CHW 469 : Embedded Systems

Instructor:

Dr. Ahmed Shalaby

http://bu.edu.eg/staff/ahmedshalaby14#

Interrupt
Chapter 10

The AVR microcontroller
and embedded
systems
using assembly and c

Contents

- Polling Vs. interrupt
- Interrupt unit
- Steps in executing an interrupt
- Edge trigger Vs. Level trigger in external interrupts
- Timer interrupt
- Interrupt priority
- Interrupt inside an interrupt
- Task switching and resource conflict
- C programming

Polling Vs. Interrupt

- Polling
  - Ties down the CPU

```
while (true)
{
    if(PIND.2 == 0)
        //do something;
}
```

- Interrupt
  - Efficient CPU use
  - Has priority
  - Can be masked

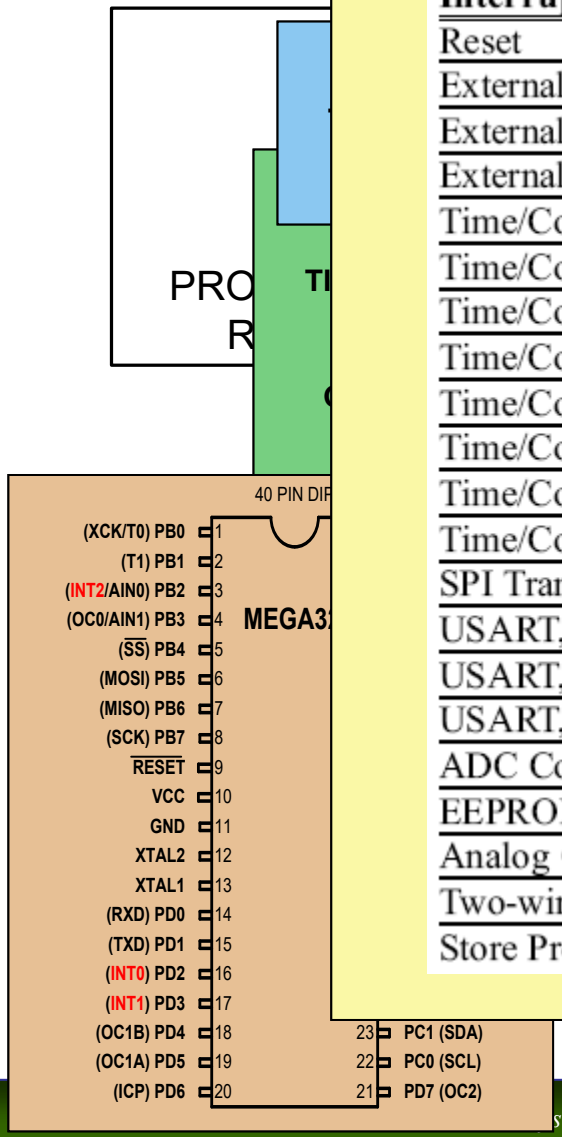
```
main( )
{
    Do your common task
}

whenever PIND.2 is 0 then
   do something
```

Interrupt unit

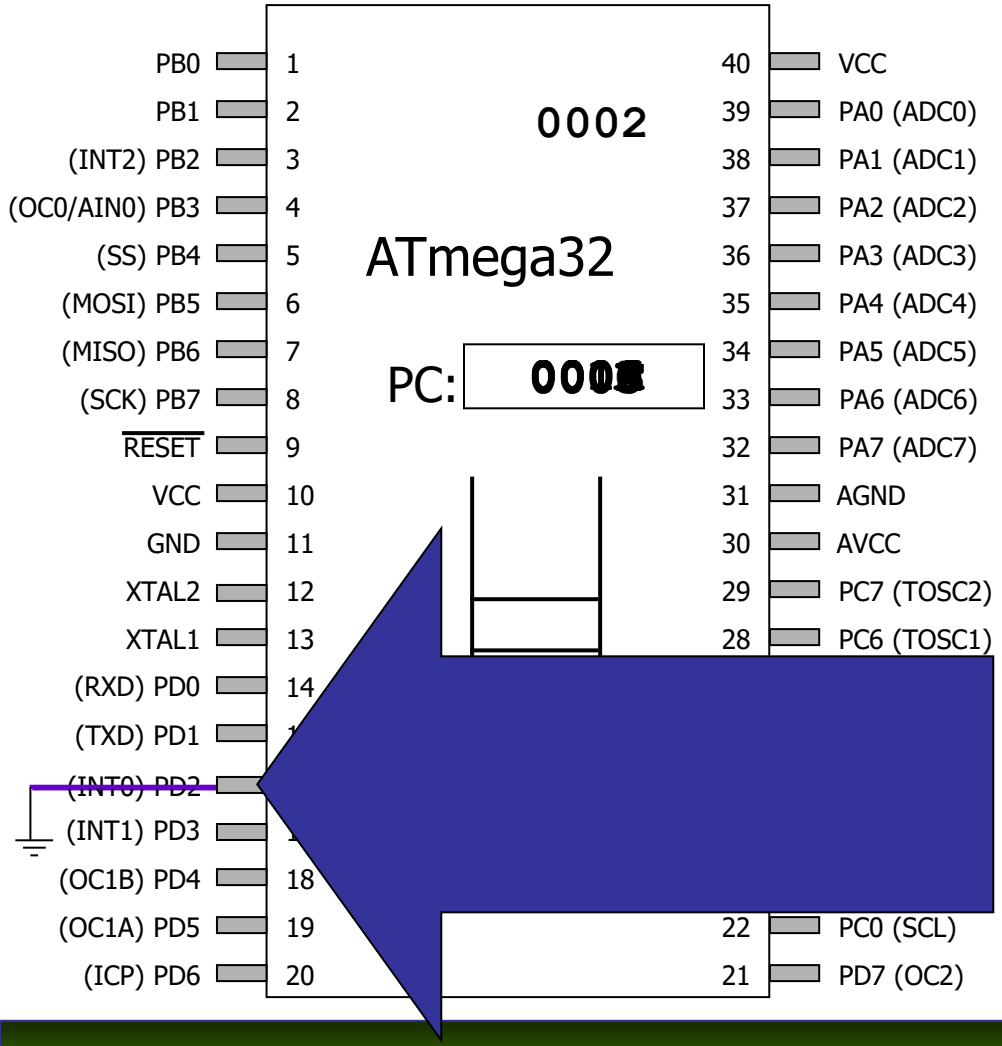
Table 10-1: Interrupt Vector Table for the Mega32

Interrupt	ROM Location (Hex)
Reset	0000
External Interrupt request 0	0002
External Interrupt request 1	0004
External Interrupt request 2	0006
Time/Counter2 Compare Match	0008
Time/Counter2 Overflow	000A
Time/Counter1 Capture Event	000C
Time/Counter1 Compare Match A	000E
Time/Counter1 Compare Match B	0010
Time/Counter1 Overflow	0012
Time/Counter0 Compare Match	0014
Time/Counter0 Overflow	0016
SPI Transfer complete	0018
USART, Receive complete	001A
USART, Data Register Empty	001C
USART, Transmit Complete	001E
ADC Conversion complete	0020
EEPROM ready	0022
Analog Comparator	0024
Two-wire Serial Interface	0026
Store Program Memory Ready	0028



PINS

Steps in executing an interrupt



Address	Code
	.INCLUDE "M32DEF.INC"
	.ORG 0 ;location for reset
0000	JMP MAIN
0002	.ORG 0x02 ;location for external INT0
0002	JMP EX0_ISR
0004	MAIN: LDI R20,HIGH(RAMEND)
0005	OUT SPH,R20
0006	LDI R20,LOW(RAMEND)
0007	OUT SPL,R20
0008	SBI DDRC,3 ;PC.3 = output
0009	SBI PORTD,2 ;pull-up activated
000A	LDI R20,1<<INT0 ;Enable INT0
000B	OUT GICR,R20
000C	SEI ;Set I (Enable Interrupts)
000D	LDI R30, 3
000E	LDI R31, 4
000F	ADD R30, R31
0010	HERE:JMP HERE
0012	EX0_ISR:IN R21,PORTC
0013	LDI R22,0x08
0014	EOR R21,R22
0015	OUT PORTC,R21
0016	RETI

Edge trigger Vs. Level trigger in external interrupts

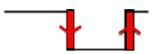
MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00

ISC01, ISC00 (Interrupt Sense Control bits) These b
external INT0 pin that activates the interrupt, as shown

```
LDI     R20,0x02    ;falling
OUT     MCUCR,R20
```

ISC01	ISC00		
0	0		The low level of INT0 generates an interrupt request.
0	1		Any logical change on INT0 generates an interrupt request.
1	0		The falling edge of INT0 generates an interrupt request.
1	1		The rising edge of INT0 generates an interrupt request.

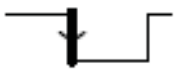
ISC11, ISC10 These bits define the level or edge that activates the INT1 pin.

ISC11	ISC10		Description
0	0		The low level of INT1 generates an interrupt request.
0	1		Any logical change on INT1 generates an interrupt request.
1	0		The falling edge of INT1 generates an interrupt request.
1	1		The rising edge of INT1 generates an interrupt request.

Edge trigger Vs. Level trigger (Cont.)

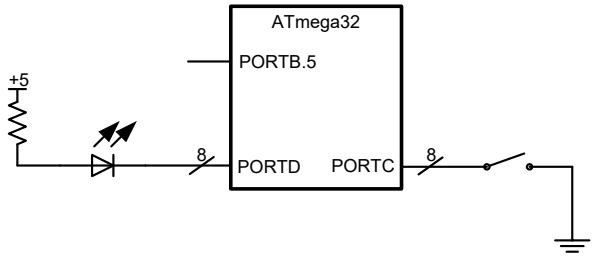
MCUCSR	JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF

ISC2 This bit defines whether the INT2 interrupt activates on the falling edge or the rising edge.

ISC2		Description
0		The falling edge of INT2 generates an interrupt request.
1		The rising edge of INT2 generates an interrupt request.

Using Timer0 overflow interrupt

- This program uses Timer0 to generate a square wave on pin PORTB.5, while at the same time data is being transferred from PORTC to PORTD.




```
1  ;Program 10-1
2  .INCLUDE "M32DEF.INC"
3  .ORG     0x0          ;location for reset
4           JMP          MAIN
5  .ORG     0x16         ;loc. for Timer0 over.
6           JMP          T0_OV_ISR
7  ;----main program for initialization
8  .ORG     0x100
9  MAIN:    LDI          R20,HIGH(RAMEND)
10          OUT          SPH,R20
11          LDI          R20,LOW(RAMEND)
12          OUT          SPL,R20
13          SBI          DDRB,5    ;output
14          LDI          R20,0
15          OUT          DDRC, R20
16          LDI          R20,0xFF
17          OUT          DDRD, R20
```

Timer int. init.

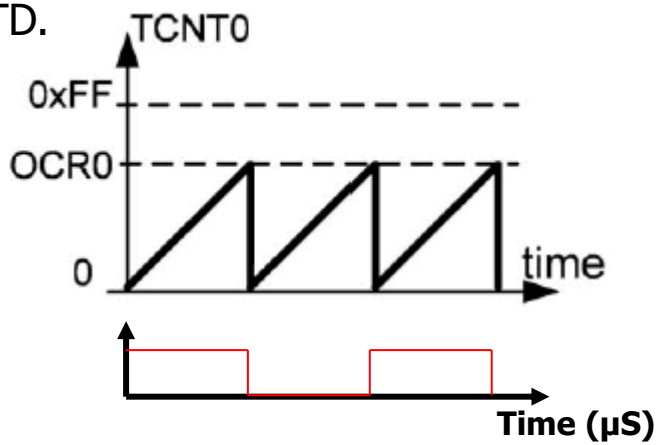
```
20          LDI          R20,(1<<TOIE0)
            OUT          TIMSK,R20
            SEI
```

Timer init.

```
21          LDI          R20,-32  ;value for 4µs
            OUT          TCNT0,R20
            LDI          R20,0x01
24          OUT          TCCR0,R20
25 HERE:    IN           R20,PINC
26          OUT          PORTD,R20
27          JMP          HERE
28 ;--------ISR for Timer 0
29 T0_OV_ISR:
30          IN           R16,PORTB
31          LDI          R17,0x20
32          EOR          R16,R17
33          OUT          PORTB,R16
34          RETI
```

Timer0 compare match interrupt

- using Timer0 and CTC mode generate a square wave on pin PORTB.5, while at the same time data is being transferred from PORTC to PORTD.




```
.INCLUDE "M32DEF.INC"
.ORG     0x0       ;location for reset
         JMP       MAIN
.ORG     0x14      ;location for Timer0 compare match
         JMP       T0_CM_ISR
;-main program for initialization and keeping CPU busy
.ORG     0x100
MAIN:    LDI       R20,HIGH(RAMEND)
         OUT       SPH,R20
         LDI       R20,LOW(RAMEND)
         OUT       SPL,R20
         LDI       R20,39
         OUT       OCR0,R20       ;OCR0 = 39
         LDI       R20,0x09
         OUT       TCCR0,R20      ;Start Timer0
         SBI       DDRB,5         ;PB5 as an output
         LDI       R20,(1<<OCIE0) ;Timer0 compare match
         OUT       TIMSK,R20
         SEI                      ;Set I
         LDI       R20,0x00
         OUT       DDRC,R20       ;make PORTC input
         LDI       R20,0xFF
         OUT       DDRD,R20       ;make PORTD output
```

Timer init.

Timer int. init.

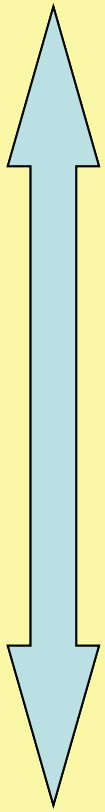
```
HERE:    IN        R20,PINC
         OUT       PORTD,R20
         JMP       HERE
;--------ISR for Timer 0
T0_CM_ISR:
         IN        R16,PORTB
         LDI       R17,0x20
         EOR       R16,R17
         OUT       PORTB,R16
         RETI
```

Interrupt priority

Table 10-1: Interrupt Vector Table for the Mega32

Interrupt	ROM Location (Hex)
Reset	0000
External Interrupt request 0	0002
External Interrupt request 1	0004
External Interrupt request 2	0006
Time/Counter2 Compare Match	0008
Time/Counter2 Overflow	000A
Time/Counter1 Capture Event	000C
Time/Counter1 Compare Match A	000E
Time/Counter1 Compare Match B	0010
Time/Counter1 Overflow	0012
Time/Counter0 Compare Match	0014
Time/Counter0 Overflow	0016
SPI Transfer complete	0018
USART, Receive complete	001A
USART, Data Register Empty	001C
USART, Transmit Complete	001E
ADC Conversion complete	0020
EEPROM ready	0022
Analog Comparator	0024
Two-wire Serial Interface	0026
Store Program Memory Ready	0028

Highest priority



Lowest priority

Interrupt inside an interrupt

- The I flag is cleared when the AVR begins to execute an ISR. So, interrupts are disabled.
- The I flag is set when RETI is executed.

Task switching and resource conflict

- Does the following program work?

```
1  .INCLUDE "M32DEF.INC"
2  .ORG     0x0          ;location for reset
3           JMP          MAIN
4  .ORG     0x14         ;Timer0 compare match
5           JMP          T0_CM_ISR
6  ;---------main program-----------------------
7  ----
8  .ORG     0x100
9  MAIN:    LDI          R20,HIGH(RAMEND)
10          OUT          SPH,R20
11          LDI          R20,LOW(RAMEND)
12          OUT          SPL,R20    ;set up stack
13          SBI          DDRB,5     ;PB5 =
14 output
15          LDI          R20,160
16          OUT          OCR0,R20
            LDI          R20,0x09
            OUT          TCCR0,R20
```

```
17          LDI          R20,(1<<OCIE0)
18          OUT          TIMSK,R20
19          SEI
20          LDI          R20,0xFF
21          OUT          DDRC,R20
22          OUT          DDRD,R20
23          LDI          R20, 0
24 HERE:    OUT          PORTC,R20
25          INC          R20
26          JMP          HERE
27 ;------------------------ISR for Timer0
28 T0_CM_ISR:
29          IN           R20,PIND
30          INC          R20
31          OUT          PORTD,R20
32          RETI
```

Solution 1: different registers

- Use different registers for different tasks.

```
1  .INCLUDE "M32DEF.INC"
2  .ORG     0x0          ;location for reset
3           JMP          MAIN
4  .ORG     0x14         ;Timer0 compare match
5           JMP          T0_CM_ISR
6  ;---------main program-----------------------
7  ----
8  .ORG     0x100
9  MAIN:    LDI          R20,HIGH(RAMEND)
10          OUT          SPH,R20
11          LDI          R20,LOW(RAMEND)
12          OUT          SPL,R20   ;set up stack
13          SBI          DDRB,5    ;PB5 =
14 output
15          LDI          R20,160
16          OUT          OCR0,R20
            LDI          R20,0x09
            OUT          TCCR0,R20
```

```
17          LDI          R20,(1<<OCIE0)
18          OUT          TIMSK,R20
19          SEI
20          LDI          R20,0xFF
21          OUT          DDRC,R20
22          OUT          DDRD,R20
23          LDI          R20, 0
24 HERE:    OUT          PORTC,R20
25          INC          R20
26          JMP          HERE
27 ;------------------------ISR for Timer0
28 T0_CM_ISR:
29          IN           R21,PIND
30          INC          R21
31          OUT          PORTD,R21
32          RETI
```

Solution 2: Context saving

- Save the contents of registers on the stack before execution of each task, and reload the registers at the end of the task.

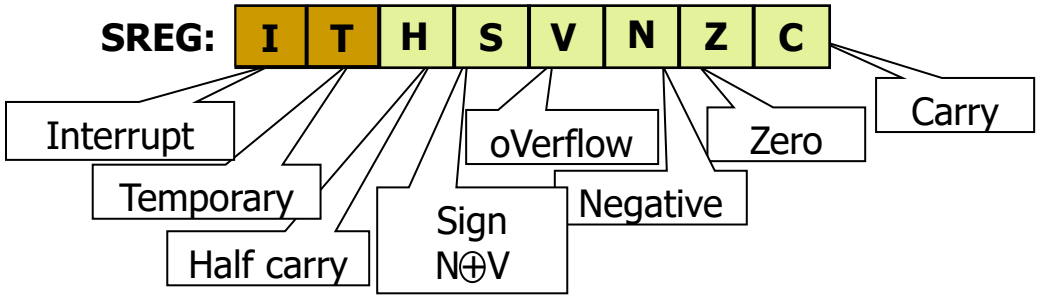
```
1  .INCLUDE "M32DEF.INC"
2  .ORG     0x0        ;location for reset
3           JMP        MAIN
4  .ORG     0x14       ;Timer0 compare match
5           JMP        T0_CM_ISR
6  ;---------main program-----------------------
7  ----
8  .ORG     0x100
9  MAIN:    LDI        R20,HIGH(RAMEND)
10          OUT        SPH,R20
11          LDI        R20,LOW(RAMEND)
12          OUT        SPL,R20    ;set up stack
13          SBI        DDRB,5     ;PB5 =
14 output
15          LDI        R20,160
16          OUT        OCR0,R20
17          LDI        R20,0x09
18          OUT        TCCR0,R20
19          LDI        R20,(1<<OCIE0)
20          OUT        TIMSK,R20
21          SEI
22          LDI        R20,0xFF
23          OUT        DDRC,R20
24          OUT        DDRD,R20
25          LDI        R20, 0
26 HERE:    OUT        PORTC,R20
27          INC        R20
28          JMP        HERE
29 ;------------------------ISR for Timer0
30 T0_CM_ISR:
31          PUSH       R20        ;save R20
32          IN         R20,PIND
33          INC        R20
34          OUT        PORTD,R20
35          POP        R20        ;restore R20
36          RETI
```

Saving SREG

- We should save SREG, when we change flags in the ISR.

```
        PUSH    R20
        IN      R20,SREG
        PUSH    R20
        ...
        POP     R20
        OUT     SREG,R20
        POP     R20
```

Status Register



C programming

- Using Timer0 generate a square wave on pin PORTB.5, while at the same time transferring data from PORTC to PORTD.

```
#include "avr/io.h"
#include "avr/interru
int main ()
{
        DDRB |= 0x20;
        TCNT0 = -32;
        TCCR0 = 0x01;
        TIMSK = (1<<TO
        sei ();
        DDRC = 0x00;
        DDRD = 0xFF;
        while (1)
          PORTD = PINC
}
ISR (TIMER0_OVF_vect)
{
        TCNT0 = -32;
        PORTB ^= 0x20;
}
```

Table 10-3: Interrupt Vector Name for the ATmega32/ATmega16 in WinAVR

Interrupt	Vector Name in WinAVR
External Interrupt request 0	INT0_vect
External Interrupt request 1	INT1_vect
External Interrupt request 2	INT2_vect
Time/Counter2 Compare Match	TIMER2_COMP_vect
Time/Counter2 Overflow	TIMER2_OVF_vect
Time/Counter1 Capture Event	TIMER1_CAPT_vect
Time/Counter1 Compare Match A	TIMER1_COMPA_vect
Time/Counter1 Compare Match B	TIMER1_COMPB_vect
Time/Counter1 Overflow	TIMER1_OVF_vect
Time/Counter0 Compare Match	TIMER0_COMP_vect
Time/Counter0 Overflow	TIMER0_OVF_vect
SPI Transfer complete	SPI_STC_vect
USART, Receive complete	USART0_RX_vect
USART, Data Register Empty	USART0_UDRE_vect
USART, Transmit Complete	USART0_TX_vect
ADC Conversion complete	ADC_vect
EEPROM ready	EE_RDY_vect
Analog Comparator	ANA_COMP_vect
Two-wire Serial Interface	TWI_vect
Store Program Memory Ready	SPM_RDY_vect

C programming Example 2

- Using Timer1 and CTC mode write a program that toggles pin PORTB.5 every second, while at the same time transferring data from PORTC to PORTD. Assume XTAL = 8 MHz.

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

int main ()
{
        DDRB |= 0x20;           //make DDRB.5 output
        OCR0 = 40;
        TCCR0 = 0x09;           //CTC mode, internal clk, no prescaler
        TIMSK = (1<<OCIE0);     //enable Timer0 compare match int.
        sei ();                 //enable interrupts
        DDRC = 0x00;            //make PORTC input
        DDRD = 0xFF;            //make PORTD output
        while (1)               //wait here
          PORTD = PINC;
}

ISR (TIMER0_COMP_vect)          //ISR for Timer0 compare match
{
        PORTB ^= 0x20;          //toggle PORTB.5
}
```

C programming Example 3

- 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.

```
#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
        sei ();                 //enable interrupts

        while (1);              //wait here
}

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

ADC and DAC Programming in AVR

The AVR microcontroller and embedded systems

using assembly and c

Topics

- What is ADC and why do we need it?
- ADC major characteristics
- ADC in AVR
- Hardware Consideration
- AVR ADC Programming
  - ADCH and ADCL
  - ADMUX
  - ADCSRA
- DAC
- Signal conditioning and sensors

What is ADC? Do we need it?

- Analogue vs. digital signal

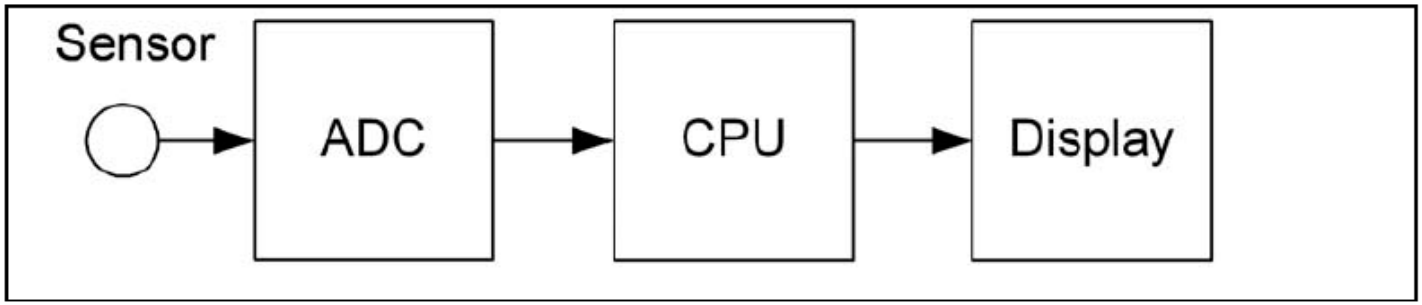


Figure 13-1. Microcontroller Connection to Sensor via ADC

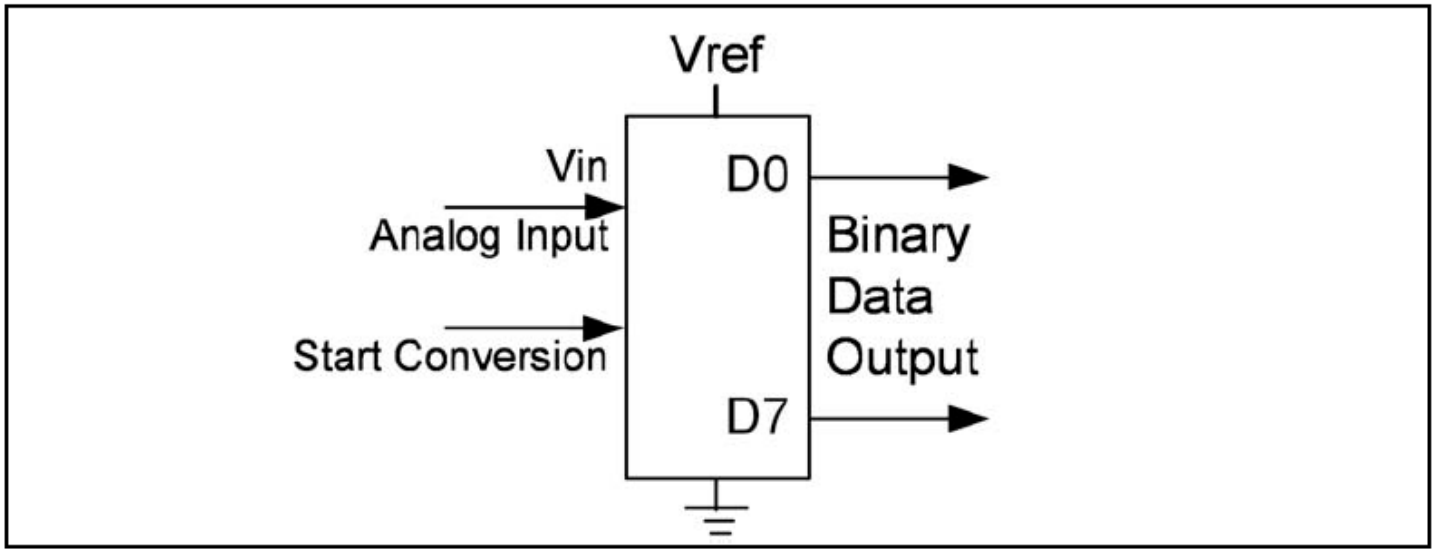
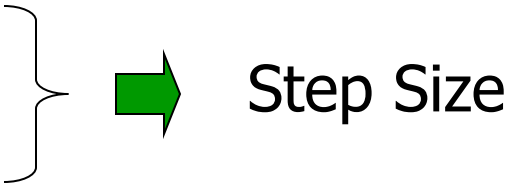


Figure 13-2. An 8-bit ADC Block Diagram

ADC major characteristics

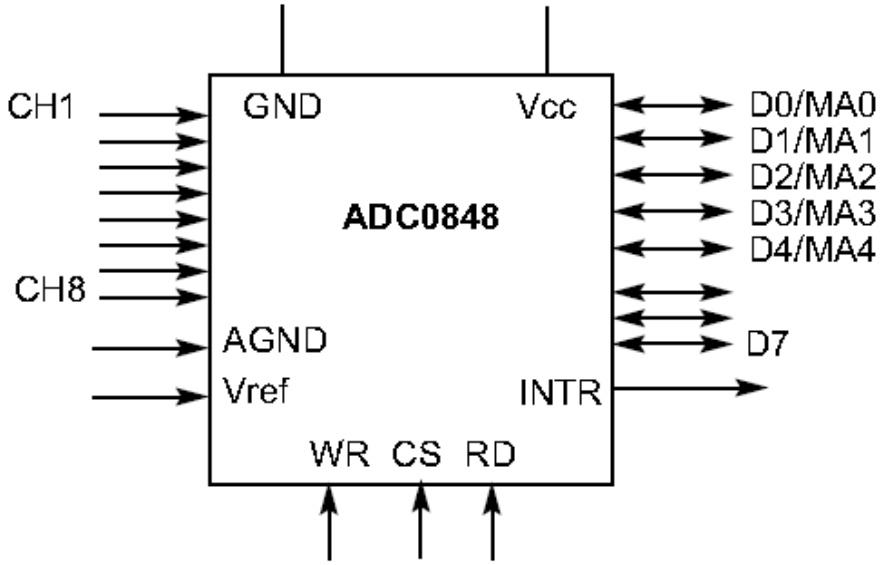
- Conversion Time
- Resolution
- Vref
- Parallel vs. serial
- Input channels



Some of ADC Signals

- Dout = Vin / Step size

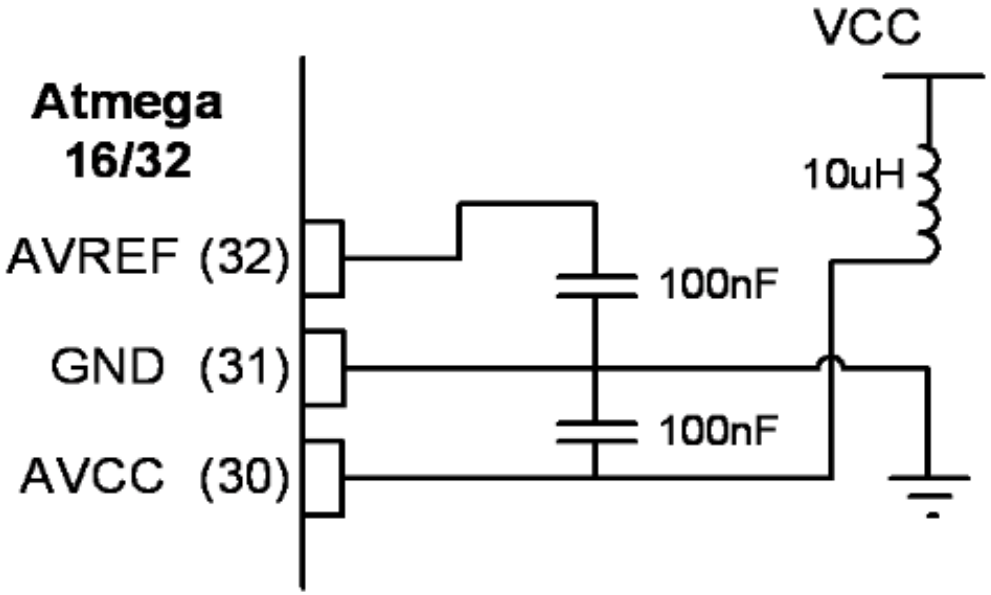
  (in 8 bit ADC and Vref = 2.56 what is D out for 20mV input voltage?)
- Start of Conversion
- Channel Selector



ADC in AVR

- Atmega 16/32 have internal ADC
  - 8 analogue input channel
  - 7 differential input channel
  - 2 differential input channel with 10x or 200x gain
  - 3 source of Vref
  - Internal 2.56V Vref generator

Hardware Consideration



ADCH and ADCL Data registers

- ADCH:ADCL store the results of conversion.
- The 10 bit result can be right or left justified:

ADLAR = 0

ADCH							
-	-	-	-	-	-	ADC9	ADC8

ADCL							
ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0

ADLAR =1

ADCH							
ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2

ADCL							
ADC1	ADC0	-	-	-	-	-	-

ADMUX

Table 13-4: V_{ref} source selection table

REFS1	REFS0	V Reference
0	0	AREF pin
0	1	AVCC pin
1	0	Reserved
1	1	Internal 2.56 V

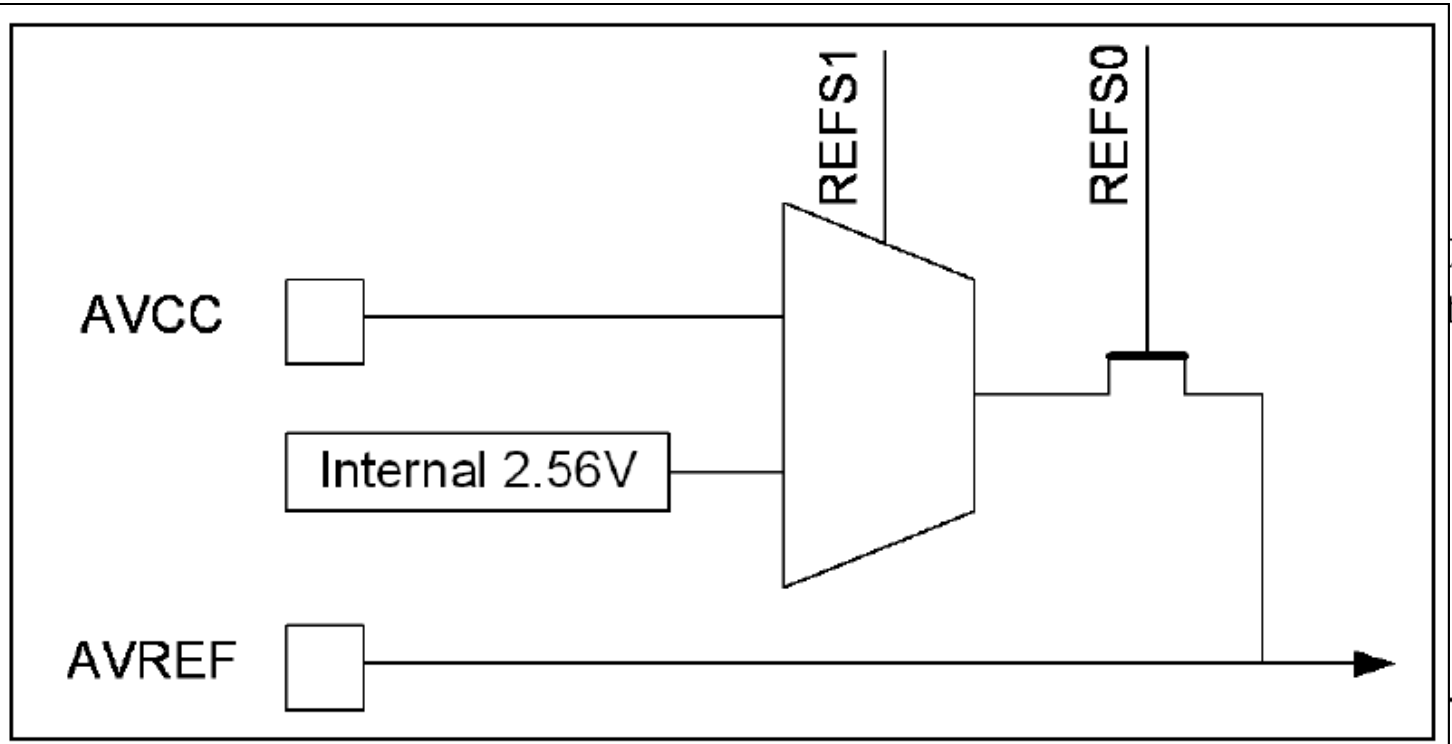


Figure 13-7: ADC Reference Source Selection

ADC input sel

Table 13-6: Single Ended Channels

MUX4..0	+ Differential Input	- Differential Input	Gain
01000	ADC0	ADC0	10x
01001	ADC1	ADC0	10x
01010	ADC0	ADC0	200x
01011	ADC1	ADC0	200x
01100	ADC2	ADC2	10x
01101	ADC3	ADC2	10x
01110	ADC2	ADC2	200x
01111	ADC3	ADC2	200x
10000	ADC0	ADC1	1x
10001	ADC1	ADC1	1x
10010	ADC2	ADC1	1x
10011	ADC3	ADC1	1x
10100	ADC4	ADC1	1x
10101	ADC5	ADC1	1x
10110	ADC6	ADC1	1x
10111	ADC7	ADC1	1x
11000	ADC0	ADC2	1x
11001	ADC1	ADC2	1x
11010	ADC2	ADC2	1x
11011	ADC3	ADC2	1x
11100	ADC4	ADC2	1x
11101	ADC5	ADC2	1x

ADCSA

ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADEN- Bit7 ADC Enable
This bit enables or disables the ADC. Writing this bit to one will enable and writing this bit to zero will disable the ADC even while a conversion is in progress.

ADSC- Bit6 ADC Start Conversion
To start each coversion you have to write this bit to one.

ADATE- Bit5 ADC Auto Trigger Enable
Auto Triggering of the ADC is enabled when you write this bit to one.

ADIF- Bit4 ADC Interrupt Flag
This bit is set when an ADC conversion completes and the Data Registers are updated

ADIE- Bit3 ADC Interrupt Enable
Writing this bit to one enables the ADC Conversion Complete Interrupt.

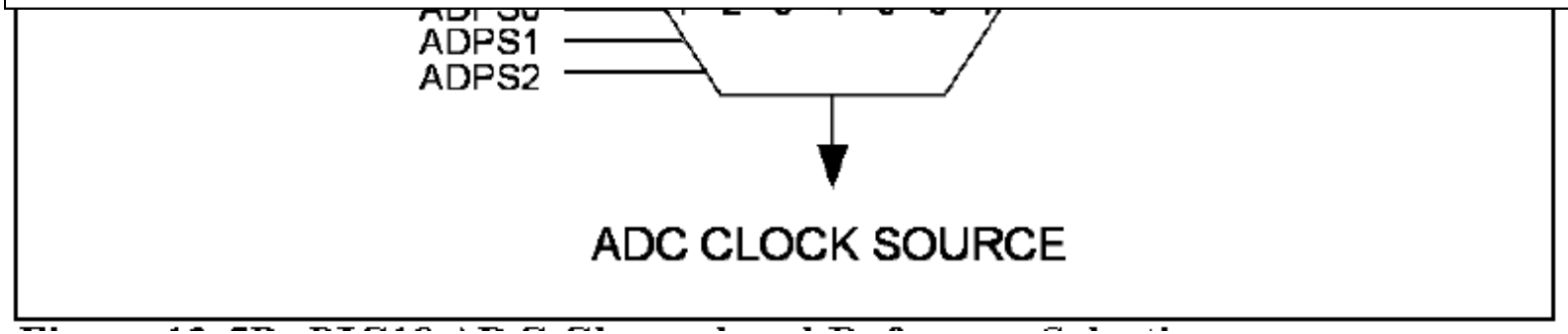
ADPS2:0- Bit2:0 ADC Prescaler Select Bits
These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

ADC Prescaler

- PreScaler Bits let us change the clock frequency of ADC
- The frequency of ADC should not be more than 200 KHz
- Conversion time is longer in the first conversion

Table 13-3: V_{ref} source selection table

Condition	Sample and Hold Time (Cycles)	Conversion Time (Cycles)
First Conversion	14.5	25
Normal Conversion, Single ended	1.5	13
Normal Conversion, Differential	2	13.5
Auto trigger conversion	1.5 / 2.5	13/14



Programming ADC

1. Make the pin for the selected ADC channel an input pin.
2. Turn on the ADC module of the AVR because it is disabled upon power-on reset to save power.
3. Select the conversion speed. We use registers ADPS2:0 to select the conversion speed.
4. Select voltage reference and A/C input channels. We use REFS0 amd REFS1 bits in ADMUX register to select voltage reference and MUX4:0 bits in ADMUX to select ADC input channel.
5. Activate the start conversion bit by writing ADSC bit of ADCSRA to one.
6. Wait for the conversion to be completed by polling the ADIF bit in ADCSRA register.
7. After the ADIF bit has gone one read the ADCL and ADCH registers to get the digital data output. Note that you have to read ADCL before ADCH otherwise the result may not be valid.
8. If you want to read the selected channel again go back to step 5.
9. If you want to select another Vref source or input channel go back to step 4.

Programming ADC - Polling

Program 13-1C is the C version of the ADC conversion for Program 13-1.

```
#include <avr/io.h>          //standard AVR header
int main (void)
{
   DDRB = 0xFF;              //make Port B an output
   DDRD = 0xFF;              //make Port D an output
   DDRA = 0;                 //make Port A an input for ADC input
   ADCSRA= 0x87;             //make ADC enable and select ck/128
   ADMUX= 0xC0;              //2.56V Vref, ADC0 single ended input
                             //data will be right-justified
   while (1){
      ADCSRA|=(1<<ADSC);     //start conversion
      while((ADCSRA&(1<<ADIF))==0);//wait for conversion to finish
      PORTD = ADCL;          //give the low byte to PORTD
      PORTB = ADCH;          //give the high byte to PORTB
   }
   return 0;
}
```

Program 13-1C: Reading ADC Using Polling Method in C

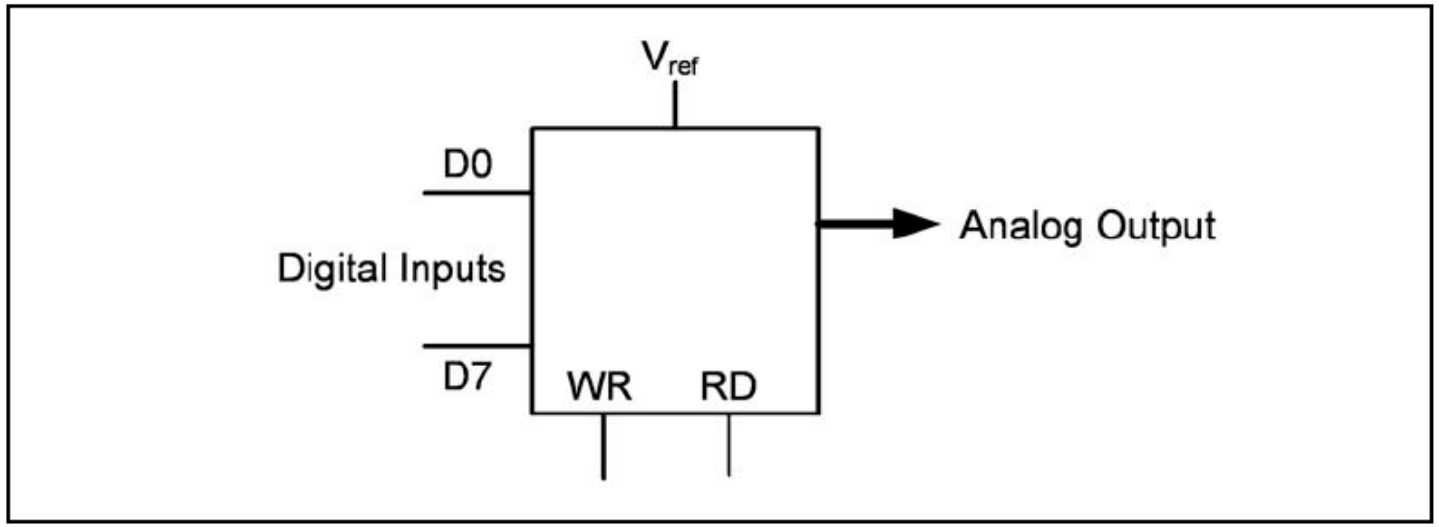
Programming ADC - Interrupts

```
#include <avr\io.h>
#include <avr\interrupt.h>
ISR(ADC_vect){
   PORTD = ADCL;              //give the low byte to PORTD
   PORTB = ADCH;              //give the high byte to PORTB
   ADCSRA|=(1<<ADSC);         //start conversion
}
int main (void){
   DDRB = 0xFF;               //make Port B an output
   DDRD = 0xFF;               //make Port D an output
   DDRA = 0;                  //make Port A an input for ADC input
   sei();                     //enable interrupts
   ADCSRA= 0x8F;              //enable and interrupt select ck/128
   ADMUX= 0xC0;               //2.56V Vref and ADC0 single-ended
                              //input right-justified data
   ADCSRA|=(1<<ADSC);         //start conversion
   while (1);                 //wait forever
   return 0;
}
```

Program 13-2C: Reading ADC Using Interrupts in C

DAC

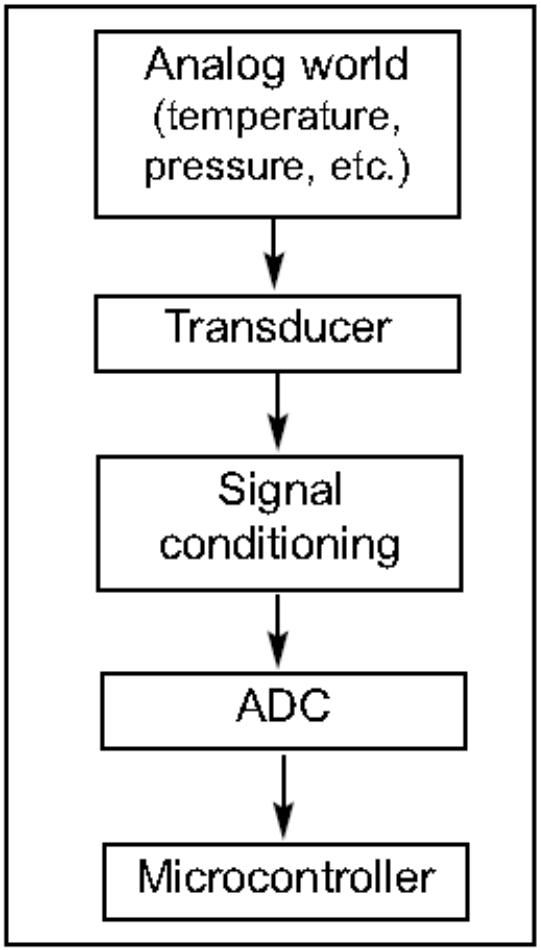
- What is DAC ?
- How to connect an DAC to AVR?



$$I_{out} - I_{ref} \left(\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256} \right)$$

Signal conditioning

- Thermo couple provides temp in form of uV
- PT100 provides resistance
- Some humidity s e result in form of Capacitance
- -> We have to c nals to Voltage to convert it by AD
  - This job is ca ning



Sensor Interfacing