Interrupts and 16-bit Timer/Counter 1 (Normal Mode)
February 2009
Chapter 10: AVR Interrupt Programming in Assembly and C

Section 10.1: AVR Interrupts

Section 10.4: Interrupt Priority in the AVR
# Table of Contents

Interrupt Basics .............................................................................................................................................. 4
The Main Reasons You Might Use Interrupts .................................................................................................. 5
ATmega328P Interrupt Vector Table .............................................................................................................. 6
ATmega328P Interrupt Processing ................................................................................................................. 8
ATmega328P Interrupt Processing – By the Numbers .................................................................................... 9
ATmega328P Interrupt Processing – Type 1 – .............................................................................................. 10
ATmega328P Interrupt Processing – Type 2 – .............................................................................................. 11
When Writing an Interrupt Service Routine (ISR) ....................................................................................... 12
Program Initialization and the Interrupt Vector Table (IVT) ........................................................................ 13
The Interrupt Service Routine (ISR) ........................................................................................................... 14
Predefined Arduino IDE Interrupts ............................................................................................................... 15
Programming the Arduino to Handle External Interrupts ........................................................................... 16
**Interrupt Basics**

- A microcontroller normally executes instructions in an orderly fetch-execute sequence as dictated by a user-written program.
- However, a microcontroller must also be ready to handle unscheduled, events that might occur inside or outside the microcontroller.
- The interrupt system onboard a microcontroller allows it to respond to these internally and externally generated events. By definition we do not know when these events will occur.
- When an interrupt event occurs, the microcontroller will normally complete the instruction it is currently executing and then transition program control to an Interrupt Service Routine (ISR). These ISRs, which handles the interrupt.
- Once the ISR is complete, the microcontroller will resume processing where it left off before the interrupt event occurred.

(a) Program with ISR  
(b) ISR called from anywhere
THE MAIN REASONS YOU MIGHT USE INTERRUPTS

- To detect pin changes (eg. rotary encoders, button presses)
- Watchdog timer (eg. if nothing happens after 8 seconds, interrupt me)
- Timer interrupts - used for comparing/overflowing timers
- SPI data transfers
- I2C data transfers
- USART data transfers
- ADC conversions (analog to digital)
- EEPROM ready for use
- Flash memory ready

1 Source: Gammon Software Solutions forum – What are interrupts?
ATmega328P Interrupt Vector Table

- The ATmega328P provides support for 25 different interrupt sources. These interrupts and the separate Reset Vector each have a separate program vector located at the lowest addresses in the Flash program memory space.

- The complete list of vectors is shown in Table 11-6 “Reset and Interrupt Vectors in ATmega328P. Each Interrupt Vector occupies two instruction words.

- The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 – the External Interrupt Request 0.
<table>
<thead>
<tr>
<th>Vector No</th>
<th>Program Address</th>
<th>Source</th>
<th>Interrupt Definition</th>
<th>Arduino/C++ ISR() Macro Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x0000</td>
<td>RESET</td>
<td>Reset</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0x0002</td>
<td>INT0</td>
<td>External Interrupt Request 0 (pin D2)</td>
<td>(INT0_vect)</td>
</tr>
<tr>
<td>3</td>
<td>0x0004</td>
<td>INT1</td>
<td>External Interrupt Request 1 (pin D3)</td>
<td>(INT1_vect)</td>
</tr>
<tr>
<td>4</td>
<td>0x0006</td>
<td>PCINT0</td>
<td>Pin Change Interrupt Request 0 (pins D8 to D13)</td>
<td>(PCINT0_vect)</td>
</tr>
<tr>
<td>5</td>
<td>0x0008</td>
<td>PCINT1</td>
<td>Pin Change Interrupt Request 1 (pins A0 to A5)</td>
<td>(PCINT1_vect)</td>
</tr>
<tr>
<td>6</td>
<td>0x000A</td>
<td>PCINT2</td>
<td>Pin Change Interrupt Request 2 (pins D0 to D7)</td>
<td>(PCINT2_vect)</td>
</tr>
<tr>
<td>7</td>
<td>0x000C</td>
<td>WDT</td>
<td>Watchdog Time-out Interrupt</td>
<td>(WDT_vect)</td>
</tr>
<tr>
<td>8</td>
<td>0x000E</td>
<td>TIMER2 COMPA</td>
<td>Timer/Counter2 Compare Match A</td>
<td>(TIMER2_COMPA_vect)</td>
</tr>
<tr>
<td>9</td>
<td>0x0010</td>
<td>TIMER2 COMPB</td>
<td>Timer/Counter2 Compare Match B</td>
<td>(TIMER2_COMPB_vect)</td>
</tr>
<tr>
<td>10</td>
<td>0x0012</td>
<td>TIMER2 OVF</td>
<td>Timer/Counter2 Overflow</td>
<td>(TIMER2_OVF_vect)</td>
</tr>
<tr>
<td>11</td>
<td>0x0014</td>
<td>TIMER1 CAPT</td>
<td>Timer/Counter1 Capture Event</td>
<td>(TIMER1_CAPT_vect)</td>
</tr>
<tr>
<td>12</td>
<td>0x0016</td>
<td>TIMER1 COMPA</td>
<td>Timer/Counter1 Compare Match A</td>
<td>(TIMER1_COMPA_vect)</td>
</tr>
<tr>
<td>13</td>
<td>0x0018</td>
<td>TIMER1 COMPB</td>
<td>Timer/Counter1 Compare Match B</td>
<td>(TIMER1_COMPB_vect)</td>
</tr>
<tr>
<td>14</td>
<td>0x001A</td>
<td>TIMER1 OVF</td>
<td>Timer/Counter1 Overflow</td>
<td>(TIMER1_OVF_vect)</td>
</tr>
<tr>
<td>15</td>
<td>0x001C</td>
<td>TIMER0 COMPA</td>
<td>Timer/Counter0 Compare Match A</td>
<td>(TIMER0_COMPA_vect)</td>
</tr>
<tr>
<td>16</td>
<td>0x001E</td>
<td>TIMER0 COMPB</td>
<td>Timer/Counter0 Compare Match B</td>
<td>(TIMER0_COMPB_vect)</td>
</tr>
<tr>
<td>17</td>
<td>0x0020</td>
<td>TIMER0 OVF</td>
<td>Timer/Counter0 Overflow</td>
<td>(TIMER0_OVF_vect)</td>
</tr>
<tr>
<td>18</td>
<td>0x0022</td>
<td>SPI, STC</td>
<td>SPI Serial Transfer Complete</td>
<td>(SPI_STC_vect)</td>
</tr>
<tr>
<td>19</td>
<td>0x0024</td>
<td>USART, RX</td>
<td>USART Rx Complete</td>
<td>(USART_RX_vect)</td>
</tr>
<tr>
<td>20</td>
<td>0x0026</td>
<td>USART, UDRE</td>
<td>USART, Data Register Empty</td>
<td>(USART_UDRE_vect)</td>
</tr>
<tr>
<td>21</td>
<td>0x0028</td>
<td>USART, TX</td>
<td>USART, Tx Complete</td>
<td>(USART_TX_vect)</td>
</tr>
<tr>
<td>22</td>
<td>0x002A</td>
<td>ADC</td>
<td>ADC Conversion Complete</td>
<td>(ADC_vect)</td>
</tr>
<tr>
<td>23</td>
<td>0x002C</td>
<td>EE READY</td>
<td>EEPROM Ready</td>
<td>(EE_READY_vect)</td>
</tr>
<tr>
<td>24</td>
<td>0x002E</td>
<td>ANALOG COMP</td>
<td>Analog Comparator</td>
<td>(ANALOG_COMP_vect)</td>
</tr>
<tr>
<td>25</td>
<td>0x0030</td>
<td>TWI</td>
<td>2-wire Serial Interface (I2C)</td>
<td>(TWI_vect)</td>
</tr>
<tr>
<td>26</td>
<td>0x0032</td>
<td>SPM READY</td>
<td>Store Program Memory Ready</td>
<td>(SPM_READY_vect)</td>
</tr>
</tbody>
</table>
ATmega328P Interrupt Processing

- When an interrupt occurs, the microcontroller completes the current instruction and stores the address of the next instruction on the stack.

- It also turns off the interrupt system to prevent further interrupts while one is in progress. This is done by clearing the SREG Global Interrupt Enable I-bit.

- The Interrupt flag bit is cleared for Type 1 Interrupts only (see the next page for Type definitions).

- The execution of the ISR is performed by loading the beginning address of the ISR specific for that interrupt into the program counter. The AVR processor starts running the ISR.

- Execution of the ISR continues until the return from interrupt instruction (reti) is encountered. The SREG I-bit is automatically set when the reti instruction is executed (i.e., Interrupts enabled).

- When the AVR exits from an interrupt, it will always return to the interrupted program and execute one more instruction before any pending interrupt is served.

- The Status Register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

```
push   reg_F
in     reg_F, SREG
:      SREG, reg_F
pop    reg_F
```
ATmega328P Interrupt Processing – By the Numbers

Working with Peripheral Subsystems
- Polling
- Interrupts
  - Polling
  - Priority Vectored

Flash
- rjmp reset
- jmp TOV_ISR
- reset:
  - cli
  - sei
  - loop:
    - rcall Delay
- clr r16
- rcall B
- rjmp loop

SRAM
- 0x0000
- 0x0010
- 0x0018
- 0x0020
- 0x0024
- 0x0028
- 0x002C
- 0x0030
- 0x0034
- 0x0038
- 0x003C
- 0x0040
- 0x0044
- 0x0048

SRAM Bank
- 0x0000
- 0x0004
- 0x0008
- 0x000C
- 0x0010
- 0x0014
- 0x0018
- 0x001C
- 0x0020
- 0x0024
- 0x0028
- 0x002C
- 0x0030
- 0x0034
- 0x0038
- 0x003C
- 0x0040
- 0x0044
- 0x0048

Flash
- PC = 0x001A
- jmp TOV_ISR
- rjmp reset
- ldi r16,0x0B
- sts TCNT1H,r16
- ldi r16,0x0C
- sts TCNT1L,r16
- pop,r16
- out SREG,r16
- pop r15
- ret

Polling
- TOV_ISR:
  - push r15
  - in r15,SREG
  - push r16
  - sbis TIFR1,TOV1
  - rjmp wait
  - sbi TIFR1,TOV1
  - ldi r16,0x0B
  - sts TCNT1H,r16
  - ldi r16,0x0C
  - sts TCNT1L,r16
  - pop,r16
  - out SREG,r16
  - pop r15
  - ret

Priority Vectored
- TIFR1
- TOV1
ATmega328P Interrupt Processing – Type 1 –

- The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine.
  - The SREG I-bit is automatically set to logic one when a Return from Interrupt instruction – RETI – is executed.
- There are basically two types of interrupts...
  - The first type (Type 1) is triggered by an event that sets the Interrupt Flag. For these interrupts, the Program Counter is vectored to the actual Interrupt Vector in order to execute the interrupt handling routine, and hardware clears the corresponding Interrupt Flag.
    - If the same interrupt condition occurs while the corresponding interrupt enable bit is cleared, the Interrupt Flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software (interrupt cancelled).
    - Interrupt Flag can be cleared by writing a logic one to the flag bit position(s) to be cleared.
  - If one or more interrupt conditions occur while the Global Interrupt Enable (SREG I) bit is cleared, the corresponding Interrupt Flag(s) will be set and remembered until the Global Interrupt Enable bit is set on return (reti), and will then be executed by order of priority.
ATmega328P Interrupt Processing – Type 2 –

- The second type (Type 2) of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have Interrupt Flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered.
When Writing an Interrupt Service Routine (ISR)²

- As a general rule get in and out of ISRs as quickly as possible. For example do not include timing loops inside of an ISR.
- If you are writing an Arduino program
  - Don't add delay loops or use function `delay()`
  - Don't use function `Serial.print(val)`
  - Make variables shared with the main code `volatile`
  - Variables shared with main code may need to be protected by "critical sections" (see below)
  - Toggling interrupts off and on is not recommended. The default in the Arduino is for interrupts to be enabled. Don’t disable them for long periods or things like timers won't work properly.

² Source: Gammon Software Solutions forum – What are interrupts?
PROGRAM INITIALIZATION AND THE INTERRUPT VECTOR TABLE (IVT)

- Start by jumping over the Interrupt Vector Table

  RST_VECT:
  
  rjmp reset

- Add jumps in the IVT to your ISR routines

  .ORG INT0addr       // 0x0002 External Interrupt 0
  jmp INT0_ISR

  .ORG OVVF1addr
  jmp TOVF1_ISR

- Initialize Variables, Configure I/O Registers, and Set Local Interrupt Flag Bits

  reset:
  
  lds r16, EICRA       // EICRA Memory Mapped Address 0x69
  sbr r16, 0b000000010
  cbr r16, 0b000000001
  sts EICRA, r16       // ISC0=[10] (falling edge)
  sbi EIMSK, INT0      // Enable INTO interrupts

- Enable interrupts at the end of the initialization section of your code.

  sei                   // Global Interrupt Enable

loop:
THE INTERRUPT SERVICE ROUTINE (ISR)

; -- Interrupt Service Routine --
INT0_ISR:
    push   reg_F
    in     reg_F,SREG
    push   r16
    ; Load
    ; Do Something
    ; Store
    pop    r16
    out    SREG,reg_F
    pop    reg_F
    reti

; ---------------------------------------------
**Predefined Arduino IDE Interrupts**

- When you push the reset button the ATmega328P automatically runs an Arduino Boot program located in a separate Boot Flash section at the top of program memory. This Boot program configures and enables two interrupts.

  17 0x0020  TIMER0 OVF Timer/Counter0 Overflow (TIMER0_OVF_vect)

- The `millis()` and `micros()` function calls make use of the "timer overflow" feature utilize timer 0. The ISR runs roughly 1000 times a second, and increments an internal counter which effectively becomes the `millis()` counter (see On your own question).

  19 0x0024  USART, RX  USART Rx Complete (USART_RX_vect)

  21 0x0028  USART, TX  USART, Tx Complete (USART_TX_vect)

- The hardware serial library uses interrupts to handle incoming and outgoing serial data. Your program can now be doing other things while data in an SRAM buffer is sent or received. You can check the status of the buffer by calling the `Serial.available()` function.

- **On your own.** Given that you are using 8-bit Timer/Counter 0, you have set TCCR0B bits CS02:CS01:CS00 = 0b011 (clk/64), and the system clock f_{clk} = 16 MHz, what value would you preload into the Timer/Counter Register TCNT0 to get a interrupt 1000 times a second.

Source: [Gammon Software Solutions forum](http://www.arduino.cc/playground/A0004) – this blog also covers how to work with all the interrupts in C++ and the Arduino scripting language.
PROGRAMMING THE ARDUINO TO HANDLE EXTERNAL INTERRUPTS

- Variables shared between ISRs and normal functions should be declared "volatile". This tells the compiler that such variables might change at any time, and thus the compiler should not “optimize” the code by placing a copy of the variable in one of the general purpose processor registers (R31..R0). Specifically, the processor must reload the variable from SRAM whenever it is referenced.

```cpp
int pin = 13;
volatile int state = LOW;
```

- Add jumps in the IVT to ISR routine, configure External Interrupt Control Register A (EICRA), and enable local and global Interrupt Flag Bits.

```cpp
void setup()
{
  pinMode(pin, OUTPUT);
  attachInterrupt(0, blink, CHANGE);
}
```

---

3 Read ATmega328P External Interrupts to learn more about this example.
• Write Interrupt Service Routine (ISR)

```c
void blink()
{
    state = !state;
}
```

• To disable interrupts globally (clear the I bit in SREG) call the `noInterrupts()` function. To once again enable interrupts (set the I bit in SREG) call the `interrupts()` function.

• Again – Toggling interrupts ON and OFF is not recommended. For a discussion of when you may want to turn interrupts off, read Gammon Software Solutions forum – Why disable Interrupts?

---

4 Read ATmega328P External Interrupts to learn more about this example.