Embedded Systems

4- External Interrupts

Dr. Khalil Ismail Khalil Yousef
Assoc. Prof., Electrical Eng. Dept.
Faculty of Eng., Assiut University
kyousef@aun.edu.eg

Polling vs Interrupts

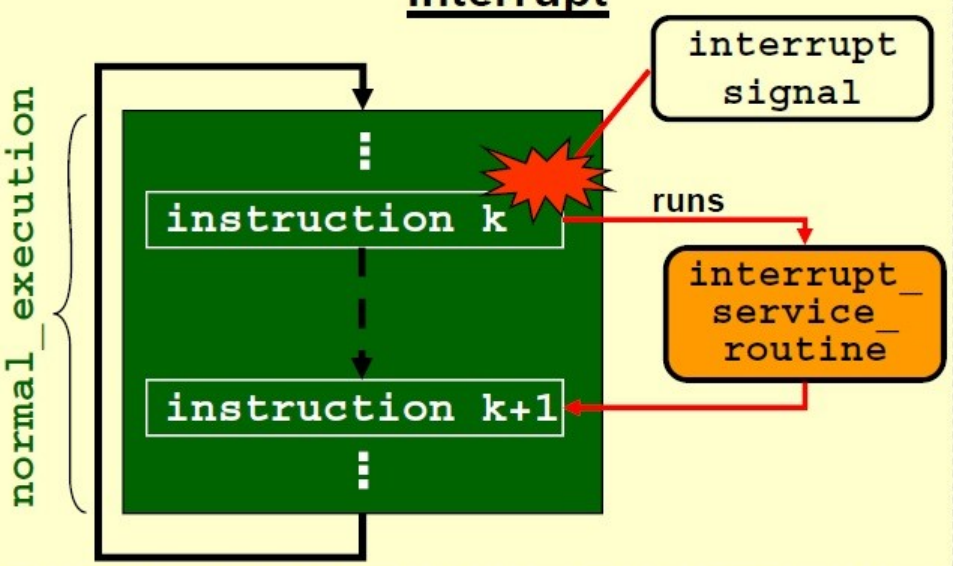
- Embedded systems often perform some tasks which are ***infrequent*** and possibly ***unpredictable***
- Regular tasks must be temporarily stopped to deal with the event
- Interrupts are the unusual events
- Interrupt handlers, or interrupt service routines, are programs which perform necessary tasks

Polling vs Interrupts

Polling

```
while (1){
    get_device_status;
    if (service_required){
        service_routine;
    }
    normal_execution;
}
```

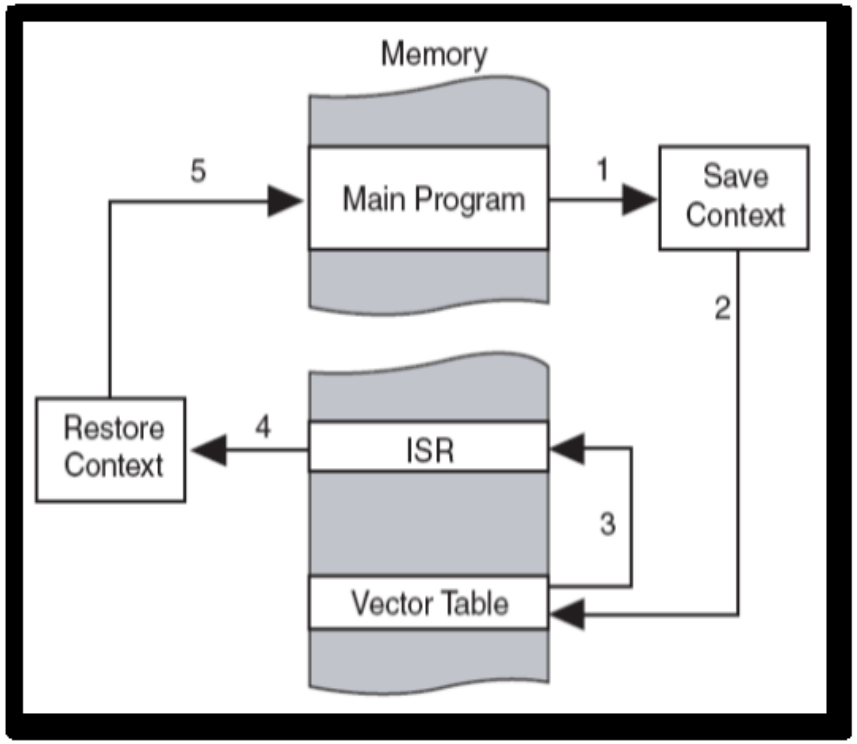
Interrupt



Interrupts

- Interrupt sources are presented in the Interrupt Vector Table
- Interrupts can be accepted after the finish of the currently executed instruction
- Bit 7 of SREG , the I-bit, is the Global Interrupt Enable (Clearing I-bit disables interrupts, setting I-bit enables them)
- I-bit is automatically cleared when an interrupt starts, set when interrupt is done (Interrupts are not interrupted)
- Use the SEI() and CLI() macros to set and clear the I-bit in C.
- Priority: decreases with the increasing of the interrupt number
- Maximum priority : Reset

Interrupt Execution Sequence



1. A device issues an interrupt
2. CPU finishes the current instruction
3. CPU acknowledges the interrupt
4. CPU saves its states and PC onto stack
5. CPU loads the address of ISR onto PC
6. CPU executes the ISR
7. CPU retrieves its states and PC from stack
8. Normal execution resumes

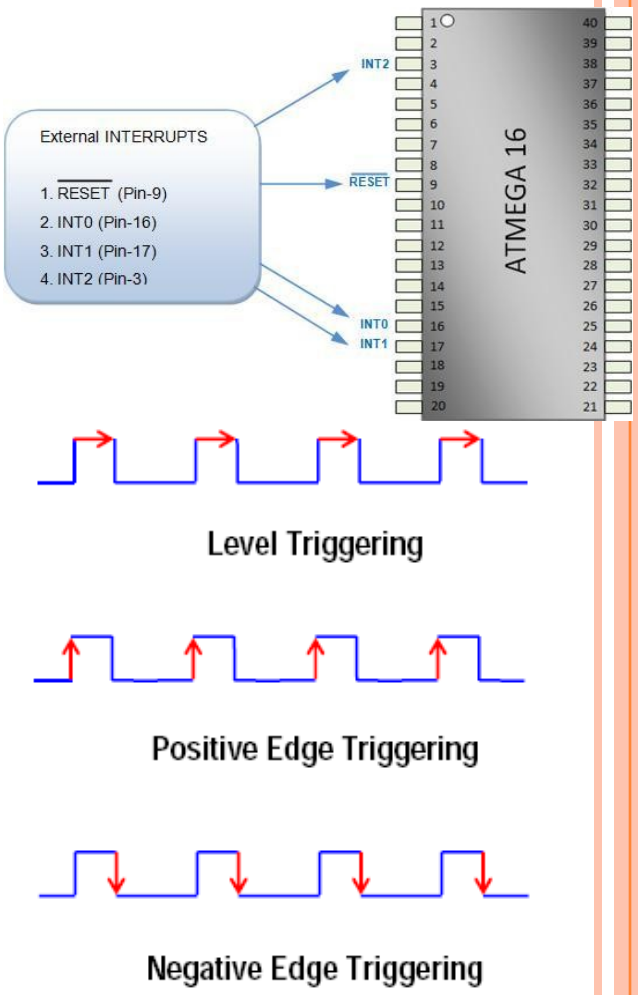
Interrupt Sources

Atmega16 has 21 different interrupts and each one has its own vector located in a predefined location at the start of the memory space.

- The ATmega16 interrupts:
- 1 reset interrupt
- 3 external interrupts
- 8 timer interrupts
- 3 serial port interrupts
- 1 ADC interrupt
- 1 analogue comparator interrupt
- 1 SPI interrupt
- 1 TWI interrupt
- 2 memory interrupts

External Interrupts

- 3 external interrupts are available
- INT0 at port PD2(pin 16) and INT1 at port PD3 (pin 17) and are both *Edge or Level triggered*, and INT2 at port PB2 (pin 3), which is only *Edge triggered*.
- Reset interrupt can be both externally (Reset input pin 9) and internally initiated.
- If enabled, the external interrupts will trigger even if the INT0 and INT1 pins are configured as outputs. This feature provides a way of generating a software interrupt .
- INT0, INT1 (level) and INT2 are detected asynchronously (no Clock needed and thus they can be used to get the µC from sleep modes.



Interrupt Sources

Vector No.	Program Address	Source Interrupt	Definition
1	$000	RESET External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR	Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	TIMER2 COMP	Timer-Counter2 Compare Match
5	$008	TIMER2 OVF	Timer-Counter2 Overflow
6	$00A	TIMER1 CAPT	Timer-Counter1 Capture Event
7	$00C	TIMER1 COMPA	Timer-Counter1 Compare Match A
8	$00E	TIMER1 COMPB	Timer-Counter1 Compare Match B
9	$010	TIMER1 OVF	Timer-Counter1 Overflow

Interrupt Sources

Vector No.	Program Address	Source Interrupt	Definition
10	$012	TIMER0 OVF	Timer-Counter0 Overflow
11	$014	SPI, STC	Serial Transfer Complete
12	$016	USART, RXC	USART, Rx Complete
13	$018	USART, UDRE	USART Data Register Empty
14	$01A	USART, TXC	USART, Tx Complete
15	$01C	ADC ADC	Conversion Complete
16	$01E	EE_RDY	EEPROM Ready
17	$020	ANA_COMP	Analog Comparator
18	$022	TWI	Two-wire Serial Interface
19	$024	INT2 External	Interrupt Request 2
20	$026	TIMER0 COMP	Timer-Counter0 Compare Match
21	$028	SPM_RDY	Store Program Memory Ready

Interrupt Vector Table

Vector No
An interrupt with a lower 'Vector No' has a higher priority.
E.g., INT0 has a higher priority than INT1 and INT2.

Program Address
The fixed memory location for a given interrupt handler.
E.g., in response to interrupt INT0, CPU runs instruction at $002.

Interrupt Vector Name
This is the interrupt name, to be used with C macro ISR().

External Interrupts

To enable INT0/INT1/INT2, the I-bit in the Status Register must be set (using the assembly Global Interrupt Enable instruction SEI) and the corresponding External Interrupt Request Enable bit in the General Interrupt Control Register (GICR)

GICR (General Interrupt Control Register)

Bit Number	7	6	5	4	3	2	1	0
GICR	INT1	INT0	INT2	—	—	—	IVSEL	IVCE
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

INT0/INT1 Triggering

The external interrupts INT0/INT1 can be triggered by a falling or rising edge or a low level.

Bit Number	7	6	5	4	3	2	1	0
MCUCR	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

By edge or toggle, pulses that last longer than one clock period will generate an interrupt.

If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

00 -> Low level

01 -> Logic Change

10 -> Falling Edge

11 -> Rising Edge

INT2 Triggering

- The external interrupts INT2 can only be triggered by a falling or rising edge.

ISC2 (Interrupt Sense Control of INT2):

ISC2 = 0 -> Falling Edge

ISC2 = 1 -> Rising Edge

Bit Number	7	6	5	4	3	2	1	0
MCUSCR	JTD	ISC2	—	JTRF	WDRF	BORF	EXTRF	PORF
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	See Bit Description				

MCU Control and Status Register

External Interrupts

When an event on the INT0/INT1/INT2 pin triggers an interrupt request, INT0-Flag/INT1-Flag/INT2-Flag (bit 6/7/5 of the General Interrupt Flag Register – GIFR) becomes set. The flag is cleared when the interrupt routine is executed.

Bit Number	7	6	5	4	3	2	1	0
GIFR	INTF1	INTF0	INTF2	–	–	–	–	–
Read/Write	R/W	R/W	R/W	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

GIFR (General Interrupt Flags Register)

Including Assembly Instructions in the Code

You can include assembly language anywhere in your program using the #asm and #endasm directives.

Example:

```
void delay(unsigned char i)
{
     while (i--)
     {
          /* Assembly language code sequence */
          #asm
          nop
          nop
          #endasm
      };
}
```

Inline assembly may also be used.

```
#asm("sei")        //set interrupts (enable)
#asm("cli")        //clear interrupts (disable)
```

Using Interrupts in C

The access to the AVR interrupt system is implemented with the interrupt keyword.

Examples:

```
interrupt [2] void external_int0(void)
{
      /* Place your code here */
 }


interrupt [10] void timer0_overflow(void)
{
      /* Place your code here */
}
```

[2] and **[10]** are the vector no. in the table above (slide 3)

Using Interrupts in C

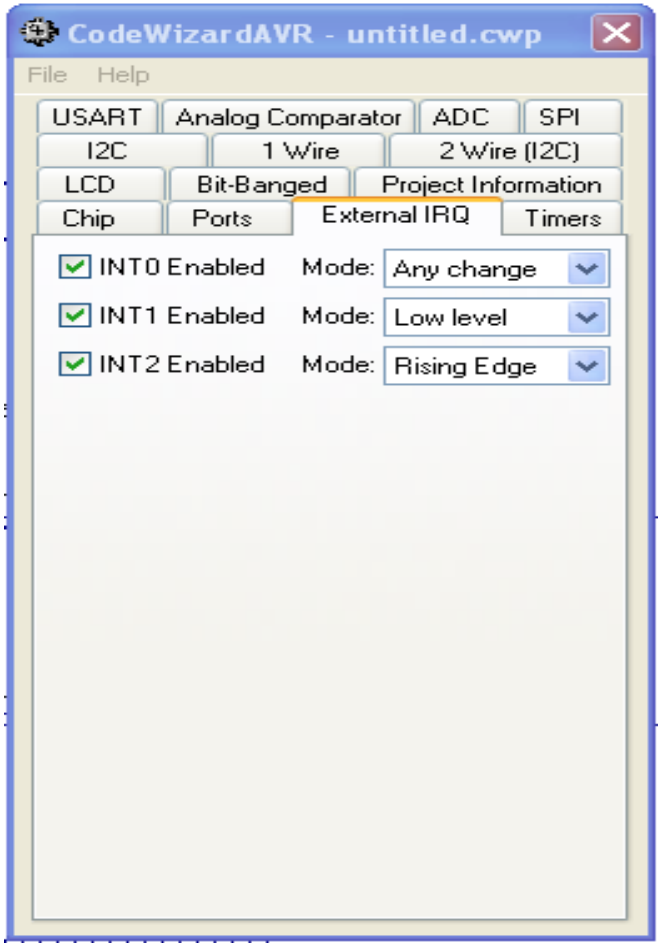
- Interrupt vector numbers start with 1.

- Note also that 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.

- Interrupt functions can't return a value nor have parameters.

- You must also set the corresponding bits in the I/O control registers to configure the interrupt system and enable the interrupts.

Using the CodeVision AVR Wizard by interrupts



NESTED INTERRUPTS

- The AVR hardware clears the global interrupt flag **I-bit** in **SREG** before entering an interrupt vector. Thus, normally interrupts will remain disabled inside the handler until the handler exits, where the **RETI** instruction (that is inserted by the compiler as part of the normal function for an interrupt handler) will eventually re-enable further interrupts.

- For that reason, interrupt handlers normally do not nest. For most interrupt handlers, this is the desired behavior, for some it is even required in order to prevent infinitely recursive interrupts (like UART interrupts, or level-triggered external interrupts).

- However, it might be desired to re-enable the global interrupt flag as early as possible in the interrupt handler, in order to not delay any other interrupt more than absolutely needed. This could be done using an **#asm("sei")** instruction right at the beginning of the interrupt handler.