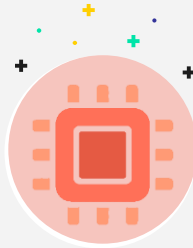




1403/03/07



Microcontrollers

Lecture 8:

Analog to Digital Converter (ADC)

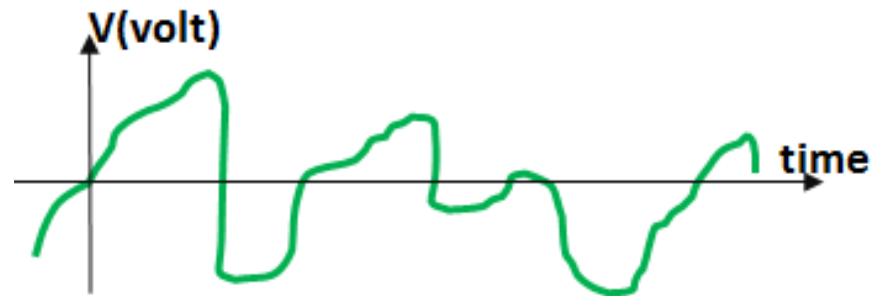
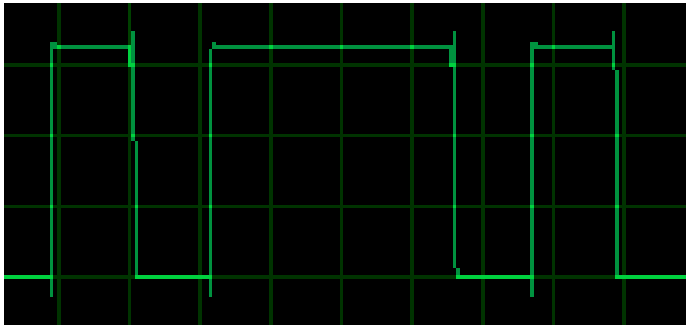
Digital to Analog Converter (DAC)

By: M.Razeghizadeh

[Start now!](#)



Analog vs. Digital Signals

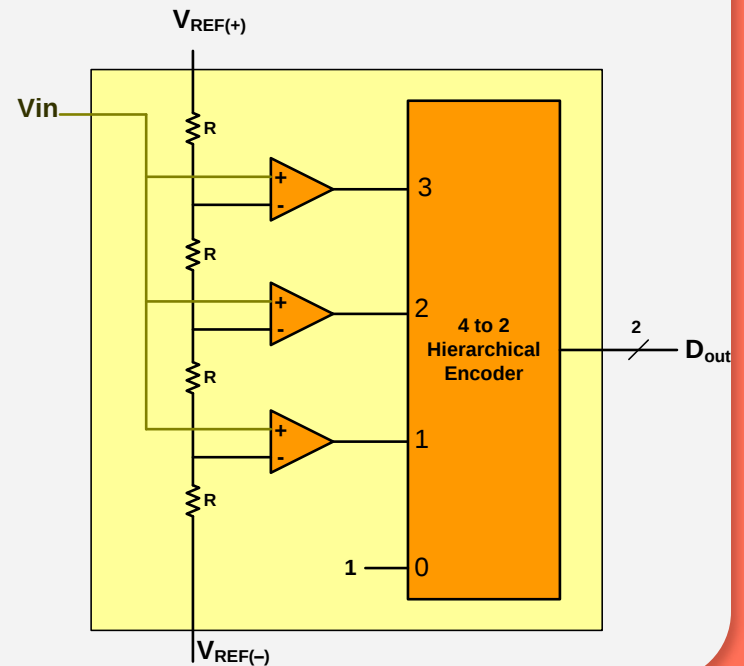
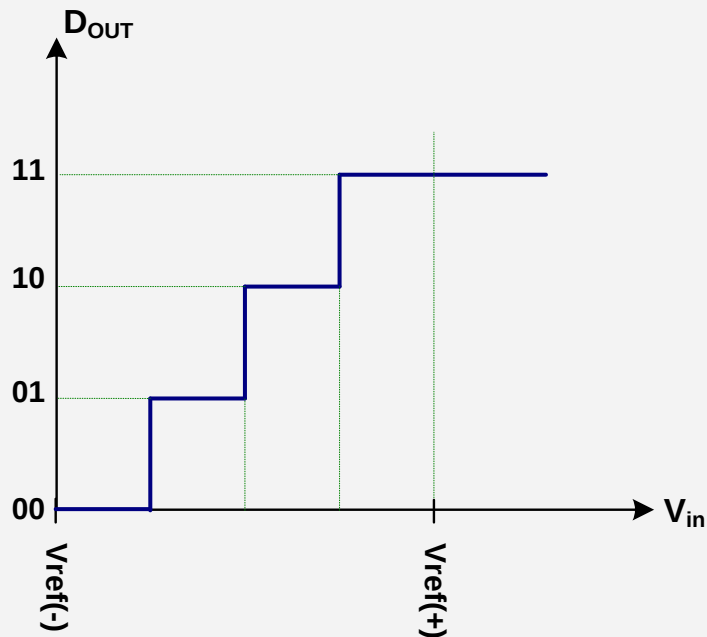
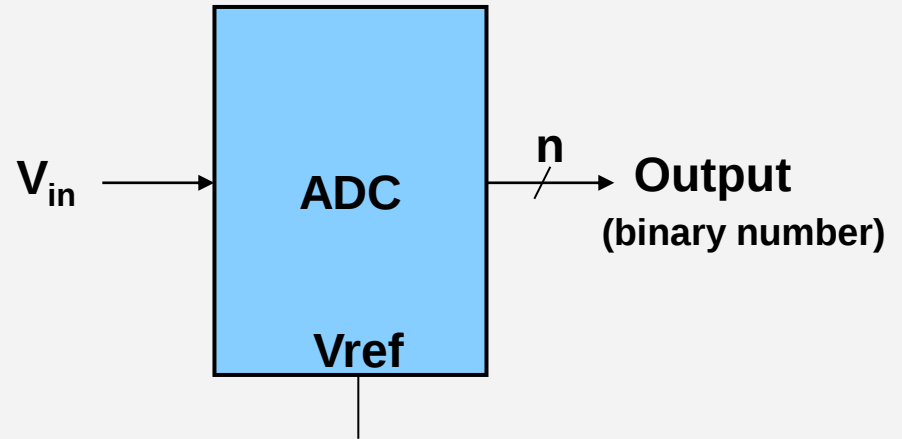




ADC

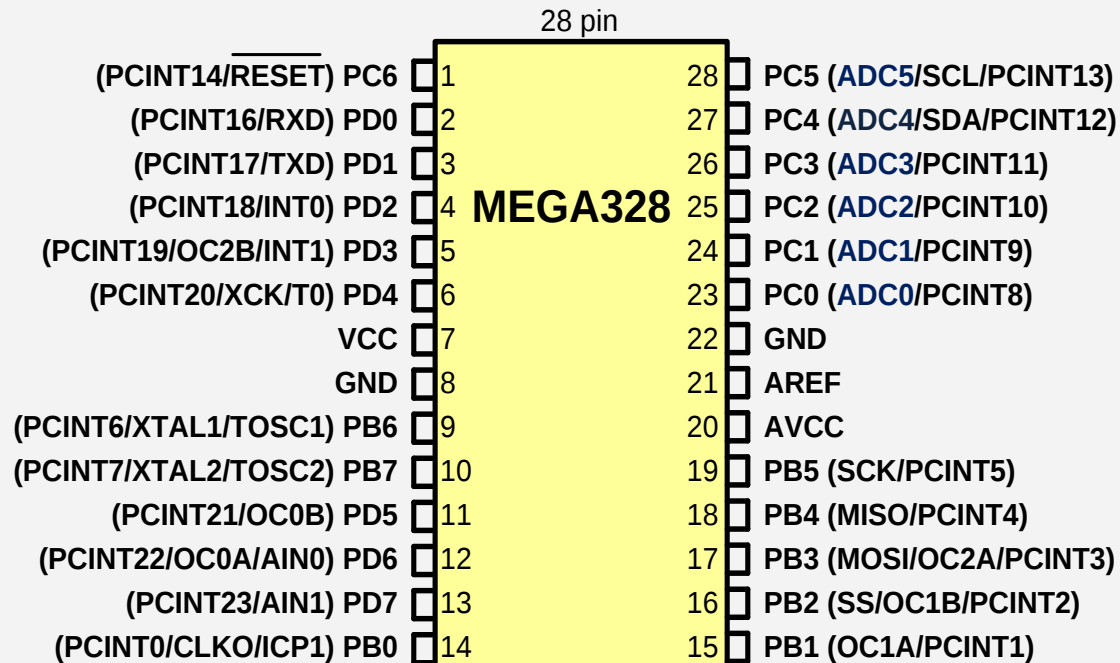
$$\text{stepSize} = \frac{V_{\text{ref}}}{\text{numofsteps}}$$

$$\text{output} = \left\lfloor \frac{V_{\text{in}}}{\text{stepSize}} \right\rfloor$$



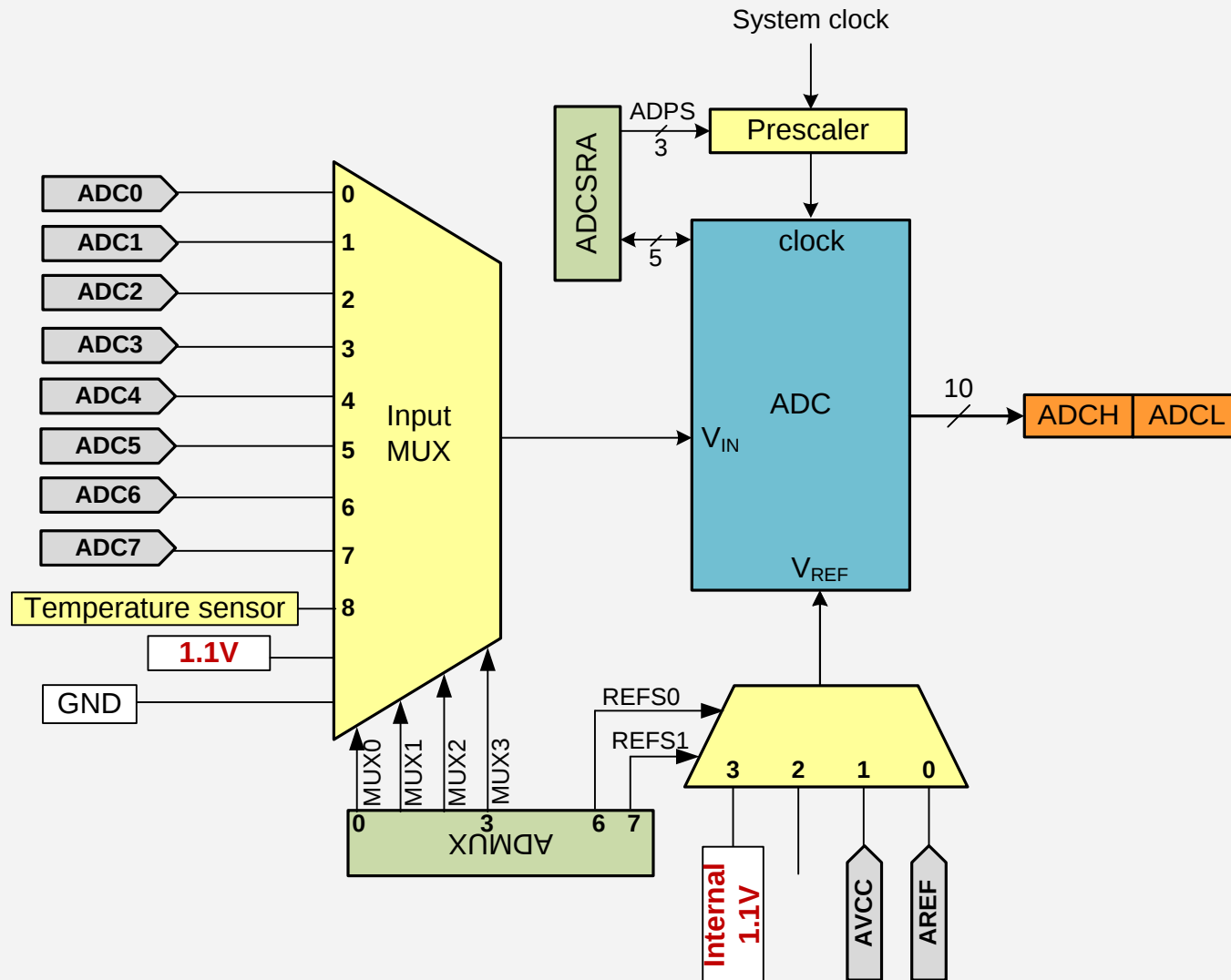


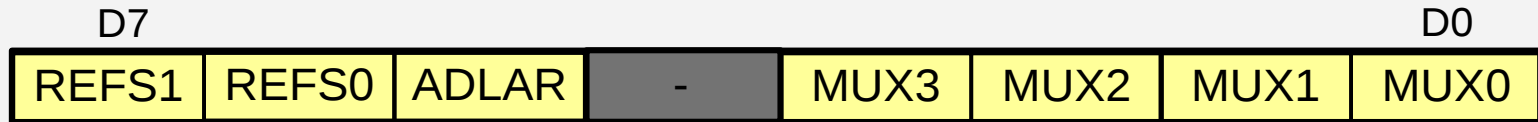
ADC in AVR





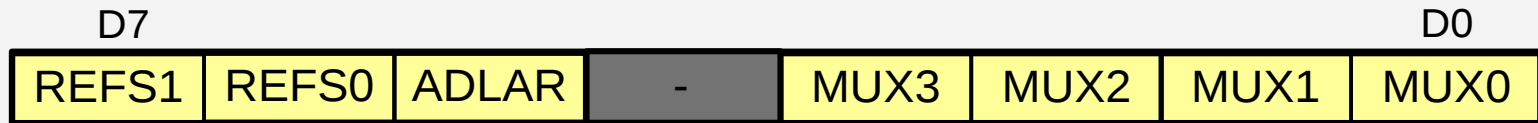
ADC in AVR



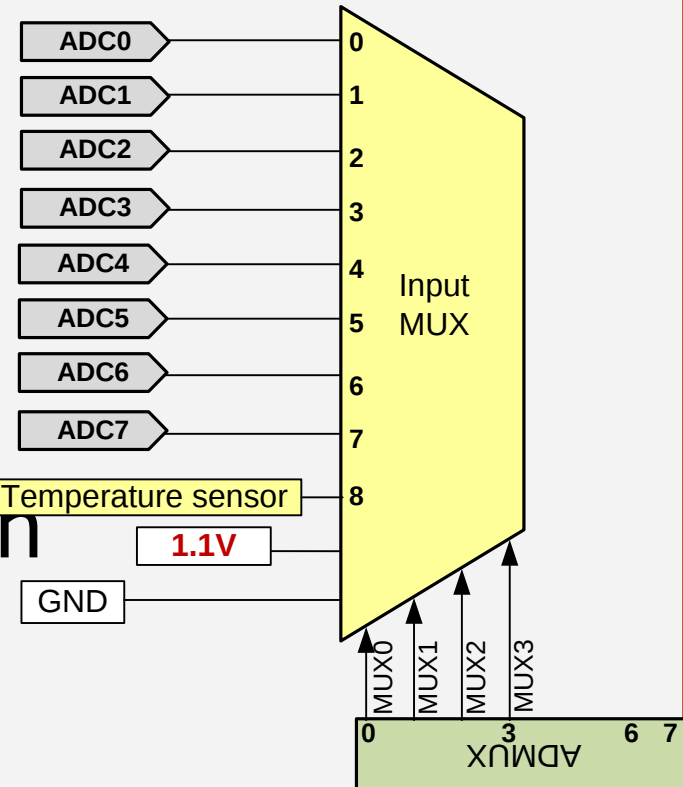


- MUX0-MUX1: input select
- ADLAR:
 - 0: right adjust the result
 - 1: left adjust the result
- REFS1-REFS0: Vref selection

ADMUX



- MUX0-MUX1: input select
- ADLAR:
 - 0: right adjust the result
 - 1: left adjust the result
- REFS1-REFS0: Vref selection





ADMUX

D7

D0

REFS1

REFS0

ADLAR

-

MUX3

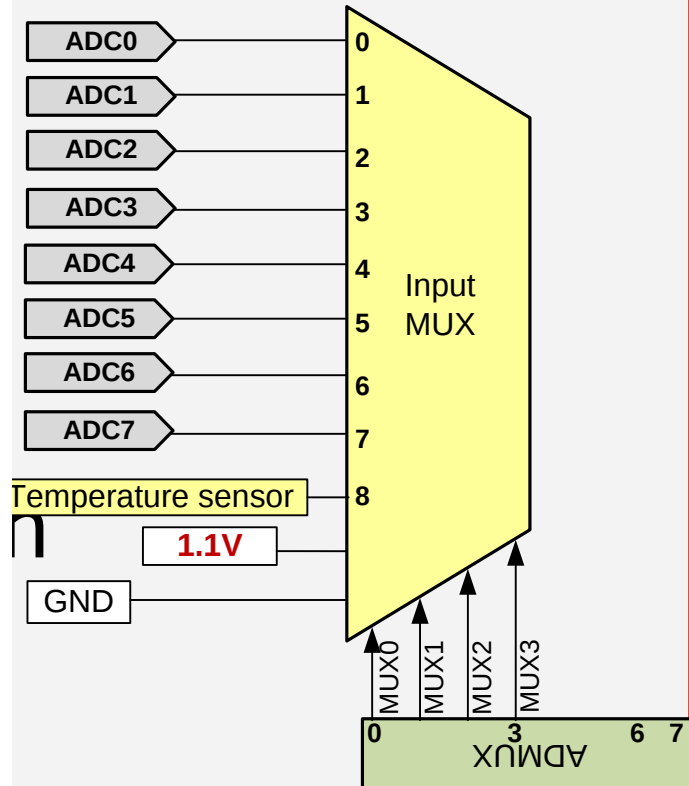
MUX2

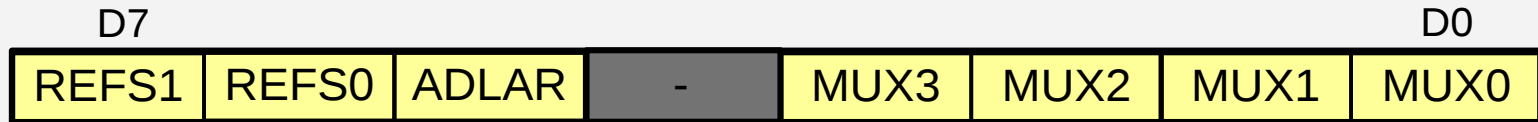
MUX1

MUX0

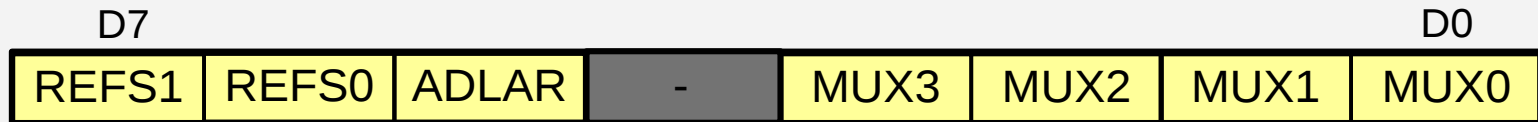
Table 24-4. Input Channel Selections

MUX3...0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8 ⁽¹⁾
1001	(reserved)
1010	(reserved)
1011	(reserved)
1100	(reserved)
1101	(reserved)
1110	1.1V (V _{BG})
1111	0V (GND)





- MUX0-MUX1: input select
- ADLAR:
 - 0: right adjust the result
 - 1: left adjust the result
- REFS1-REFS0: Vref selection

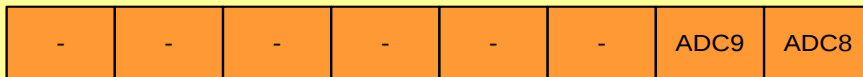


■ MUX0-MUX1: input select

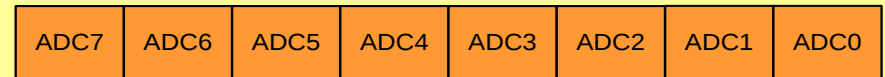
ADLAR = 0

ADLAR = 0

ADCH

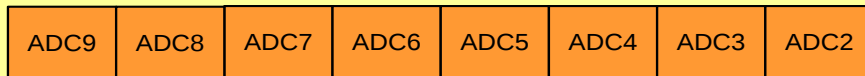


ADCL

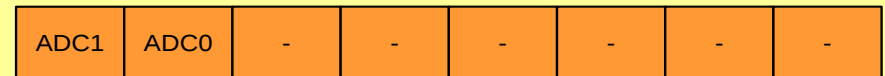


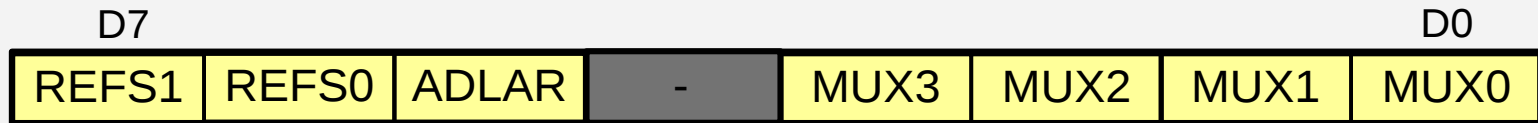
ADLAR = 1

ADCH

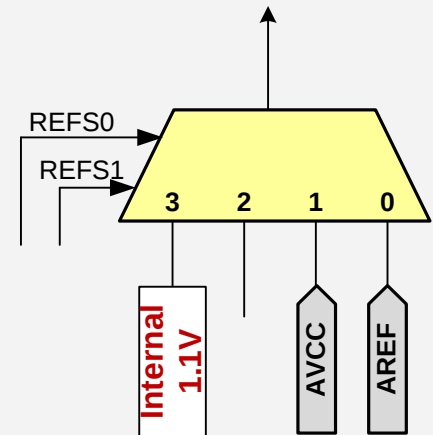


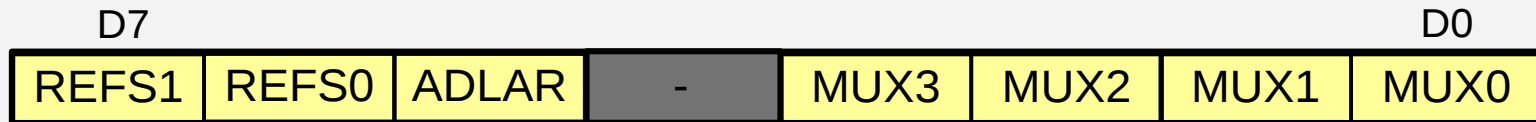
ADCL





- MUX0-MUX1: input select
- ADLAR:
 - 0: right adjust the result
 - 1: left adjust the result
- REFS1-REFS0: Vref selection





- MUX0-MUX1: input select
- ADLAR:
 - 0: right adjust the result
 - 1: left adjust the result
- REFS1-REFS0: Vref selection

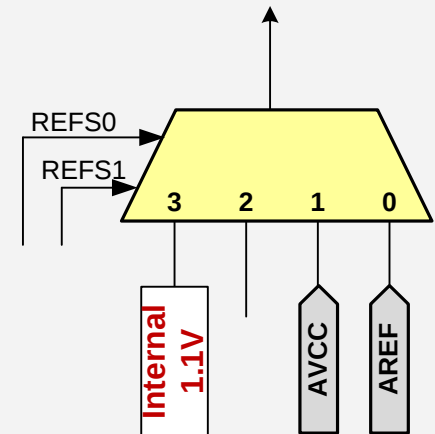
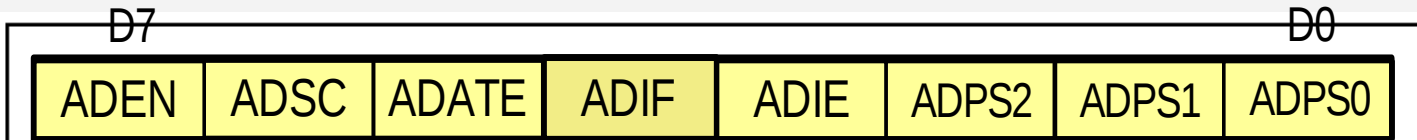


Table 24-3. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal V_{ref} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V Voltage Reference with external capacitor at AREF pin

ADCSA

**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 conversion 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

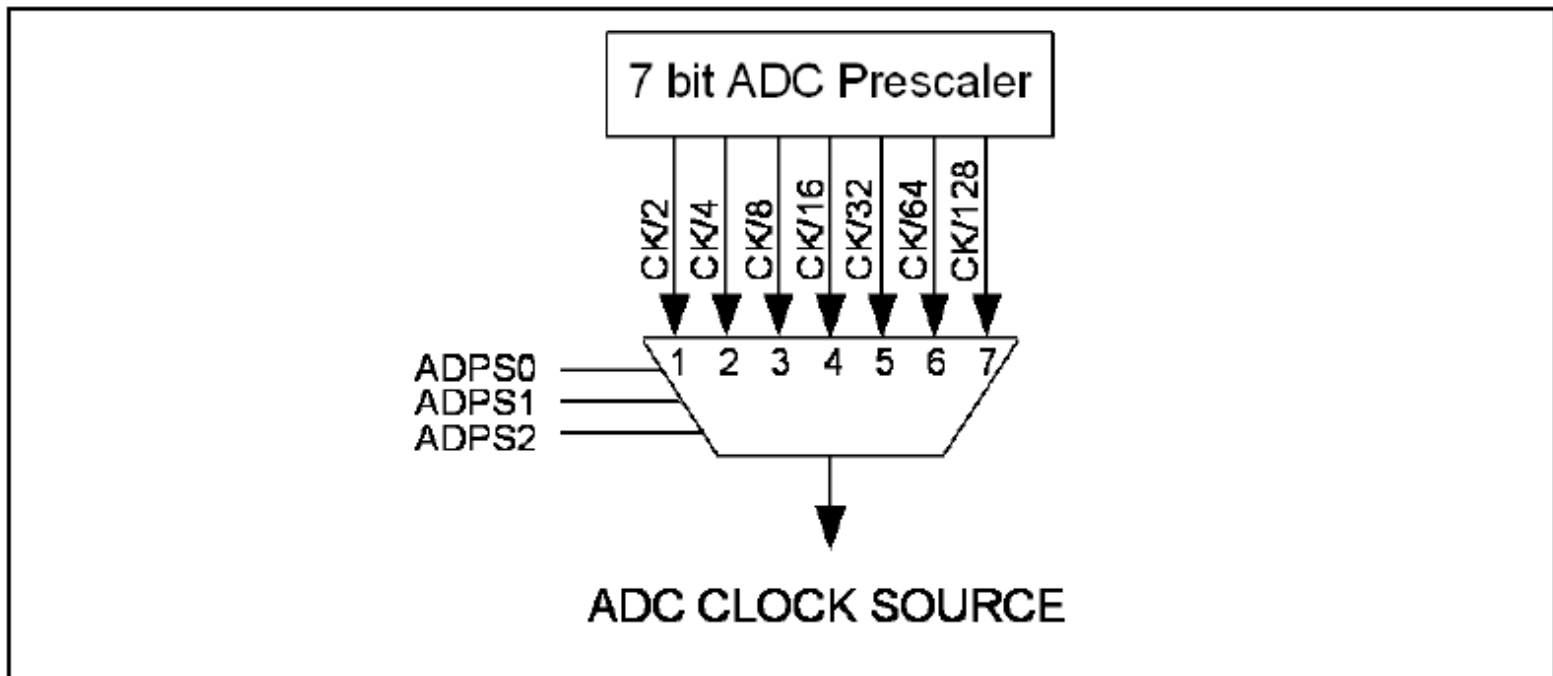
Writing this bit to one enables the ADC Conversion Complete Interrupt.

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





ADCSA

D7

D0

ADEN

ADSC

ADATE

ADIF

ADIE

ADPS2

ADPS1

ADPS0

ADEN- Bit7 ADC Enable

This bit enables the ADC. Writing this bit to one enables the ADC. Writing this bit to zero disables the ADC.

ADSC

To start

ADATE

Auto

ADIF

This bit is related

Table 24-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

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.

~~D0~~



ADCSA

7	6	5	4	3	2	1	0	ADCSRA
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
7	6	5	4	3	2	1	0	ADMUX
REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	
R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
15	14	13	12	11	10	9	8	ADCH ADCL
ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	
ADC1	ADC0	–	–	–	–	–	–	
7	6	5	4	3	2	1	0	
–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
R	R/W	R	R	R	R/W	R/W	R/W	



Steps in programming ADC

1. Make the pin for the selected ADC channel an input pin.
2. Turn on the ADC module
3. Select the conversion speed
4. Select voltage reference and ADC input channels.
5. Activate the start conversion bit by writing a one to the ADSC bit of ADCSRA.
6. Wait for the conversion to be completed by polling the ADIF bit in the ADCSRA register.
7. After the ADIF bit has gone HIGH, read the ADCL and ADCH registers to get the digital data output.
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.



A program with ADC

- This program gets data from channel 0 (ADC0) of ;ADC and displays the result on Port B and Port D.

```
#include <avr/io.h>
#define F_CPU 16000000UL
#include <util/delay.h>

int main (void)
{
    DDRB = 0xFF; //make Port B an output
    DDRD = 0xFF; //make Port D an output

    ADCSRA= 0x87; //make ADC enable and select ck/128
    ADMUX= 0xC8; //1.1V Vref, temp. sensor, right-justified

    while(1)
    {
        ADCSRA |= (1<<ADSC); //start conversion

        while((ADCSRA&(1<<ADIF))==0); //wait for conversion to finish

        ADCSRA |= (1<<ADIF);

        PORTD = ADCL; //give the low byte to PORTD
        PORTB = ADCH; //give the high byte to PORTB

        _delay_ms(100);
    }
}
```

Sensors

- Sensor: Converts a physical signal (e.g. light, temperature, humidity, etc.) to an electrical signal (e.g. resistance, voltage, current, capacitance, etc)



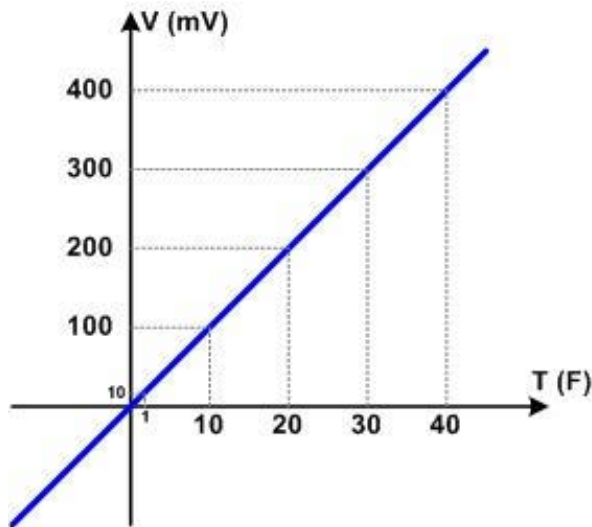
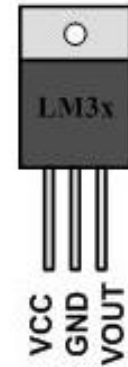
LM35 & LM34 (Temperature Sensors)

- LM35 and LM34:
 - convert temp. to voltage
 - 10mV for each degree

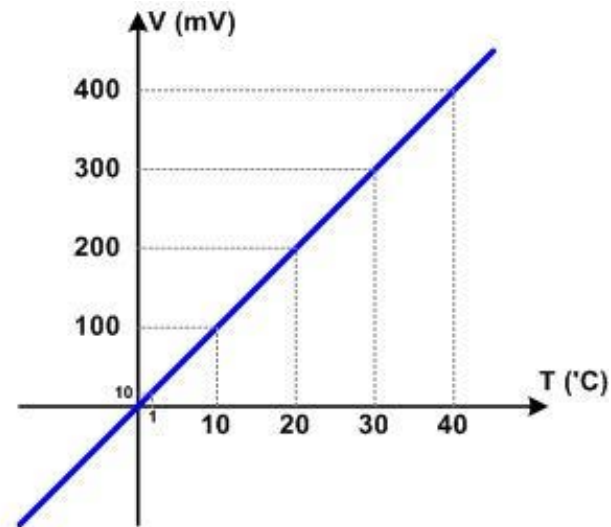
Bottom view
TO92 Package



Top view
TO220 Package



(a) LM34



(b) LM35

Using LM35

```
//this program reads the sensor and displays it on Port D
```

```
#include <avr/io.h> //standard AVR header
```

```
int main (void)
```

```
{
```

```
    DDRB = 0xFF; //make Port B an output
```

```
    DDRC = 0; //make Port C an input for ADC input
```

```
    ADCSRA = 0x87; //make ADC enable and select ck/128
```

```
    ADMUX = 0xC0; //1.1V Vref, ADC0, right-justified
```

```
while (1){
```

```
    ADCSRA |= (1<<ADSC); //start conversion
```

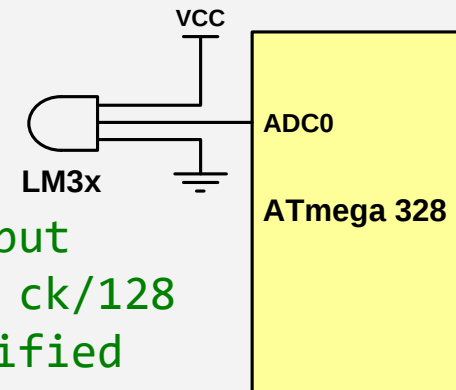
```
    while((ADCSRA&(1<<ADIF))==0); //wait for end of conversion
```

```
    ADCSRA |= (1<<ADIF); //clear the ADIF flag
```

```
    PORTB = (ADCL|(ADCH<<8))*10/93; //PORTB = adc value/9.3
```

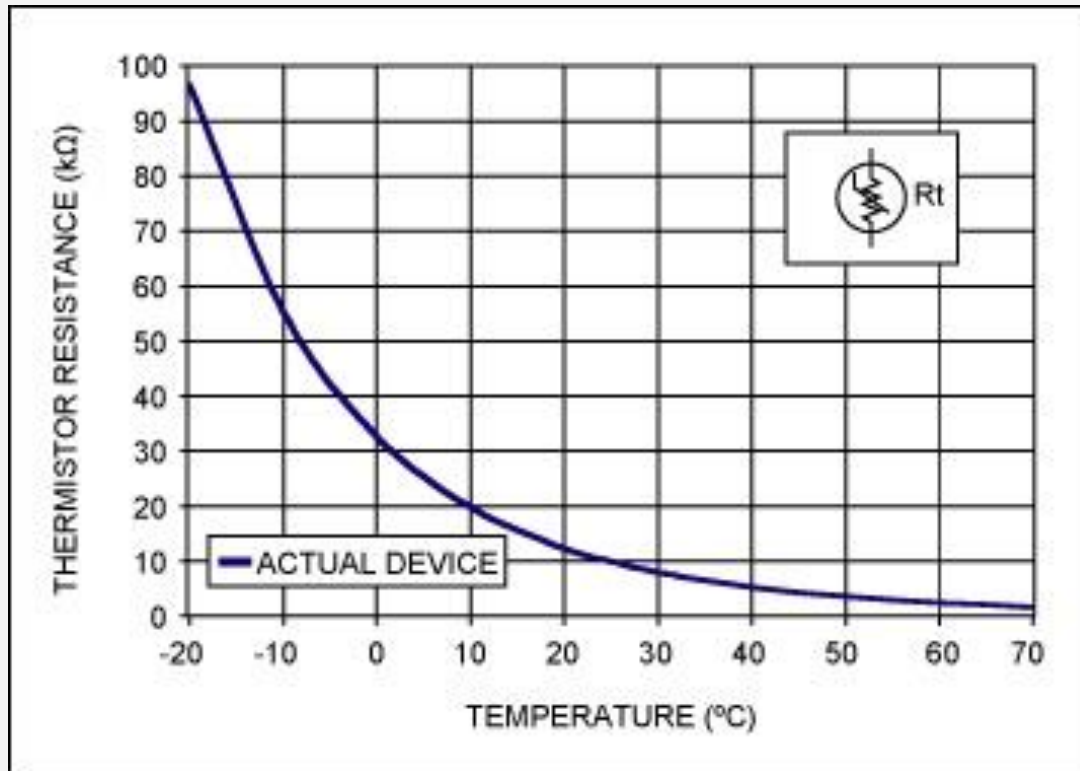
```
}
```

```
}
```



Thermistor (a temperature sensor)

- Converts temperature to resistance
- It is not linear



Signal conditioning

- The output of some sensors (e.g. PT100) is in form of resistance
- Some humidity sensor provide the result in form of Capacitance
- We need to convert these signals to voltage, however, in order to send input to an ADC. This conversion is called signal conditioning.

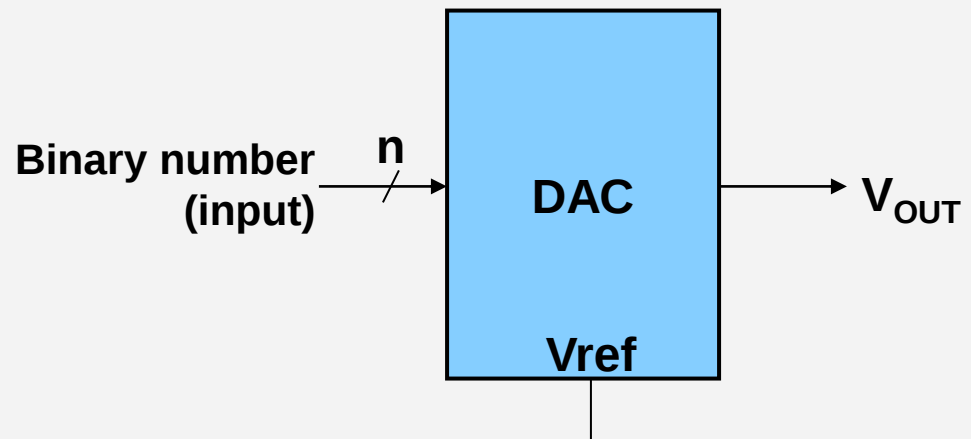
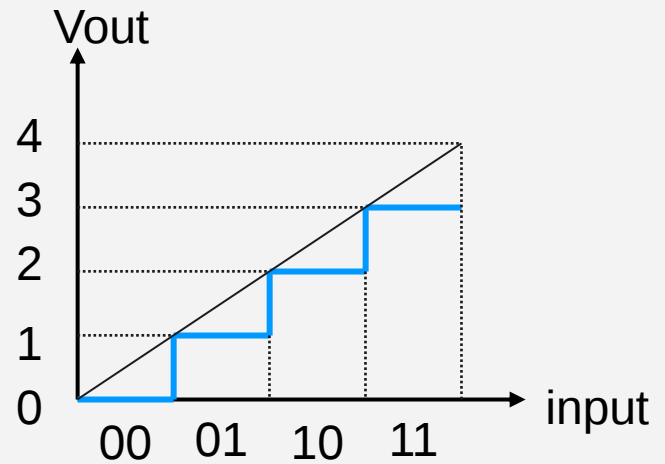




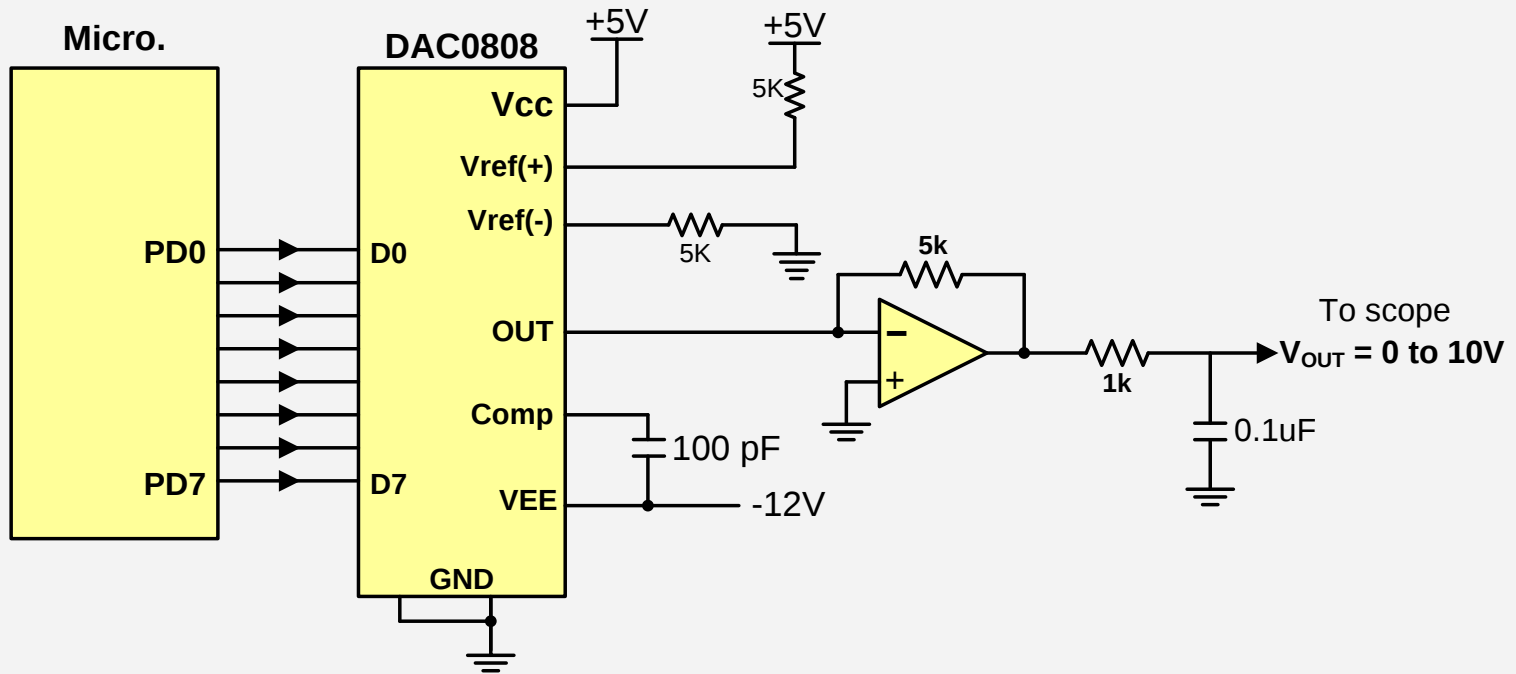
DAC

$$\text{Step size} = \frac{V_{\text{REF}}}{\text{Num of steps}}$$

$$V_{\text{OUT}} = \text{num} \times \text{step size}$$



Connecting a DAC to the microcontroller

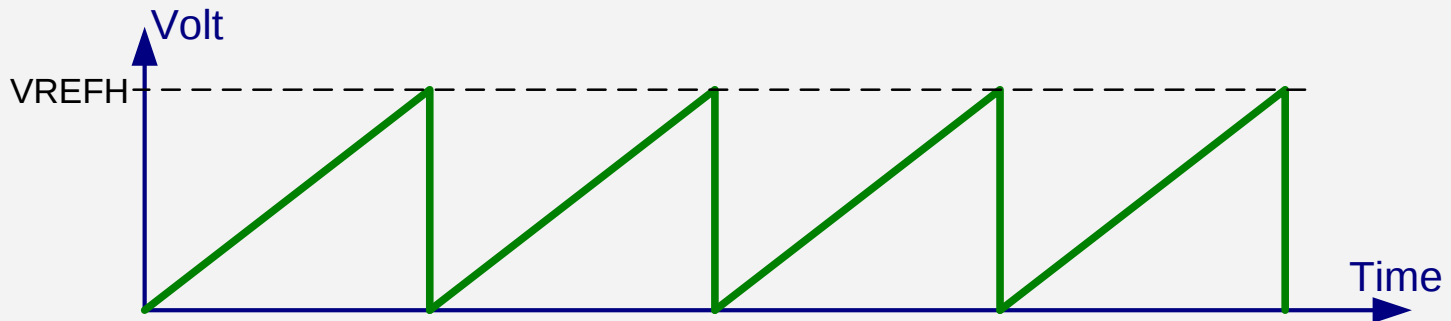




Generating a saw-tooth wave using DAC

```
#include <avr/io.h>

int main (void)
{
    unsigned char i = 0; //define a counter
    DDRD = 0xFF; //make Port D an output
    while (1) //do forever
    {
        PORTD = i; //copy i into PORTD to be converted
        i++; //increment the counter
    }
}
```





Thanks!



Do you have any questions?

razeghizade@gmail.com

www.pudica.ir

CREADITS: This presentation was created by M.Razeghizadeh
Please keep this slide for attribution