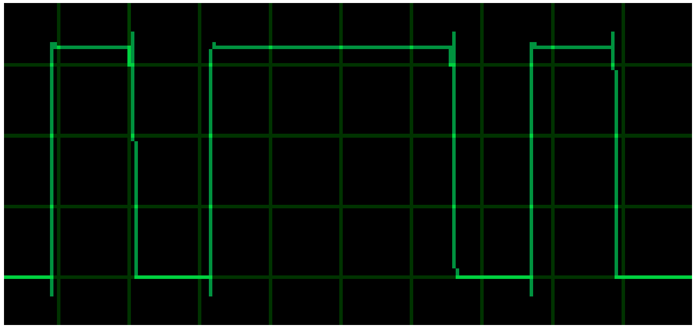
EE-222: Microprocessor Systems

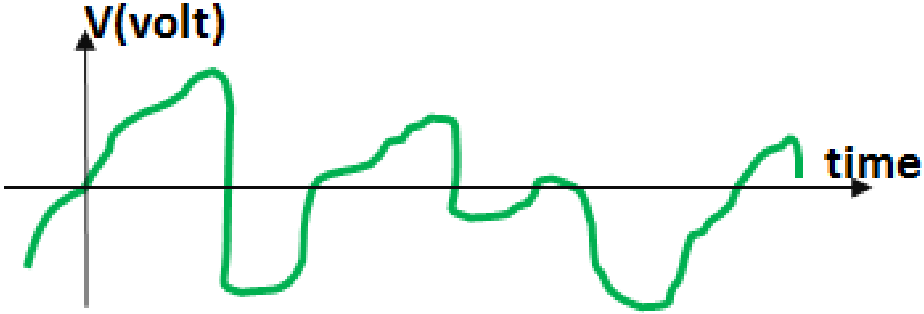
Programming AVR ADC

Instructor: Dr. Arbab Latif

NUST
SCHOOL OF ELECTRICAL ENGINEERING &
COMPUTER SCIENCE (SEECS)

Analog vs. Digital Signals

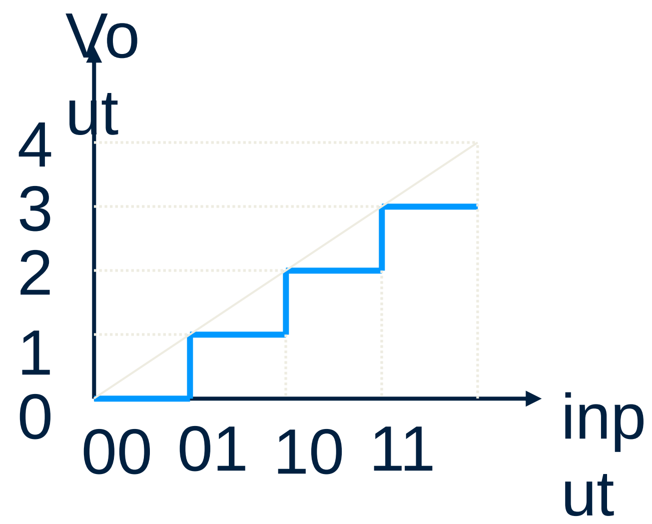


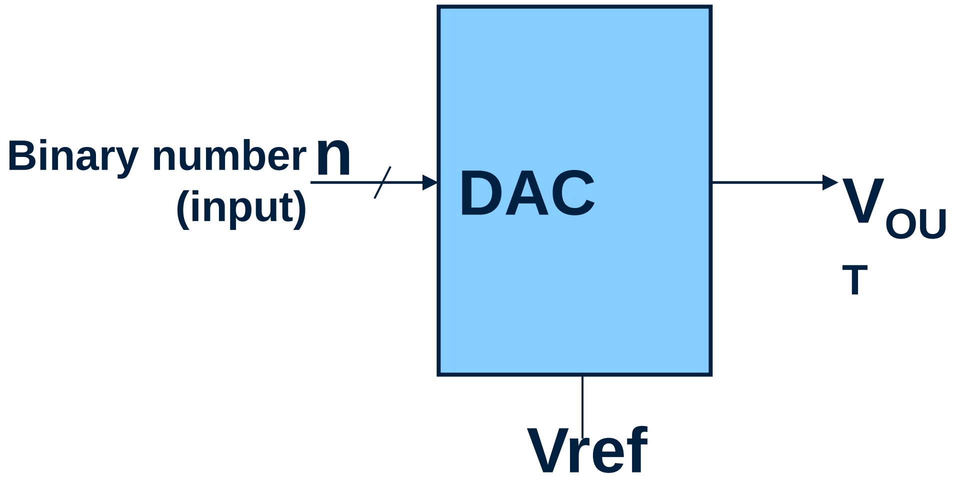


DAC

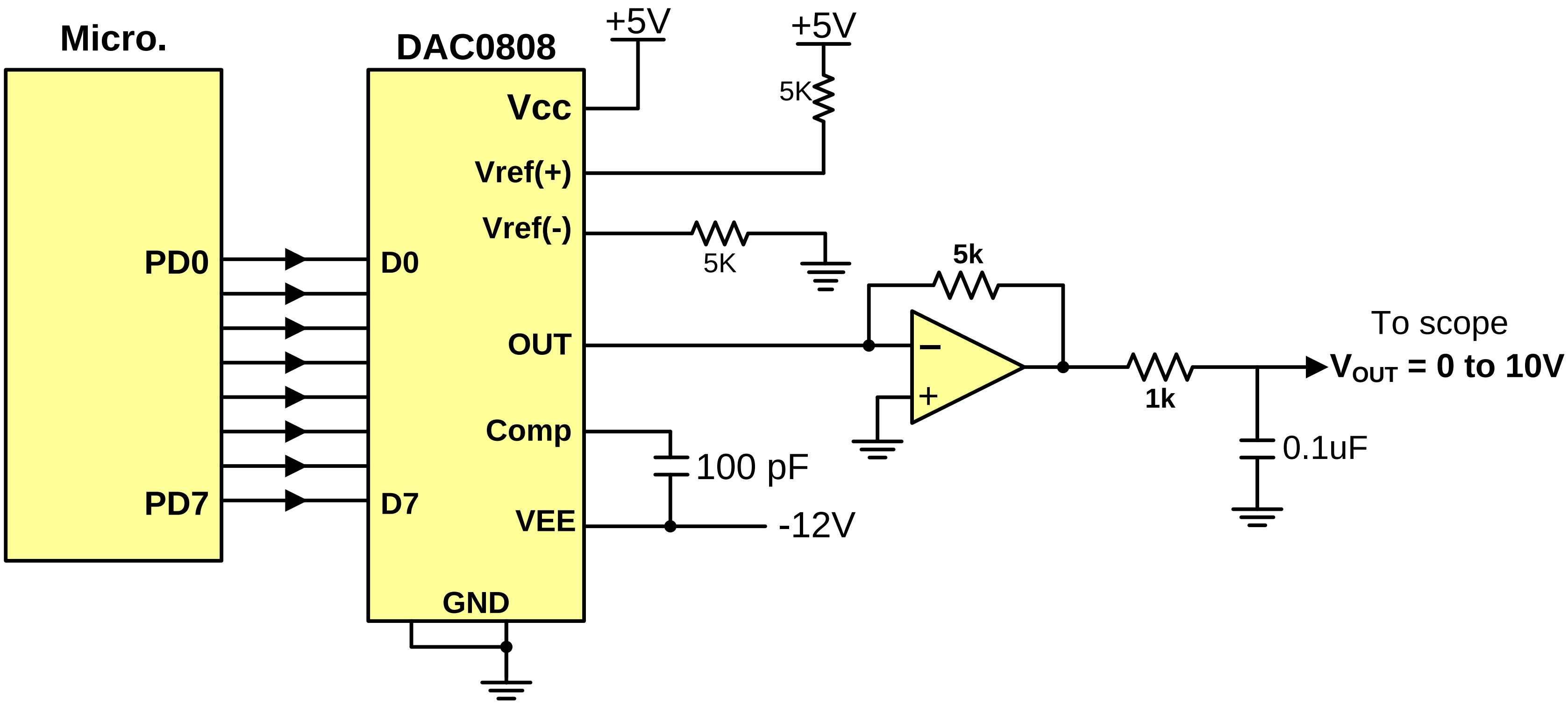
$$\text{Step size} = \frac{V_{REF}}{\text{Num of steps}}$$

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



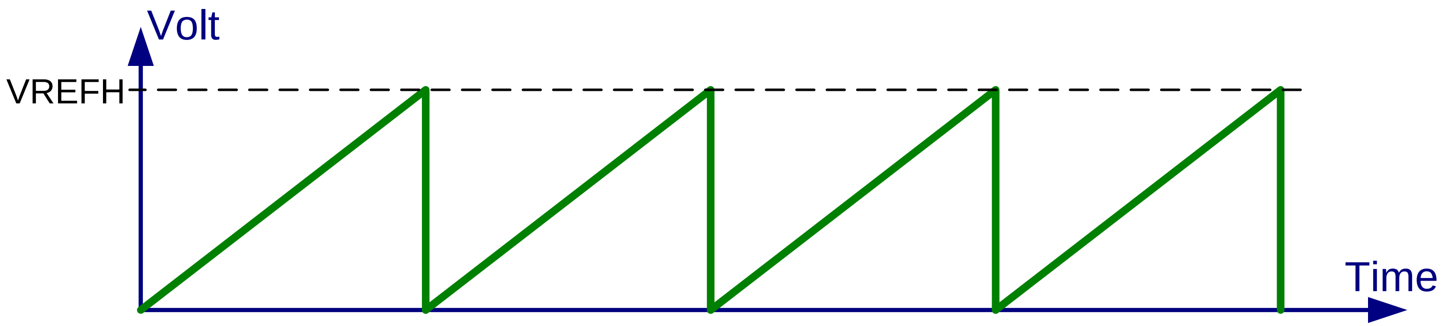


Connecting a DAC to the microcontroller



Generating a saw-tooth wave using DAC

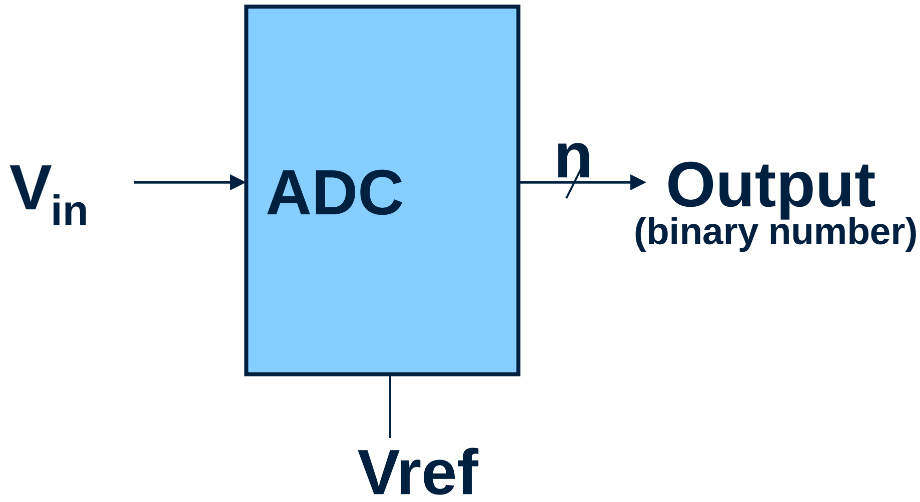
```c
#include <avr/io.h>

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
     }
}
```

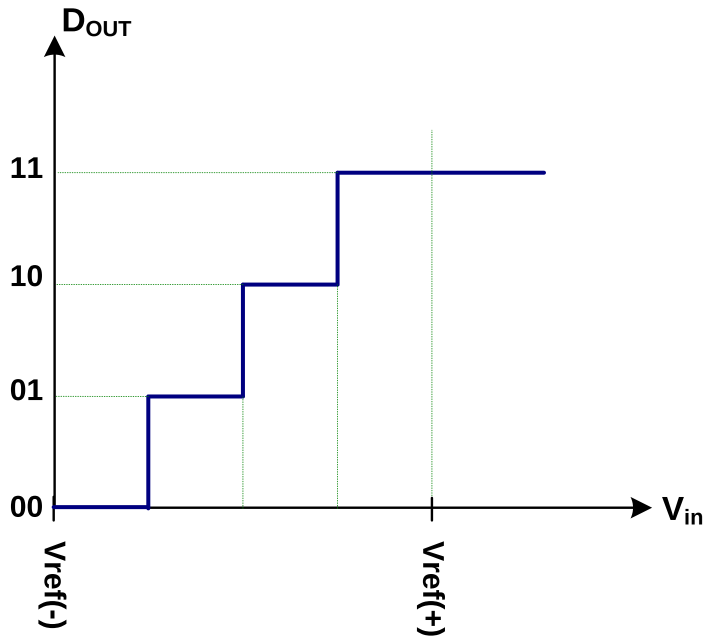


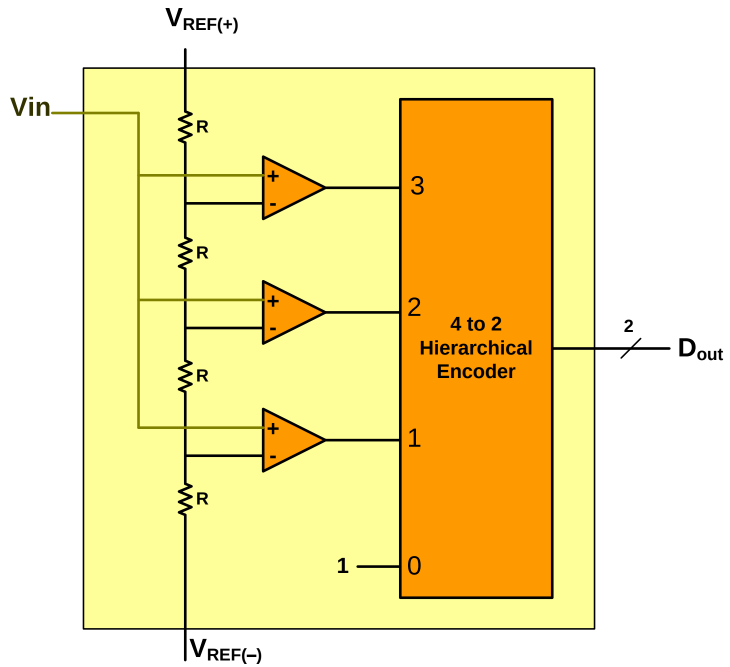
ADC

$$stepSize = \frac{Vref}{numofsteps}$$

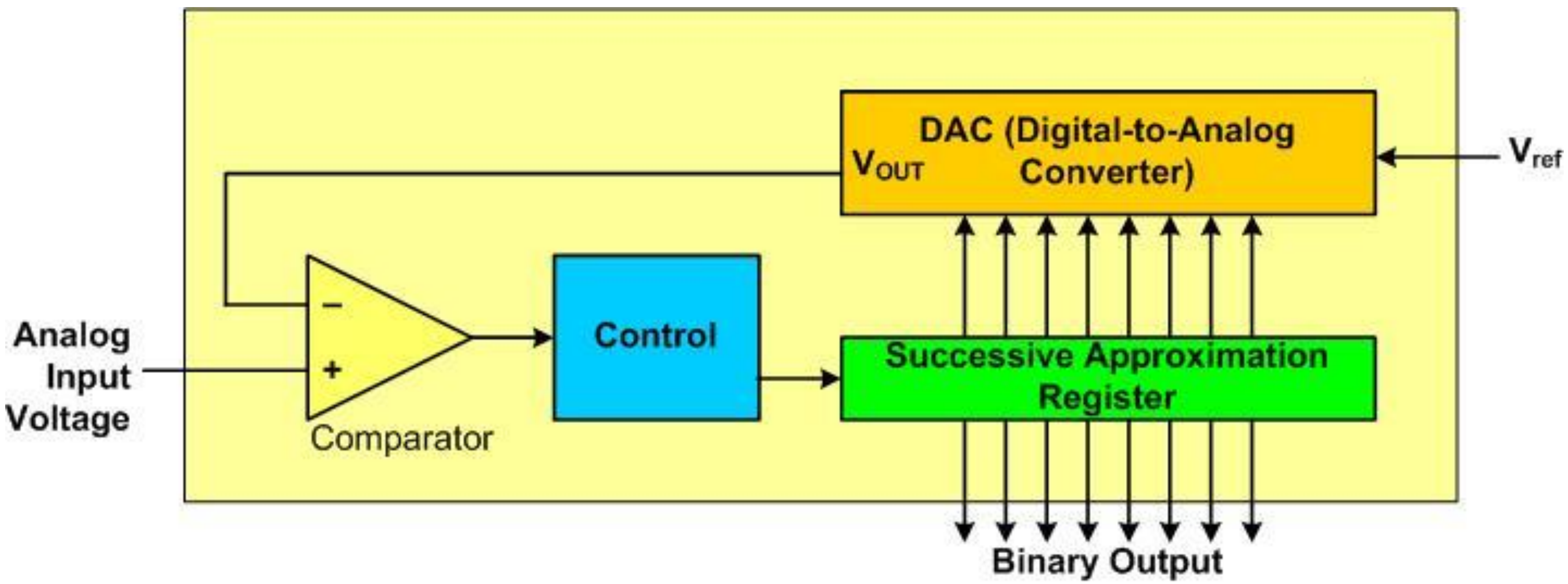
$$output = \left\lfloor \frac{Vin}{stepSize} \right\rfloor$$



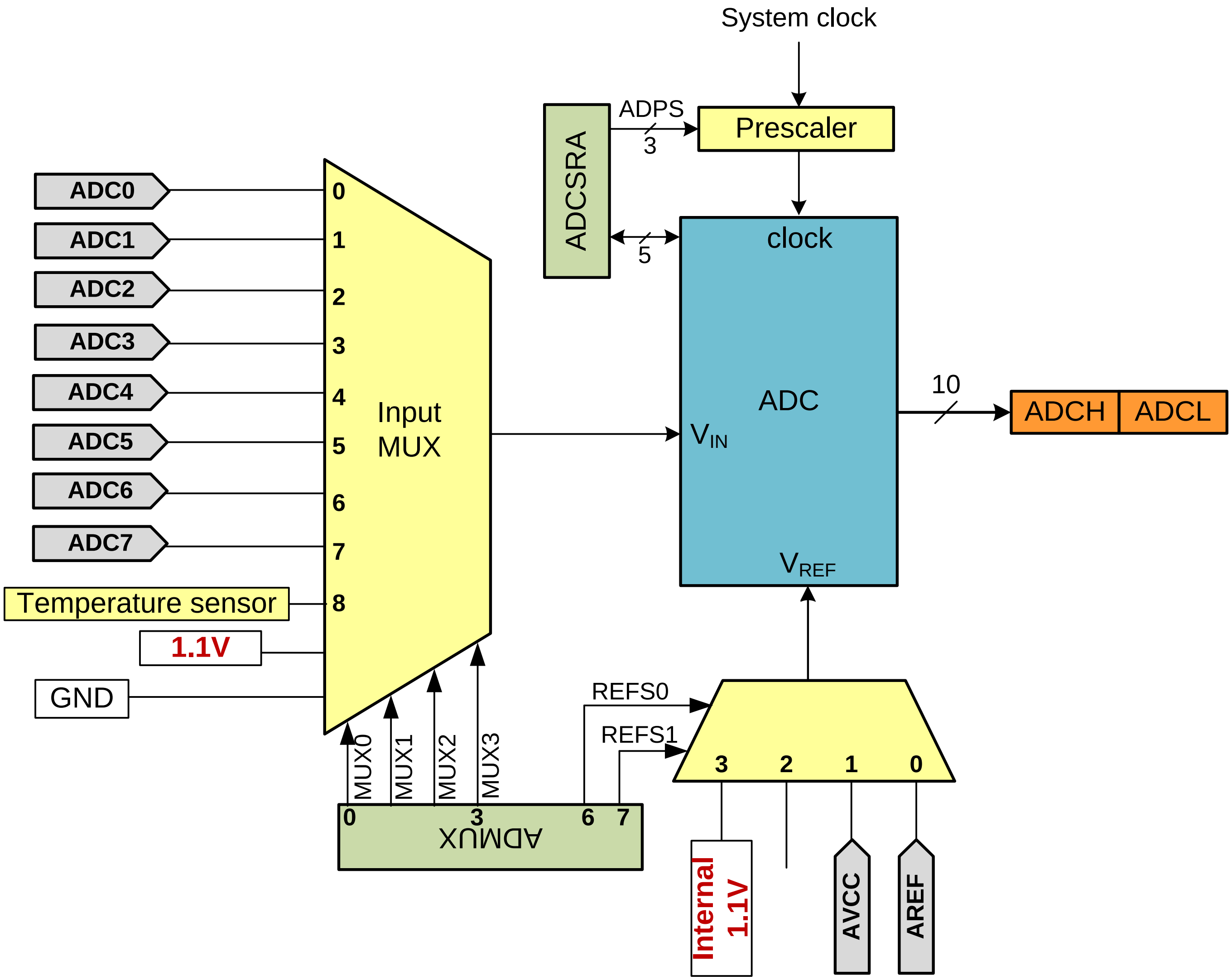




Successive Approximation ADC



ADC in AVR

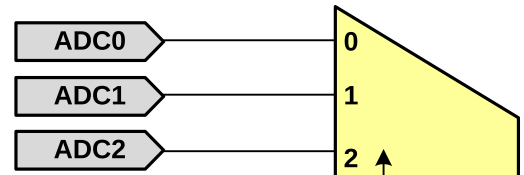


System clock
ADPS
3
Prescaler
ADCSRA
5
clock
ADC
V_IN
V_REF
10
ADCH
ADCL
ADC0
ADC1
ADC2
ADC3
ADC4
ADC5
ADC6
ADC7
Temperature sensor
1.1V
GND
0
1
2
3
4
5
6
7
8
Input
MUX
MUX0
MUX1
MUX2
MUX3
0
3
6
7
ADMUX
REFS0
REFS1
3
2
1
0
Internal 1.1V
AVCC
AREF

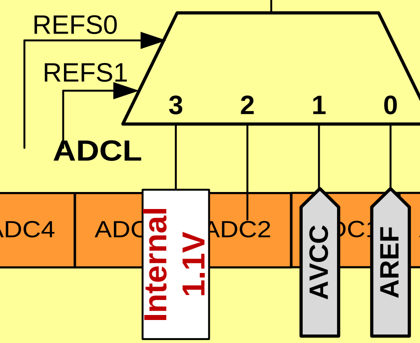
ADMUX

D7							D0
REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0

- MUX0-MUX1: input select
- ADLAR:



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	-	-	-	-	-	-

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.

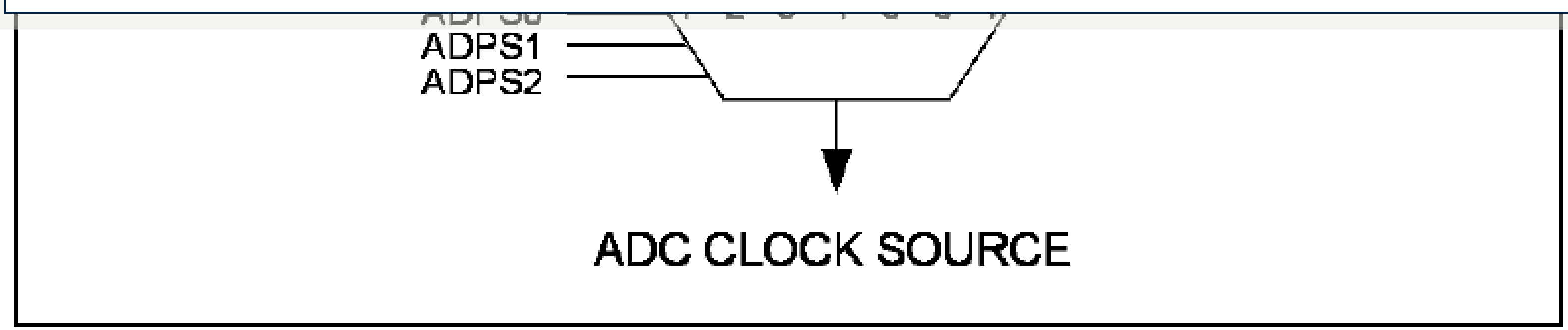
ADPS2:0- Bit2:0 ADC Prescaler Select Bits
These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

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



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.

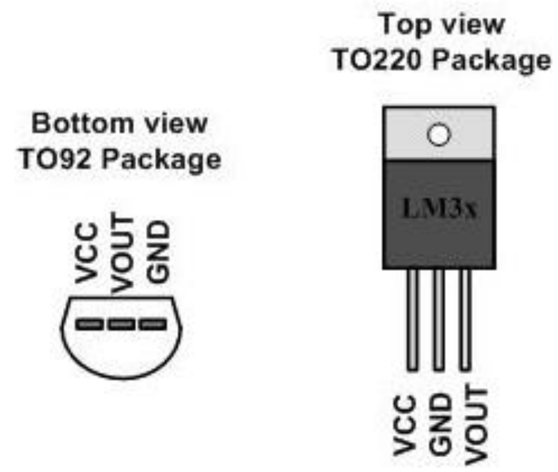
```
#include <avr/io.h>
#define F_CPU 16000000UL
#include <util/delay.h>

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);
 }
}
```

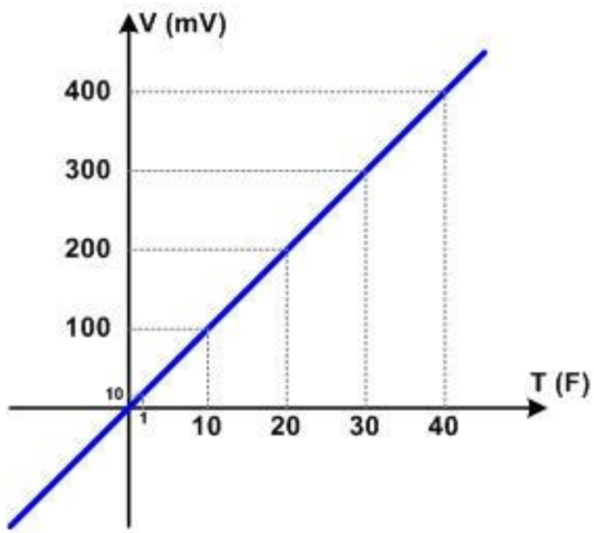
Sensors

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

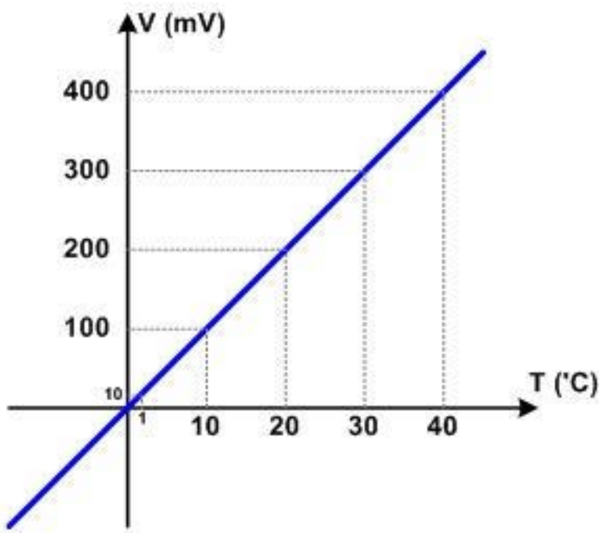
LM35 & LM34 (Temperature Sensors)

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





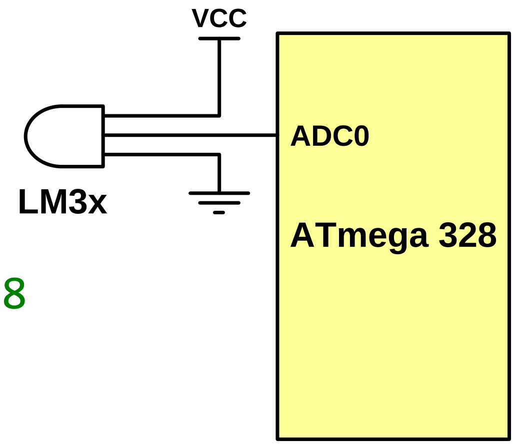
(a) LM34



(b) LM35

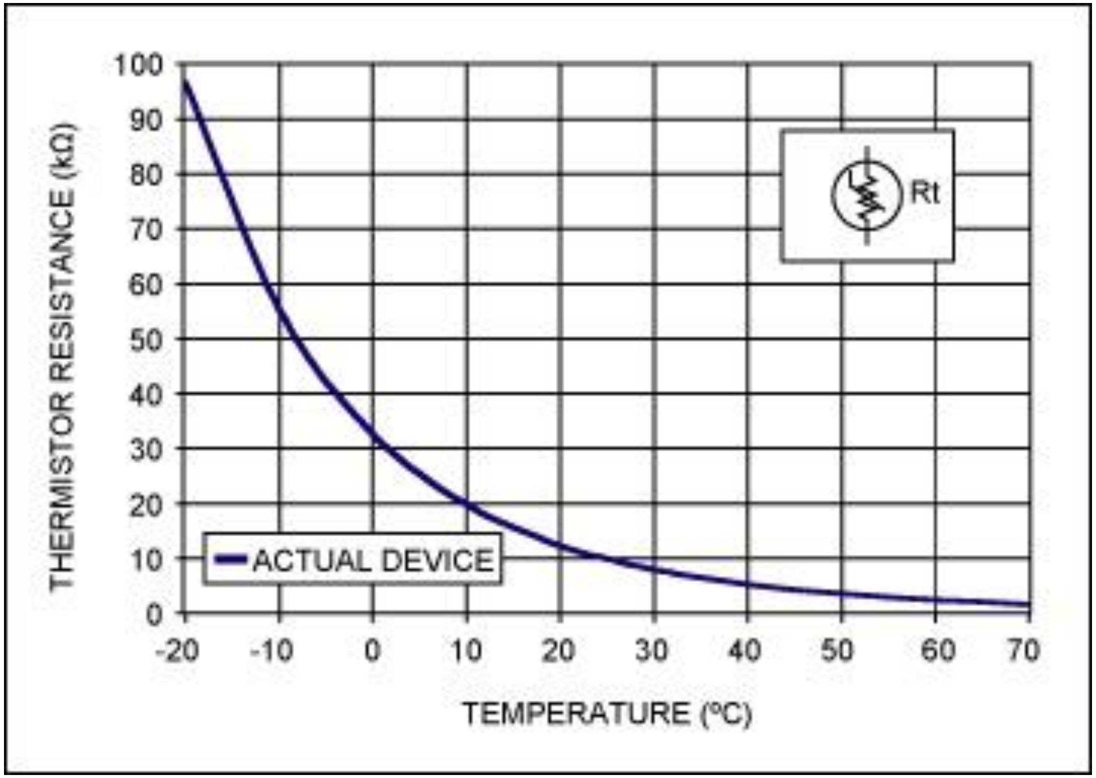
Using LM35

```c
//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.