AVR 8-bit Microcontroller

AVR1200: Using External Interrupts for megaAVR Devices

APPLICATION NOTE

Introduction

This application note illustrates the functionality and provides steps to configure the external interrupts available on the Atmel® megaAVR® 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 an ATmega2560 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 ATmega2560
  - Basic external interrupt usage on ATmega2560
  - Nested external interrupt usage on ATmega2560
  - Generating software interrupt with external interrupt pin configured as output on ATmega2560
  - Using external interrupt to wake up device on ATmega2560

Table of Contents

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 megaAVR 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 ATmega2560 the numbers on external interrupt pins are:

- INT7:0 Pins (with various input sense configurations)
- Pin change interrupts pins (PCINT23:0)

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 ATmega2560 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	$0000	RESET	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	PD0	External Interrupt Request 0
3	$0004	INT1	PD1	External Interrupt Request 1
4	$0006	INT2	PD2	External Interrupt Request 2
5	$0008	INT3	PD3	External Interrupt Request 3
6	$000A	INT4	PE4	External Interrupt Request 4
7	$000C	INT5	PE5	External Interrupt Request 5

Vector no.	Program address	Source	Port pins in ATmega2560	Interrupt definitions
8	$000E	INT6	PE6	External Interrupt Request 6
9	$0010	INT7	PE7	External Interrupt Request 7
10	$0012	PCINT0	PCINT0:7 – PB0:7	Pin Change Interrupt Request 0
11	$0014	PCINT1	PCINT8 – PE0 PCINT9:15 – PJ0:PJ6	Pin Change Interrupt Request 1
12	$0016	PCINT2	PCINT16:23 – PK0:7	Pin Change Interrupt Request 2

The eight external interrupt sources (INT7:0) have dedicated interrupt vectors where a group of pin change interrupts share the same interrupt vector as listed in Table 1-1 External Interrupt Vector Address.

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 $0012. 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.

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.

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 ATmega2560 is given in Table 1-2 External Interrupts Sense Configuration. For device specific sense configuration, refer to the respective datasheet.

Table 1-2. External Interrupts Sense Configuration

Program address	Interrupt source	Sensing
$0002	INT0	Asynchronous (Edges and level)
$0004	INT1	Asynchronous (Edges and level)
$0006	INT2	Asynchronous (Edges and level)
$0008	INT3	Asynchronous (Edges and level)
$000A	INT4	Synchronous (Edges and level)
$000C	INT5	Synchronous (Edges and level)
$000E	INT6	Synchronous (Edges and level)
$0010	INT7	Synchronous (Edges and level)

Program address	Interrupt source	Sensing
$0012	PCINT0	Asynchronous
$0014	PCINT1	Asynchronous
$0016	PCINT2	Asynchronous

From Table 1-2 External Interrupts Sense Configuration all the pin change interrupts are detected asynchronously. Other interrupts (INT7:0) 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 to 7 for Atmel ATmega2560) 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: PCINT23:0 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 ATmega2560

From Table 1-2 External Interrupts Sense Configuration interrupts INT3:0 are registered asynchronously. Pulses on INT3:0 pins wider than the minimum pulse width (typically 50ns for ATmega2560) will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt.

1.2.2. Synchronous Sensing in ATmega2560

From Table 1-2 External Interrupts Sense Configuration, interrupts INT7:4 are registered synchronously. The value on the INT7:4 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. For both types of sensing, if low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

1.3. Interrupt Response Time

The interrupt execution response for all the enabled AVR interrupts is four/five clock cycle's 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.

If an interrupt occurs during execution of a multi-cycle instruction, this instruction is completed before the interrupt is served.

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

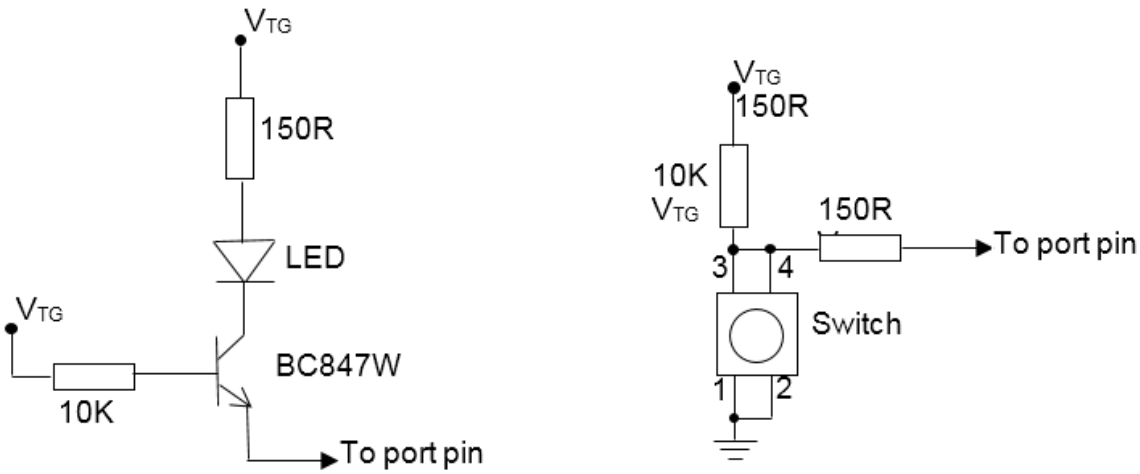
1. If a level triggered interrupt is used for waking up the device from Power-down, the required level must be held long enough for the MCU to complete the wake-up, to trigger the level interrupt. If the level disappears before the end of the start-up time, the MCU will still wake up, but no interrupt will be generated.
2. Both INT7:0 and pin change interrupts can be used to wake up the device from any sleep modes. But INT7:4 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. If a logic high level (“one”) is present on an asynchronous external interrupt pin configured as “Interrupt on Rising Edge, Falling Edge, or Any Logic Change on Pin” while the external interrupt is not enabled, the corresponding External Interrupt Flag will be set when resuming from the power-down, power-save, and standby sleep modes.
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 section 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 Figure 2-1 Basic LED and Switch Circuit.

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 PORTA 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 PORTA header to the LED header on STK600 using a ten-wire ribbon cable.
2. Connect two wires; one between the pins SW0 and PD0, and the other between SW1 and PB0.

Without using STK600, connect two LED circuits as shown in Figure 2-1 Basic LED and Switch Circuit; one at PA0 and another at PA1, and two switch circuits, one at PD0 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 PD0 & PD1 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 PD0 (for INT0), and the other between SW1 and PD1 (for INT1).

Without using STK600 connect two LED circuits as shown in Figure 2-1 Basic LED and Switch Circuit; 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 in Step 6 once an interrupt is enabled it will be triggered even when the corresponding pin is configured as output.

Task: Enable 16-bit Timer 3 in CTC mode with OC3B pin (PE4 pin - configured as output) (also INT4 pin) toggling on compare match. Enable the interrupt INT4 with the ISR containing a routine that turns on and off the LED0 connected to PORTA.

1. Configure PORTA0 as output to drive LED0.
2. Configure INT4 to sense rising edge and enable INT4. Also set the global interrupt enable bit.
3. Configure Timer3 to operate in CTC mode (OCR3A as TOP) with OC3B pin toggling on compare match. Load OCR3A with some value.
4. Configure OC3B pin (INT4 pin/PE4 pin) as output and start the timer with some pre-scalar value (in the example code it is divide by 256).
5. Within the ISR, turn ON LED0 connected to PORTA0 and turn OFF LED0 after some delay.

Hardware Setup:

1. Connect a wire between pins PA0 and LED0.
2. Connect a wire between pins PE4 and LED1 to view the OC3B output.

Without using STK600, connect two LED circuits as shown in Figure 2-1 Basic LED and Switch Circuit; one at PA0 and another at PE4.

When running the example code, the LED1 (connected to OC3B pin) toggles because of timer action. Whenever the LED1 switches OFF (means a transition from low to high – a rising edge), INT4 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 PORTA0 as output to drive LED0.
2. Enable pull-up on PORTB0 and PORTD0 to connect to switches SW0 and SW1 respectively.
3. Configure INT0 (on PORTD0) to sense rising edge 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. 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 PA0 and LED0.
2. Connect a wire between pins PB0 and SW0 (this connection is to make the device enter sleep mode).
3. Connect a wire between pins PD0 and SW1 (External Interrupt INT0 connection).

Without using STK600, connect one LED circuit as shown in Figure 2-1 Basic LED and Switch Circuit at PA0 and two switch circuits; one at PB0 and another at PD0.

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

This application note includes a source code package written for Atmel Studio 7.0 IDE with AVR tool chain. 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:

- Basic external interrupt usage on Atmel ATmega2560
- Nested external interrupt usage on ATmega2560
- Generating software interrupt with external interrupt pin configured as output in ATmega2560
- External interrupt for device wake-up of ATmega2560

4. Revision History

Doc. Rev.	Date	Comments
8468B	03/2016	Updated the firmware to the latest version of Atmel Studio (Atmel Studio 7) and made minor changes in the document.
8468A	11/2011	Initial document release.

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**