Interrupts in AVR Microcontrollers
(Chapter 10 of the text book)
Contents

• Interrupts ATmega32
• Using interrupts in C programming
Interrupts in AVR

Interrupts: A way o improve performance
How do interrupts work?
Interrupt vs. polling

• Polling:
  – Continuously check the flags to know when the expected event occurs, then go for the next event
  – Example: Timer time-out problem of the previous lecture:

```c
void T0Delay ( )
{
    TCNT0 = 0x20; //load TCNT0
    TCCR0 = 0x01; //Timer0, Normal mode, no prescaler
    while ((TIFR&0x1)==0); //wait for TF0 to roll over
    TCCR0 = 0;
    TIFR = 0x1; //clear TF0
}
```
Interrupt vs polling

• Polling: wasting time to check the devices continuously

• What if we are to generate two delays at the same time?
  – Example: Toggle bit PB.5 every 1s and PB.4 every 0.5s.

• What if there are some task to be done simultaneously with the timers?
  – Example: (1) read the contents of port A, process the data, and send them to port D continuously, (2) toggle bit PB.5 every 1s, and (3) PB.4 every 0.5s.
Interrupt vs polling

• Interrupts
  – A mechanism to work with peripheral devices
• No need for the processor to monitor the status of the devices and events
• Let the events notify the processor when they occur
  – By sending an interrupt signal to processor
Interrupt vs polling

- Interrupts:
  - Example: Copy the contents of port A to port D continuously and toggle bit PB.5 every 1s and PB.4 every 0.5s.
  - Solution:
    » Copying the contents of port A to port D as the main program
    » Get timers 0 and 1 to generate the delays
    » Define two interrupts for timers 0 and 1 to notify the processor when they finish counting
    » Upon an interrupt, stop the main program, service the timers and continue the main program
Interrupts

• Interrupting mechanism in all microprocessors and microcontrollers is almost the same:
  – Define the set of devices and events that can generate an interrupt
  – Write a function for each interrupt that will be executed when the corresponding interrupt is activated
    » The address of this function must be saved somewhere
  – Set a priority scheme among interrupts
  – A mechanism is needed to disable all or some interrupts
Interrupts

• ISR: Interrupt Service Routine
  – The function that is executed when an interrupt is enabled

• Interrupt Vector Table: a table that keeps the address of each ISR in the instruction memory
# Interrupt sources in AVR

<table>
<thead>
<tr>
<th>Address</th>
<th>ATtiny:AVR</th>
<th>Atmega32:AVR</th>
<th>Atmega16:AVR</th>
<th>Atmega8:AVR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td>0x000</td>
<td>0x000</td>
<td>0x000</td>
<td>0x000</td>
</tr>
<tr>
<td>0x001</td>
<td>0x002</td>
<td>0x002</td>
<td>0x001</td>
<td>0x001</td>
</tr>
<tr>
<td>...</td>
<td>0x004</td>
<td>0x004</td>
<td>0x002</td>
<td>0x002</td>
</tr>
<tr>
<td>...</td>
<td>0x006</td>
<td>0x024</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>0x008</td>
<td>0x006</td>
<td>0x003</td>
<td>Compare Match</td>
</tr>
<tr>
<td>...</td>
<td>0x00A</td>
<td>0x008</td>
<td>0x004</td>
<td>Overflow</td>
</tr>
<tr>
<td>0x00C</td>
<td>0x00A</td>
<td>0x005</td>
<td>input capture</td>
<td>Timer 1 on 6th bit</td>
</tr>
<tr>
<td>0x00E</td>
<td>0x00C</td>
<td>0x006</td>
<td>Compare Match A</td>
<td>Timer 1 on 5th bit</td>
</tr>
<tr>
<td>...</td>
<td>0x010</td>
<td>0x00E</td>
<td>0x007</td>
<td>Compare Match B</td>
</tr>
<tr>
<td>0x012</td>
<td>0x010</td>
<td>0x008</td>
<td>Overflow</td>
<td>Timer 1 on 3rd bit</td>
</tr>
<tr>
<td>0x016</td>
<td>0x012</td>
<td>0x009</td>
<td>Overflow</td>
<td>Timer 0 on 3rd bit</td>
</tr>
<tr>
<td>...</td>
<td>0x014</td>
<td>0x026</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>0x01A</td>
<td>0x016</td>
<td>0x00B</td>
<td>...</td>
</tr>
<tr>
<td>0x020</td>
<td>0x01C</td>
<td>0x00E</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>0x024</td>
<td>0x020</td>
<td>0x010</td>
<td>...</td>
</tr>
<tr>
<td>0x007</td>
<td>0x024</td>
<td>0x020</td>
<td>0x010</td>
<td>...</td>
</tr>
</tbody>
</table>

- **Reset**: Reset
- **INT0**: External interrupt 0
- **INT1**: External interrupt 1
- **INT2**: External interrupt 2
- **Compare Match**: Compare match
- **Overflow**: Overflow
- **input capture**: Input capture
- **Compare Match A**: Compare match A
- **Compare Match B**: Compare match B
- **Timer 1 on 6th bit**: Timer 1 on 6th bit
- **Timer 1 on 5th bit**: Timer 1 on 5th bit
- **Timer 1 on 4th bit**: Timer 1 on 4th bit
- **Timer 1 on 3rd bit**: Timer 1 on 3rd bit
- **Timer 0 on 3rd bit**: Timer 0 on 3rd bit
- **Port C**: Port C
- **Port D**: Port D
- **Port E**: Port E
- **Port F**: Port F
- **USART (RS422)**: USART (RS422)
- **ADC**: ADC
- **Analog**: Analog
External interrupts

- 3 pins of ATmega32
Interrupts in AVR

- Just 2-bytes for each interrupt service routine
- Too small to write the interrupt service routine
- Write the routine somewhere in the memory and put a code to jump to the address of the function in the 2-byte assigned to the ISR
Interrupts in AVR

• How is an interrupt serviced?
  1. Stop fetching the next instruction and save PC
  2. Go to Interrupt Vector Table to find the address of the ISR of the interrupting device
  3. Execute the function
  4. Resume normal execution by retrieving PC
Enabling Interrupts

- Interrupts can be enabled or disabled by programmer
  - Bit7 (I) in SREG (status register)
  - SREG keeps the processor status (remember the first lecture on AVR)
  - Disabled on reset (I=0)
Enabling Interrupts

- In addition to Bit7 (I) in SREG each interrupt should be enabled independently.
- The enable bit of each interrupt is in some register.
  - Example: TIMSK register to enable/disable timer interrupts
  - TIMSK = Timer Interrupt Mask
  - Mask, غیرفعال کردن
- To enable timer1 overflow interrupt:
  - Bit7 (I) in SREG ← 1
  - Bit 0 of TIMSK (TOIE0) ← 1
Interrupt priority

• What if two or more interrupts occur at the same time?
  – The interrupt with lower ISR address is prioritized (external int. 0 has the highest priority)

• When an interrupt is serviced, the I bit becomes automatically 0 to disable interrupts
  – Will be enabled when returning from the ISR
<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value 1</th>
<th>Value 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCIE2</td>
<td>Timer0 output compare match interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCIE0</td>
<td>Timer0 output compare match interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCIE0</td>
<td>Timer0 output compare match interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCIE0</td>
<td>Timer0 output compare match interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOIE1</td>
<td>Timer1 overflow interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOIE1</td>
<td>Timer1 overflow interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOIE1</td>
<td>Timer1 overflow interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCIE1B</td>
<td>Timer1 output compare B match interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCIE1B</td>
<td>Timer1 output compare B match interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCIE1B</td>
<td>Timer1 output compare B match interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCIE1A</td>
<td>Timer1 output compare A match interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCIE1A</td>
<td>Timer1 output compare A match interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCIE1A</td>
<td>Timer1 output compare A match interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TICIE1</td>
<td>Timer1 input capture interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TICIE1</td>
<td>Timer1 input capture interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TICIE1</td>
<td>Timer1 input capture interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOIE2</td>
<td>Timer2 overflow interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOIE2</td>
<td>Timer2 overflow interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOIE2</td>
<td>Timer2 overflow interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCIE2</td>
<td>Timer2 output compare match interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCIE2</td>
<td>Timer2 output compare match interrupt enable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCIE2</td>
<td>Timer2 output compare match interrupt enable</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Interrupt programming in C

• Enable interrupts
• Set the mask register (TIMSK for timers)
  – Example: TIMSK=0x01;
• Write the ISR function to specify what operation should be done when the interrupt occurs
  → Compiler dependent: different in different compilers!
Interrupt programming in C

- A way to use assembly instructions in C:
  
  \#asm("instruction")

- SEI: (Set I) an assembly instruction that enables interrupts (bit 7 of SREG=1)

- CLI: Clear I

- \#asm("sei"); enable interrupts in C

- No way to access SREG.7 in C

- Mazidi’s book uses a different compiler→ different instructions: sei() instead of \#asm("sei"), different ISR names

\[\text{On entering the timers ISR, the TOV0 bit is automatically cleared, no need to be cleared by software}\]
Interrupt programming in C

- In *codevision*, ISR is generated during project setup. Just fill the function body!

```c
#include <mega32.h>

// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
    // Place your code here
}

// Declare your global variables here
void main(void)
{
    TCCR0=0x00;
    TCNT0=0x00;
    OCR0=0x00;

    // Timer(s)/Counter(s) Interrupt(s) initialization
    TIMSK=0x01;

    // Global enable interrupts
    #asm("sei")

    while (1)
    {
        // Place your code here
    }
}
```
External Interrupts

- To allow external sources interrupt the microcontroller
- Can be masked by CIGR register
- GICR: General interrupt control register
External Interrupts- GICR

<table>
<thead>
<tr>
<th></th>
<th>INT1</th>
<th>INT0</th>
<th>INT2</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IVSEL</th>
<th>IVCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT0</td>
<td>External Interrupt Request 0 Enable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 0 Disables external interrupt 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 1 Enables external interrupt 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT1</td>
<td>External Interrupt Request 1 Enable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 0 Disables external interrupt 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 1 Enables external interrupt 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT2</td>
<td>External Interrupt Request 2 Enable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 0 Disables external interrupt 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 1 Enables external interrupt 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These bits, along with the I bit, must be set high for an interrupt to be responded to.
External interrupts

- Interrupts can be edge triggered or level triggered
- Edge trigger: activated when a change in signal level occurs
- Level trigger: activated when a signal has a specific value
- INT0 and INT1 can be programmed to be edge or level triggered
  - Low-level active by default
- INT 2 is only edge triggered
External interrupts

- A register called ISC (interrupt sense control) can set the interrupt type of INT0 and INT1

<table>
<thead>
<tr>
<th>ISC01</th>
<th>ISC00</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>The low level of INT0 generates an interrupt request.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Any logical change on INT0 generates an interrupt request.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>The falling edge of INT0 generates an interrupt request.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>The rising edge of INT0 generates an interrupt request.</td>
</tr>
</tbody>
</table>

ISC10 and ISC11 set the same setting for INT1
External Interrupts- C programming

Example 10-12 (C version of Example 10-5)

Assume that the INT0 pin is connected to a switch that is normally high. Write a program that toggles PORTC.3, whenever INT0 pin goes low. Use the external interrupt in level-triggered mode.

Solution:

```c
#include "avr/io.h"
#include "avr/interrupt.h"

int main ()
{
    DDRC = 1<<3;  // PC3 as an output
    PORTD = 1<<2; // pull-up activated
    GICR = (1<<INT0);  // enable external interrupt 0
    asm("sei");   // enable interrupts
    while (1);   // wait here
}

ISR (INT0_vect) // ISR for external interrupt 0
{
    PORTC ^= (1<<3);  // toggle PORTC.3
}
```