVR-11/11

2 External interrupt – Overview

Interrupts are signals given to the CPU of the microcontroller unit, either from internal peripheral modules or from external pins of the MCU that alters the regular flow of the program execution by making the CPU to make a jump to execute instruction routines in some other pre-defined location depending on the interrupt that occurred. Once 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 like for AVR Microcontrollers the PC register is the only register that will be saved and the stack pointer register is updated on an 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 time (like multitasking) than the conventional polling method (waiting for the event to occur indefinitely).

2.1 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. Please refer to respective device datasheet for the specific values of n.

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

1. INT0 and INT1 Pins (with various input sense configurations).
2. Pin change interrupts pins (PCINT27:0).

Generally 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 the external interrupts for ATtiny88 are shown in Table 2-1. For device specific interrupt vectors, please refer to the respective datasheet.

Table 2-1. External interrupt vector address table.

Vector no.	Program address	Source	Port pins in ATtiny88	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	PCINT1	PCINT8:15 – PC0:7	Pin change interrupt request 1
6	$005	PCINT2	PCINT15:23 – PD0:7	
7	$006	PCINT3	PCINT24:27 – PA0:4*	Pin change interrupt request 2

*- 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 2-1.

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 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. Similar is the case for PCINT8:15, PCINT16:23 and PCINT24:27.

2.2 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 2-2. For device specific sense configuration, please refer to the respective datasheet.

Table 2-2. 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 Table 2-2, all the pin change interrupts are detected asynchronously. Other interrupts (INT0 & 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 given in Table 2-3.

Table 2-3. 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).

2.2.1 Asynchronous sensing in Atmel ATtiny88

From Table 2-2, 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.

2.2.2 Synchronous sensing in Atmel ATtiny88

From Table 2-2, 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.

2.3 Interrupt response time

The interrupt execution response for all the enabled AVR interrupts is minimum four/five clock cycles. 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:

1. Two/three cycles for pushing the Program Counter (PC) value into the stack.
2. One cycle for updating the stack pointer.
3. 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.

2.4 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.

2.5 Important points to be noted when using external interrupts

1. 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.
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.

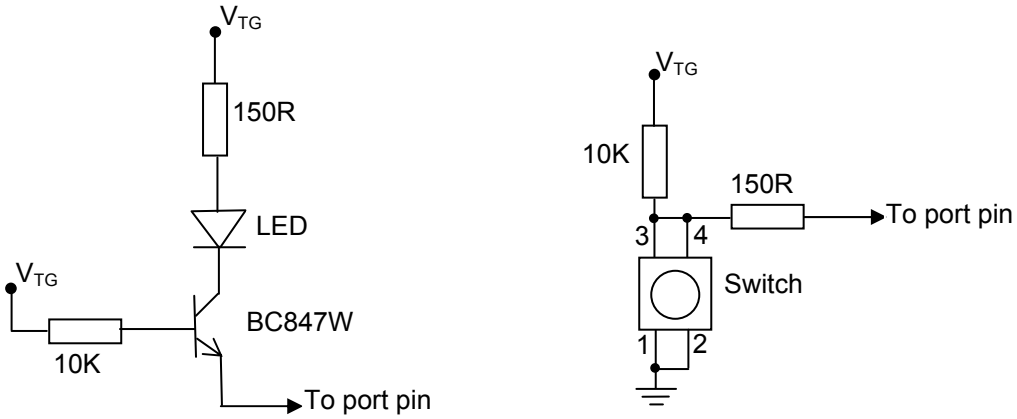
5. Before enabling an interrupt, it is recommended to clear the flag bit of the corresponding interrupt. Because when the flag bit is set, the interrupt will be triggered the moment we enable the interrupt.
6. If enabled, interrupts will be triggered even when the pins are configured as outputs. This 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 resets 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.

3 Getting started

This section walks you through the basic steps for getting started and running with the external interrupts on Atmel tinyAVR devices. The tasks given below simply use the switches and LEDs available on Atmel STK600. Without using STK600, these tasks can be verified by simply connecting a switch circuit and LED circuit to the port pins directly as shown in Figure 3-1.

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 3-1. Basic LED and switch circuit.



3.1 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 in the STK600 using a ten wire ribbon cable.
2. Connect two wires, one between the pins SW0 and PD2 and the other between SW1 and PB0.

Without using STK600, connect two LED circuits as shown in Figure 3-1, 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.

3.2 Task 2 – Nested external interrupt usage

As mentioned before, 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 & PD3 are INT0 & 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 the 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 3-1, one at PB0 and another at PC0 and two switch circuits, one at PD2 and another at PD3.

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.

Please comment the line ‘sei();’ in the ISR of INT0 and check what happens!

3.3 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.

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 prescalar 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 Atmel STK600, connect two LED circuits as shown in Figure 3-1, one at PC0 and another at PB1.

When running the example code, the LED1 (connected to OC1A pin) toggles because of timer action. Whenever the 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.

3.4 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 the switch SW0 is pressed and 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 3-1, at PB0 and two switch circuits, one at PC0 and another at PD2.

By running 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.

4 Driver implementation

This application note includes a source code package written for Atmel AVR Studio 5 IDE with AVR tool chain. Please note that 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:

1. Basic external interrupt usage on Atmel ATtiny88.
2. Nested external interrupt usage on ATtiny88.
3. Generating software interrupt with external interrupt pin configured as output in ATtiny88.
4. External interrupt for device wake-up of ATtiny88.

5 Resources

- Atmel tinyAVR datasheets
  http://www.atmel.com/avr
- Atmel AVR Studio with help files
  http://www.atmel.com/avrstudio

6 Atmel technical support center

Atmel has several support channels available:

- Web portal: http://support.atmel.no/ All Atmel microcontrollers
- Email: avr@atmel.com All Atmel AVR products
- Email: avr32@atmel.com All 32-bit AVR products

Please register on the web portal to gain access to the following services:

- Access to a rich FAQ database
- Easy submission of technical support requests
- History of all your past support requests
- Register to receive Atmel microcontrollers' newsletters
- Get information about available trainings and training material

7 Table of contents

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

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Milennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan
16F, Shin Osaki Kangyo Bldg.
1-6-4 Osaki Shinagawa-ku
Tokyo 104-0032
JAPAN
Tel: (+81) 3-6417-0300
Fax: (+81) 3-6417-0370

8469A-AVR-11/11