ICT 5307: Embedded System Design

Lecture 6
Interrupts
(ADC Interrupt)

Professor S.M. Lutful Kabir

IICT, BUET

How Did We Check Whether the ADC Conversion Was Finished or Not?

```
while (1)
    {
     delay_us(10);
     // Start the AD conversion
     ADCSRA|=(1<<ADSC);
     // Wait for the AD conversion to complete
     while (ADIF==0);
     volt=ADCW/204.8;
     ftoa(volt,2,display);
     lcd_gotoxy(0,0);
     lcd_puts(display);
     delay_ms(200);
    }
```

This is called Polling Method

Polling versus Interrupt : A Real Life Scenario

- Polling is almost the equivalent of a real-life scenario where you hand-off to work to someone and then you keep bugging him all the time asking whether he is done.
- Interrupts on the other hand are quite different.
- When we hand-off some work to someone else we also ask him to let us know when he is done.
- This way we do not need to keep asking him all the time whether he is done. When the work is finished he will come back and notify us.

Another Real Life Scenario

- Consider you are reading a novel. Your elder brother calls you as he needs some help in finding a book. So you leave the task in hand by bookmarking the page you are reading and run to his help.
- Now as you are helping him find the book, your mom calls you as she needs you to remove some container from the top of the shelf.
- You prioritize your mother's request over your brother's. So you run to help her and climb up a chair to reach the rack.
- As you open the cupboard, your dad arrives and knocks the door. Now this can be your highest priority, so you jump of the chair, open the door, welcome dad.
- You go back climb up the chair, remove the container, and run back to help you brother. Then you resume reading the novel.
- Here you kept two tasks pending, to help someone with a higher priority. This is called NESTED INTERRUPT.

What is Interrupt in Case of a Microcontroller?

- **Interrupts** are basically events that require immediate attention by the microcontroller.
- When an interrupt event occurs the microcontroller pause its current task and attend to the interrupt by executing an **Interrupt Service Routine (ISR).**
- At the end of the ISR the microcontroller returns to the task it had paused and continue its normal operations.

Interrupt Service Routine (ISR)

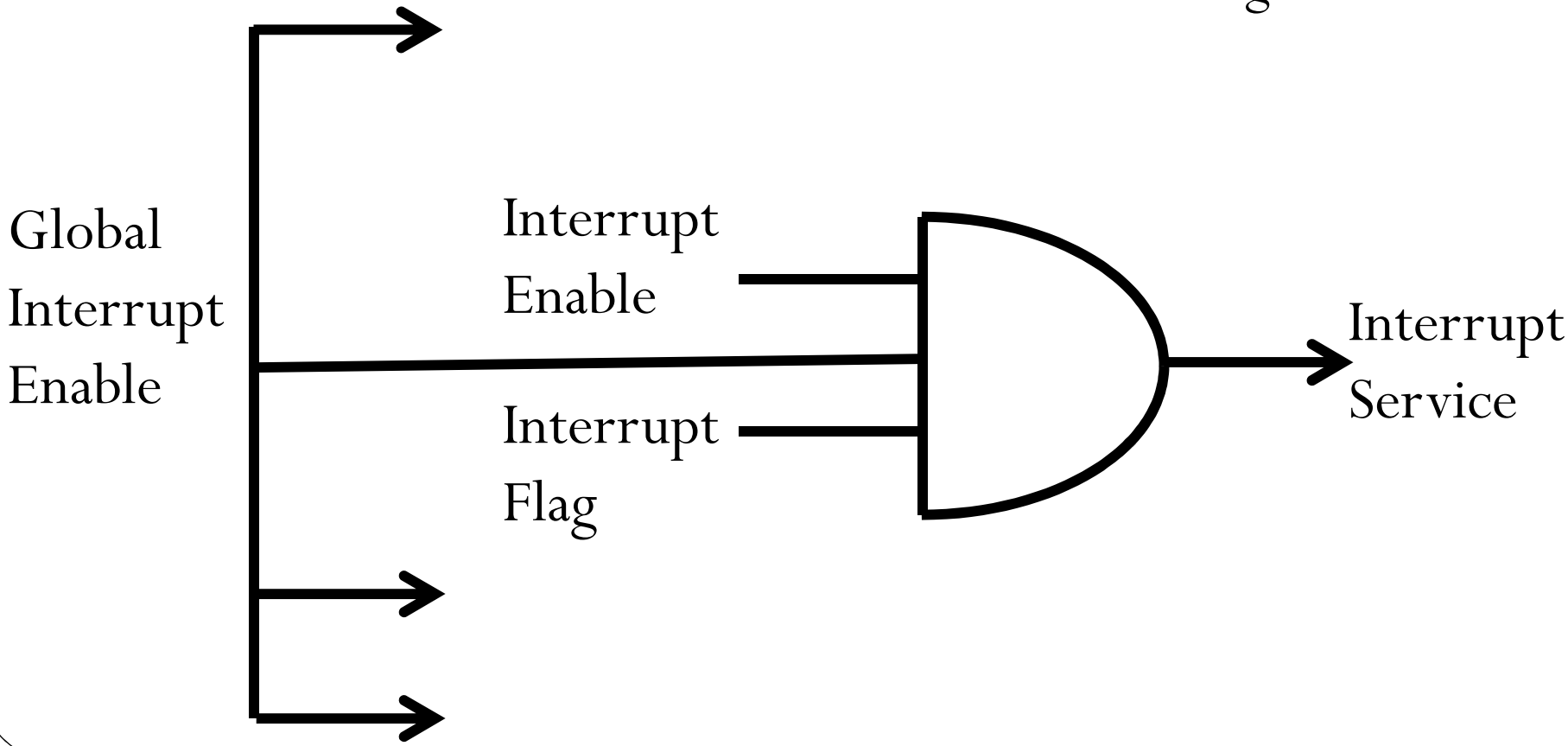
- An **Interrupt Service Routine (ISR)** or **Interrupt Handler** is a piece of code that is executed when an interrupt is triggered.
- Usually each enabled interrupt has its own ISR.
- ISR is also called Interrupt Handler.

Interrupt Flags and Enabled bits

- Each interrupt is associated with two (2) bits, an **Interrupt Flag Bit** and an **Interrupt Enabled Bit**. These bits are located in the I/O registers associated with the specific interrupt.
- The **interrupt flag** bit is set whenever the interrupt event occurs, whether or not the interrupt is enabled.
- The **interrupt enabled** bit is used to enable or disable a specific interrupt. Basically it tells the microcontroller whether or not it should respond to the interrupt if it is triggered.

Both bits should be high

- In summary basically both the **Interrupt Flag** and the **Interrupt Enabled** are required for an interrupt service routine to be executed as shown in the figure below.



Global Interrupt and Enable Bit

- Apart from the enabled bits for the specific interrupts the global interrupt enabled bit **MUST** be enabled for interrupts to be activated in the microcontoller.
- For the AVR 8-bits microcontroller this bit is located in the **Status I/O Register (SREG)**.

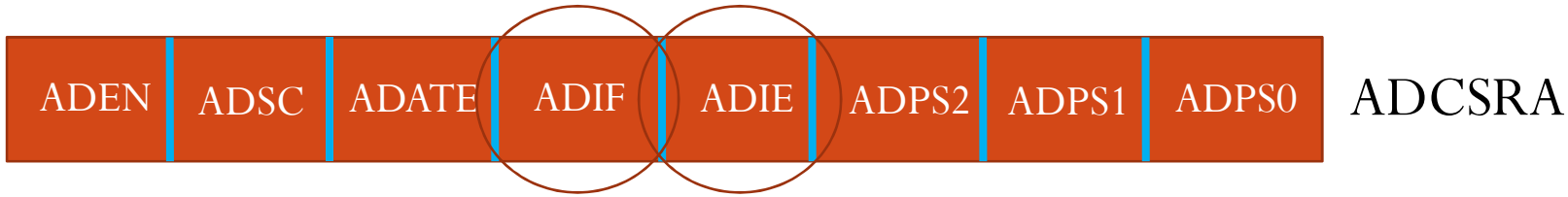
Status Register, SREG

- Upon RESET all interrupts are disabled (masked).
- It means that none will be responded by the microcontroller if they occur.
- So, the interrupts must be enabled by software in order for the microcontroller to respond to them.

Interrupt Vector Table

Sl	Interrupt	ROM Loc.	Sl	Interrupt	ROM Loc.
1	Reset	0000	12	T0 Overflow	0016
2	Ext. Int. 0	0002	13	SPI transfer	0018
3	Ext. Int. 1	0004	14	USART Receive	001A
4	Ext. Int. 2	0006	15	USAR Data Reg.	001C
5	T2 Compare	0008	16	USAR Transmit	001E
6	T2 Overflow	000A	17	ADC Conv. Comp	0020
7	T1 Inp. Capture	000C	18	EEPROM ready	0022
8	T1 Compare A	000E	19	Analog Comp.	0024
9	T1 Compare B	0010	20	I2C	0026
10	T1 Overflow	0012	21	Store Prog. Mem	0028
11	T0 Compare	0014			

ADC Interrupt



- After enabling Global Interrupt bit you have to enable ADIE bit in ADCSRA register
- Then if you want to start the conversion set ADSC bit of the same register.
- At the end of the conversion ADIF bit will automatically be set.
- You do not have to monitor that bit.
- As soon as the bit is set microcontroller will start executing the ADC service routine.
- At the end of ISR it will return back to the main program

ADCMUX Register

- REFS1:0 – Bit 7:6 – Reference Selection bits

  This bits select reference voltage for the ADC. 00- AREF pin, 01- AVCC pin and 10- Reserved and 11- Internal 2.56V

- ADLAR – Bit 5 – ADC Left adjust result

  1- left adjusted and 0- for right adjusted result

- MUX 4:0 – Bit 4:0 – Analogue Channel and Gain Selection bits

  Selects the gain of differential channels and selects which combination of inputs are connected to the ADC

An Experiment on ADC

- A variable voltage source will be connected to one of the channel of ADC. It is channel ADC0 (1st channel).
- The analogue voltage will be converted into digital form by the ADC.
- The voltage is read and after converting into decimal will be displayed on LCD.
- Reference voltage will be connected to AVCC.
- ADC frequency to be chosen to CLK/128. So, for 16 MHz crystal, ADC frequency will be 16 MHz/128=125 kHz
- We shall use ADC Interrupt for this experiment

The Code

```
#include <mega32.h>
#include <stdlib.h>
#include <delay.h>
#include <alcd.h>
// Declare your global variables
int i=0;
float volt=0;
unsigned char display[16];
```

Part-2 is on the next slide

```
// ADC interrupt service
   routine
interrupt [ADC_INT]
   void adc_isr(void)
{
   itoa(i,display);
   lcd_gotoxy(0,0);
   lcd_puts(display);
   volt=ADCW/204.8;
   ftoa(volt,2,display);
   lcd_gotoxy(0,1);
   lcd_puts(display);
}
```

```
void main(void)   {
lcd_init(16);
ADMUX=0b01000000;
ADCSRA=0b10001111;
// Global enable interrupts
#asm("sei")
while (1)      {
     delay_us(10);
     ADCSRA |= (1<<ADSC);
     for (i=0;i<1000;i++) ;
     delay_ms(3000);
     }
}
```

Thanks