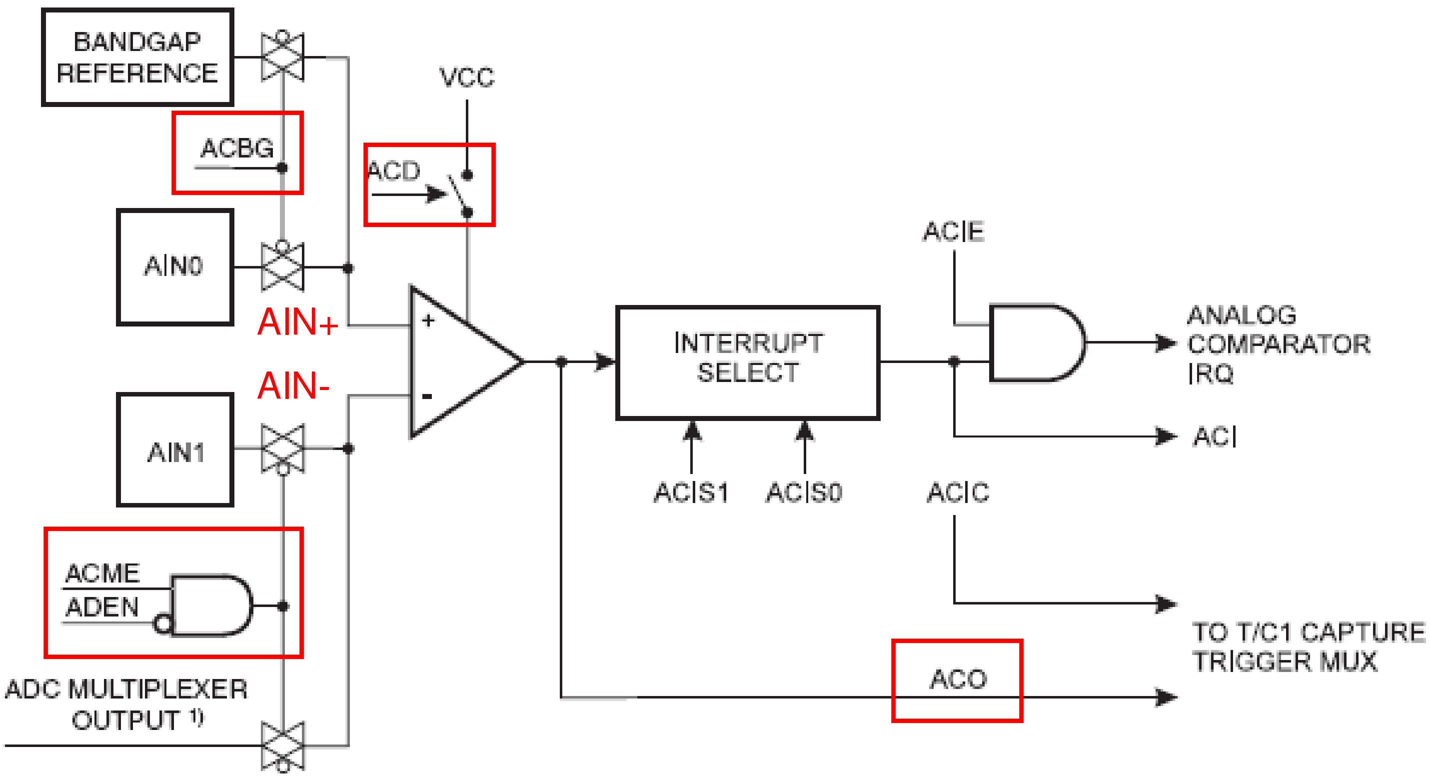
# Design with Microprocessors

## Lecture 9

Year 3 CS
Academic year 2017/2018
1st Semester

Lecturer: Radu Dănescu

# Analog Comparator




Compares the analog values from AIN+ (positive) & AIN- (negative)

If **(AIN+) > (AIN-)** ➔ **ACO** = **1**

Enabling the Analog Comparator: bit 7 (ACD) from ACSR register

AIN+ (Input signal): external signal AIN0 or internal reference selected through ACBG (bit 6 from ACSR)

AIN- (Input signal): external signal (AIN1) (ACME=0 or ADEN=1) or input from $AD_{0..7}$ (ACME=1 and ADEN=0).

# Analog Comparator

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x7B) | – | ACME | – | – | – | ADTS2 | ADTS1 | ADTS0 | ADCSRB |
| Read/Write | R | R/W | R | R | R | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

## ADCSRB – ADC Control and Status Register B

• **Bit 6 – ACME: Analog Comparator Multiplexer Enable**

**ACME ← 1, if** ADC is disconnected (ADEN=0) **then** $AD_{0..7}$ is applied on AIN-

**ACME ← 0, external** AIN1 signal is applied on AIN-

## Analog Comparator Multiplexed Input (ATmega328P / UNO)

| ACME | ADEN | MUX2..0 | Analog Comparator Negative Input (AIN-) |
|---|---|---|---|
| 0 | x | xxx | AIN1 |
| 1 | 1 | xxx | AIN1 |
| 1 | 0 | 000 | ADC0 |
| 1 | 0 | 001 | ADC1 |
| 1 | 0 | 010 | ADC2 |
| 1 | 0 | 011 | ADC3 |
| 1 | 0 | 100 | ADC4 |
| 1 | 0 | 101 | ADC5 |
| 1 | 0 | 110 | ADC6 |
| 1 | 0 | 111 | ADC7 |

# Analog Comparator

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x7B) | – | ACME | – | – | MUX5 | ADTS2 | ADTS1 | ADTS0 | ADCSRB |
| Read/Write | R | R/W | R | R | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

## Analog Comparator Multiplexed Input (ATmega2560 / MEGA)

| ACME | ADEN | MUX5 | MUX2..0 | Analog Comparator Negative Input (AIN-) |
|---|---|---|---|---|
| 0 | x | x | xxx | AIN1 |
| 1 | 1 | x | xxx | AIN1 |
| 1 | 0 | 0 | 000 | ADC0 |
| 1 | 0 | 0 | 001 | ADC1 |
| 1 | 0 | 0 | 010 | ADC2 |
| 1 | 0 | 0 | 011 | ADC3 |
| 1 | 0 | 0 | 100 | ADC4 |
| 1 | 0 | 0 | 101 | ADC5 |
| 1 | 0 | 0 | 110 | ADC6 |
| 1 | 0 | 0 | 111 | ADC7 |
| 1 | 0 | 1 | 000 | ADC8 |
| 1 | 0 | 1 | 001 | ADC9 |
| 1 | 0 | 1 | 010 | ADC10 |
| 1 | 0 | 1 | 011 | ADC11 |
| 1 | 0 | 1 | 100 | ADC12 |
| 1 | 0 | 1 | 101 | ADC13 |
| 1 | 0 | 1 | 110 | ADC14 |
| 1 | 0 | 1 | 111 | ADC15 |

# Analog Comparator

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 0x30 (0x50) | ACD | ACBG | ACO | ACI | ACIE | ACIC | ACIS1 | ACIS0 | ACSR |
| Read/Write | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | N/A | 0 | 0 | 0 | 0 | 0 | |

## ACSR – Analog Comparator Control and Status Register

**• Bit 7 – ACD: Analog Comparator Disable**

ACD ← 1, Analog Comparator is disconnected (reduces power consumption)

**• Bit 6 – ACBG: Analog Comparator Bandgap Select**

ACBG ← 1, fixed bandgap reference voltage to Analog Comparator(AN+)

ACBG ← 0, external AIN0 is applied to the Analog Comparator (AN+)

**• Bit 5 – ACO: Analog Comparator Output**

Analog Comparator output is synchronized and connected to ACO (the synchronization is introducing a delay (1…2 clocks)

**• Bit 4 – ACI: Analog Comparator Interrupt Flag**

ACI ← 1, by hardware, when the output of the comparator triggers an interrupt according to ACIS1 and ACIS0. Analog Comparator Interrupt (AC_IR) is generated if ACSR (ACIE) && SREG(I) are set (=1)

ACI ← 0, by hardware (AC-ISR is in execution) or by software

# Analog Comparator

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 0x30 (0x50) | ACD | ACBG | ACO | ACI | ACIE | ACIC | ACIS1 | ACIS0 | ACSR |
| Read/Write | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | N/A | 0 | 0 | 0 | 0 | 0 | |

## ACSR – Analog Comparator Control and Status Register

• **Bit 3 – ACIE: Analog Comparator Interrupt Enable**

ACIE ← 1 and SREG(I) ← 1, Analog Comparator interrupt is validated

ACIE ← 0, Analog Comparator interrupt is invalidated

• **Bit 2 – ACIC: Analog Comparator Input Capture Enable**

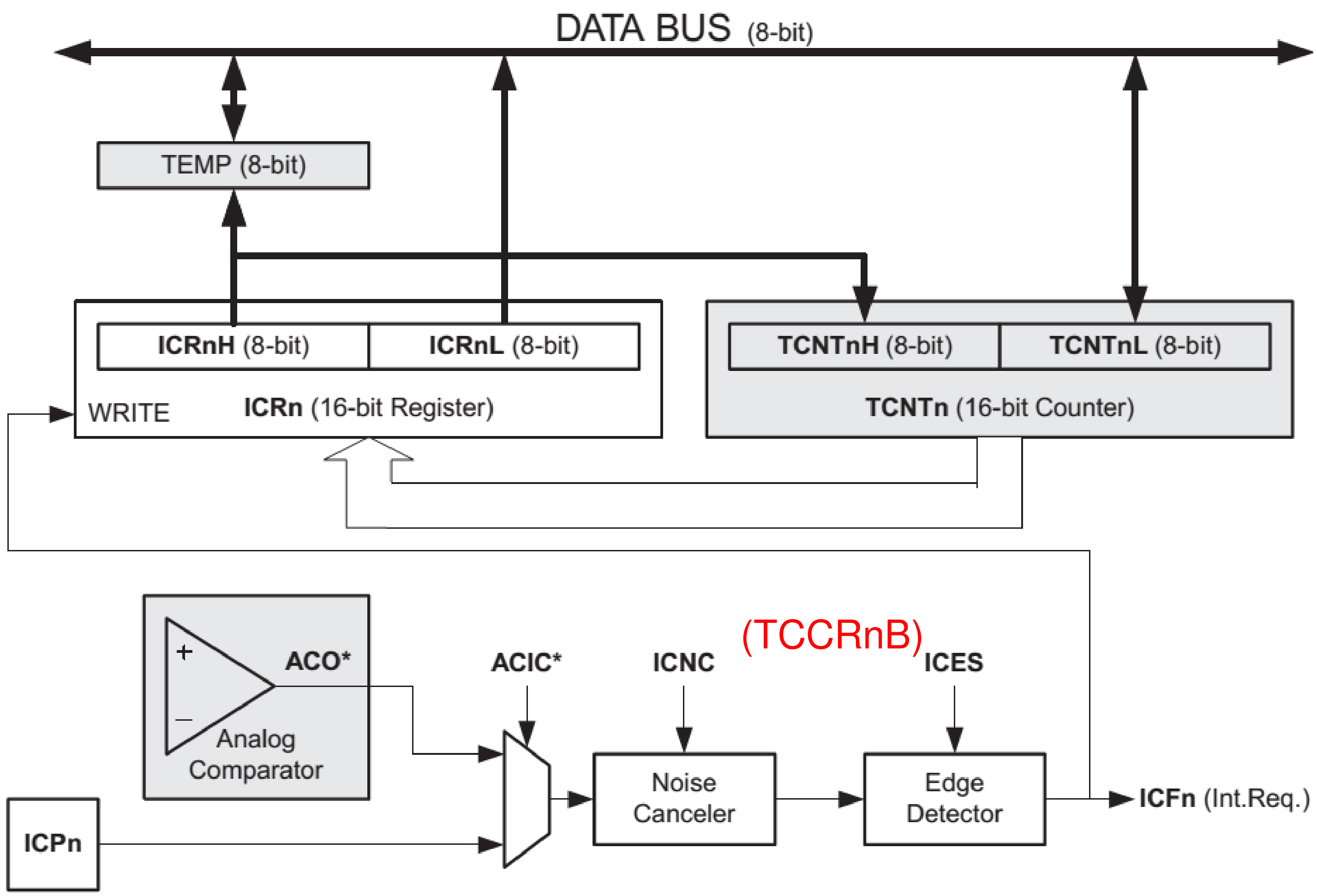
ACIC ← 1, Input Capture function (interrupt) in Timer/Counter1 will be triggered by the Analog Comparator (if bit ICIE1 in Timer1 Interrupt Mask Register (TIMSK) is enabled

ACIC ← 0, Analog Comparator not connected to Timer/Counter1 input capture (input capture can be triggered from ICPn pin (hardware or software)

• **Bits 1, 0 – ACIS1, ACIS0: Analog Comparator Interrupt Mode Select**

| ACIS1 | ACIS0 | Interrupt Mode |
|---|---|---|
| 0 | 0 | Comparator Interrupt on Output Toggle. |
| 0 | 1 | Reserved |
| 1 | 0 | Comparator Interrupt on Falling Output Edge |
| 1 | 1 | Comparator Interrupt on Rising Output Edge. |

# Analog Comparator & Timer1




**Capture: ICF1 ← 1 & WRITE ← 1 ⇨ ICR1 = TCNT1;**

ICR1 ⇨ Time-stamp for external events (measure frequency, fill factor, … )

# Analog Comparator & Timer1

**Example: measuring the capacity (of a capacitor)**

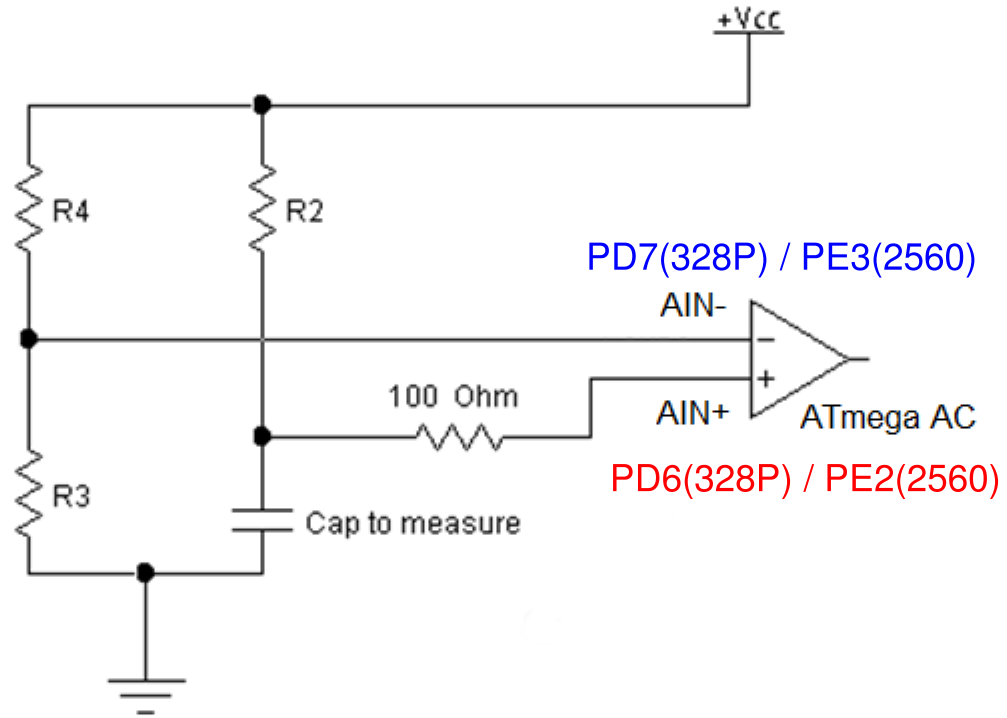
$v(t)=V_{cc}(1\text{-}exp(-t/T))$ (1)
T=R2 * C (2)
R2>> (100 ohms)

Algorithm:

1. Set PORTnx(AIN-) as input
2. Configure AC and Timer1
3. Set PORTny(AIN+) as output and write “0” (discharge the capacitor)
4. Set PORTny(AIN+) as input and Start Timer1. The capacitor will begin to charge ….

ISR for timer 1 capture:

1. Read ICR1 register
2. Convert ICR1 to sec ⇨ t
3. Compute C from (1) + (2) + (3)




ISR will be triggered when the voltage over the capacitor $V_+$ equals $V_-$ :

$$v(t) = V_{cc}{}^*R_3/(R_3+R_4) \quad (3)$$

# Analog to digital converter (ADC)

**Atmega 328P**

- 10-bit Resolution
- 8 Multiplexed Single Ended Input Channels
- Temperature sensor input
- Optional Left Adjustment for ADC Result Readout
- 0 … $V_{CC}$ ADC Input Voltage Range
- 1.1 V internal ADC Reference Voltage
- Free Running or Single Conversion Mode
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceller

**ATmega2560**

- 10-bit Resolution
- 16 Multiplexed Single Ended Input Channels
- 14 Differential input channels
- 4 Differential Input Channels with Optional Gain of 10× and 200×
- Optional Left Adjustment for ADC Result Readout
- 0V … VCC ADC Input Voltage Range
- 2.7V - VCC Differential ADC Voltage Range
- Selectable 2.56V or 1.1V internal ADC Reference Voltage
- Free Running or Single Conversion Mode
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceller

Single ended input channel measurement:

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

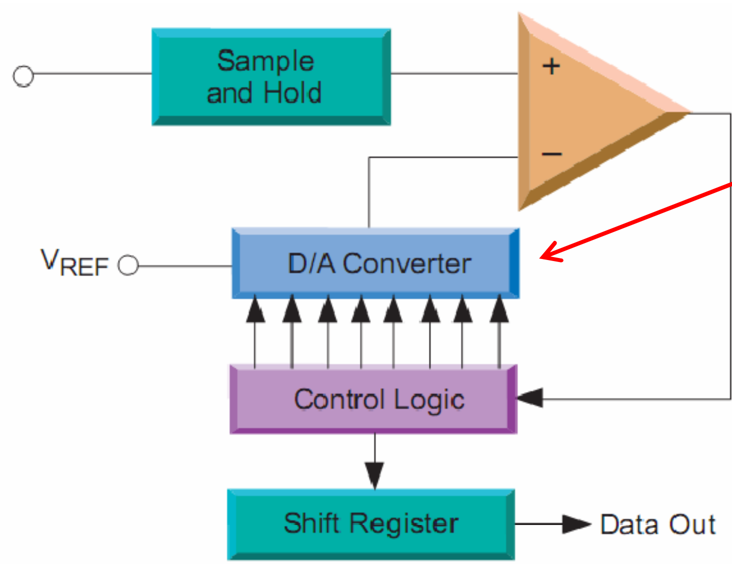
Differential input channel measurement:

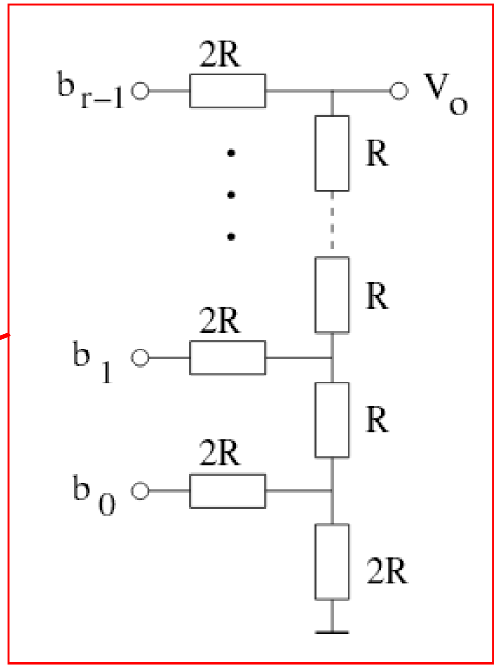
$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot GAIN \cdot 512}{V_{REF}}$$

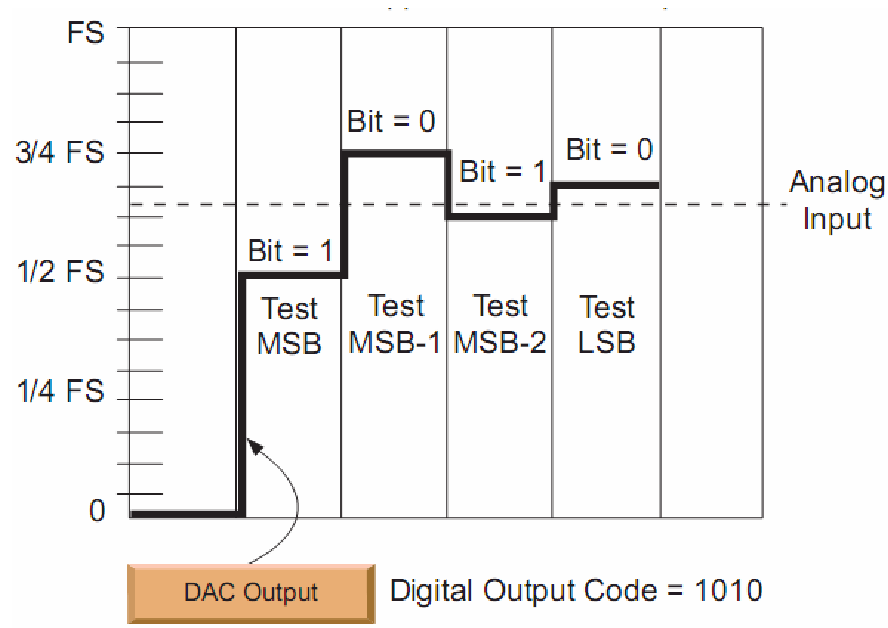
# ADC

## ADC principle:

- Successive comparisons with a reference voltage



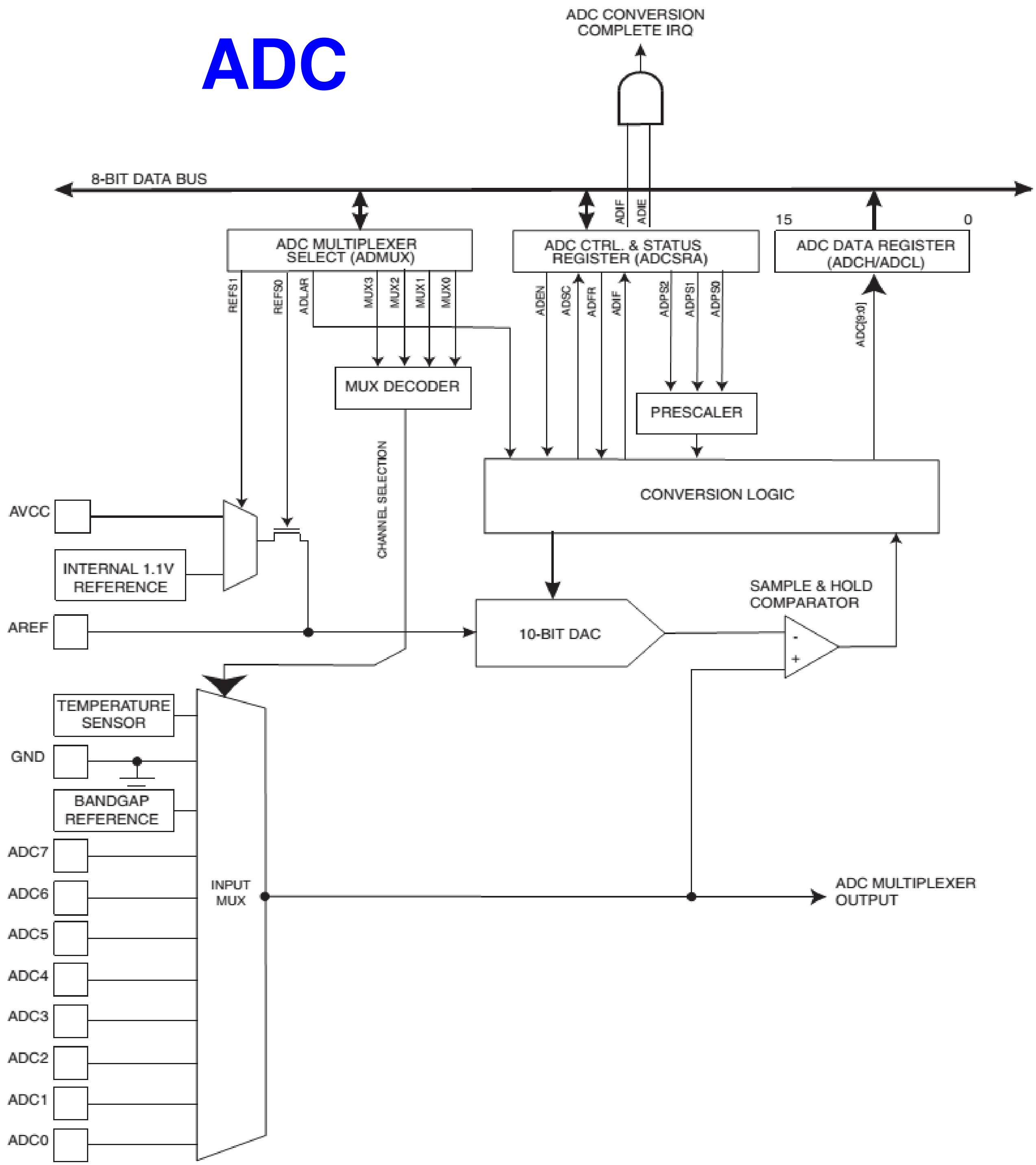



# ADC

## Bloc diagram

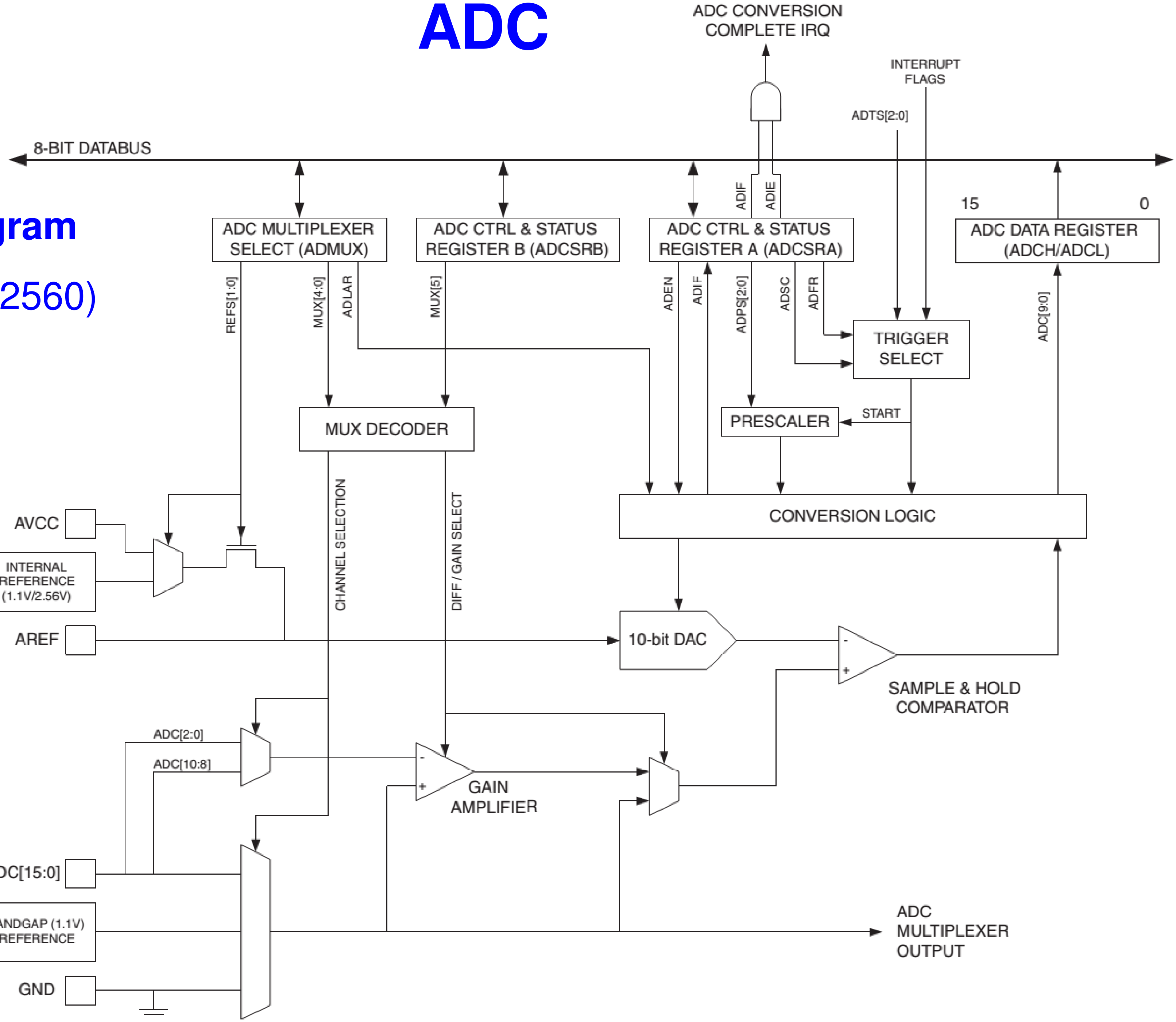
(Atmega 328P)



# ADC

## Bloc diagram

(ATmega 2560)



# ADC

## ADC configuration

### ADCSRA – ADC Control and Status Register A

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| ADEN | ADSC | ADATE | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 | ADCSRA |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

ADEN – ADC activation (ADEN=1)

ADIE – ADC Interrupt Enable (ADIE= 1 & SREG(I)=1 $\Rightarrow$ ADC IRQ activated)

ADPS2 ..0 – clock prescaler

| 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 |

# ADC

## ADC configuration

### ADMUX – ADC Multiplexer Selection Register

### REFS1..0 – reference voltage selection

ATmega328P (UNO)

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x7C) | REFS1 | REFS0 | ADLAR | MUX4 | MUX3 | MUX2 | MUX1 | MUX0 | ADMUX |
| 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 | |

ATmega2560 (MEGA)

| REFS1 | REFS0 | Voltage Reference Selection[1] |
|---|---|---|
| 0 | 0 | AREF, Internal $V_{REF}$ turned off |
| 0 | 1 | AVCC with external capacitor at AREF pin |
| 1 | 0 | Internal 1.1V Voltage Reference with external capacitor at AREF pin |
| 1 | 1 | Internal 2.56V Voltage Reference with external capacitor at AREF pin |

ATmega328P (UNO)

| 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 |

### ADLAR: ADC Left Adjust Result

**ADLAR ← 1,** result aligned to left (if only ADCH is read – 8 bit result – lower resolution)

**ADCH = Vin*256/Vref**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|
| ADC9 | ADC8 | ADC7 | ADC6 | ADC5 | ADC4 | ADC3 | ADC2 | ADCH |
| ADC1 | ADC0 | – | – | – | – | – | – | ADCL |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

**ADLAR ← 0,** result aligned to right

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | ADC9 | ADC8 | ADCH |
| ADC7 | ADC6 | ADC5 | ADC4 | ADC3 | ADC2 | ADC1 | ADC0 | ADCL |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

# ADC

## MUX5:0: Analog Channel and Gain Selection Bits

ATmega328P

| 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) |

Note: 1. For Temperature Sensor.

See ATmega328 and 2560 datasheets for the complete table

ATmega2560

| MUX5:0 | Single Ended Input | Positive Differential Input | Negative Differential Input | Gain |
|---|---|---|---|---|
| 000000 | ADC0 | N/A | | |
| 000001 | ADC1 | | | |
| 000010 | ADC2 | | | |
| 000011 | ADC3 | | | |
| 000100 | ADC4 | | | |
| 000101 | ADC5 | | | |
| 000110 | ADC6 | | | |
| 000111 | ADC7 | | | |
| 001000[(1)] | N/A | ADC0 | ADC0 | 10× |
| 001001[(1)] | | ADC1 | ADC0 | 10× |
| 001010[(1)] | | ADC0 | ADC0 | 200× |
| 001011[(1)] | | ADC1 | ADC0 | 200× |
| 001100[(1)] | | ADC2 | ADC2 | 10× |
| 001101[(1)] | | ADC3 | ADC2 | 10× |
| 001110[(1)] | | ADC2 | ADC2 | 200× |
| 001111[(1)] | | ADC3 | ADC2 | 200× |
| 010000 | | ADC0 | ADC1 | 1× |
| ..... | | | | |
| 011110 | 1.1V ($V_{BG}$) | N/A | | |
| 011111 | 0V (GND) | | | |
| 100000 | ADC8 | N/A | | |
| 100001 | ADC9 | | | |
| 100010 | ADC10 | | | |
| 100011 | ADC11 | | | |
| 100100 | ADC12 | | | |
| 100101 | ADC13 | | | |
| 100110 | ADC14 | | | |
| 100111 | ADC15 | | | |

# ADC

## Starting the conversion

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| ADEN | ADSC | ADATE | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 | ADCSRA |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

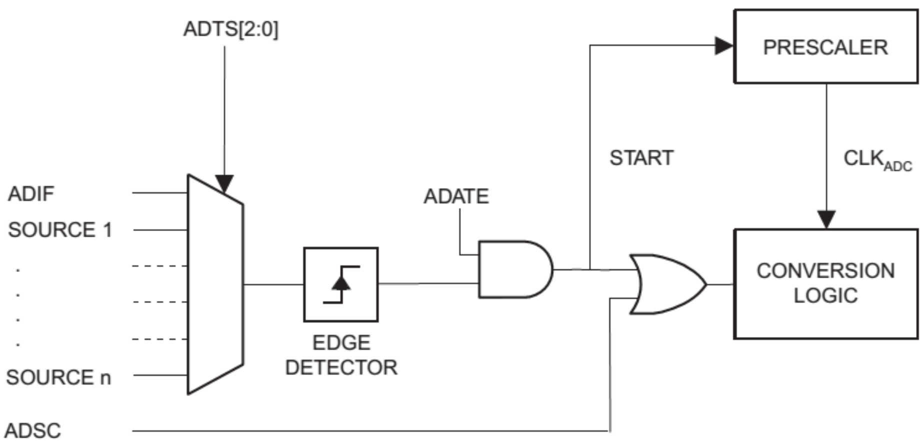
- At request by: ADSC =1 (remains set during conversion and is erased at the end of conversion)

- Automatically: ADATE = 1 (Auto Trigger Enable)

New conversion is started at the end of current conversion (ADIF=1 – free running mode)

Other external sources for automated conversion triggering:

ADC Auto Trigger Logic



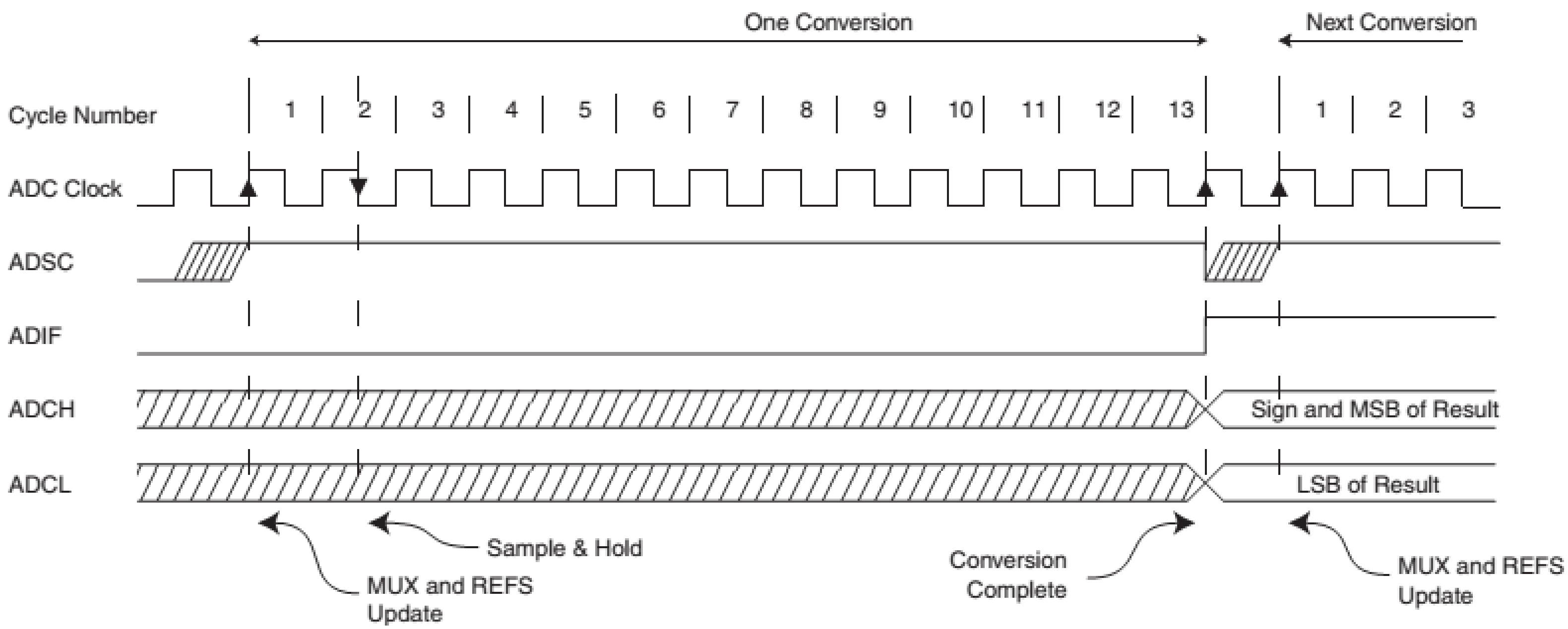

| ADTS2 | ADTS1 | ADTS0 | Trigger Source |
|---|---|---|---|
| 0 | 0 | 0 | Free Running mode |
| 0 | 0 | 1 | Analog Comparator |
| 0 | 1 | 0 | External Interrupt Request 0 |
| 0 | 1 | 1 | Timer/Counter0 Compare Match A |
| 1 | 0 | 0 | Timer/Counter0 Overflow |
| 1 | 0 | 1 | Timer/Counter1 Compare Match B |
| 1 | 1 | 0 | Timer/Counter1 Overflow |
| 1 | 1 | 1 | Timer/Counter1 Capture Event |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| – | ACME | – | – | MUX5 | ADTS2 | ADTS1 | ADTS0 | ADCSRB |
| R | R/W | R | R | R/W | R/W | R/W | R/W | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

# ADC

Conversion times, diagrams

ADC Timing Diagram, Single Conversion




ADC Conversion Time

| Condition | Sample & Hold (Cycles from Start of Conversion) | Conversion Time (Cycles) |
|---|---|---|
| First conversion | 13.5 | 25 |
| Normal conversions, single ended | 1.5 | 13 |
| Auto Triggered conversions | 2 | 13.5 |

# ADC

**Example 1: Digital thermometer**

Sensor: LM35 (http://www.ti.com/lit/ds/symlink/lm35.pdf)

Vout=T[°C] * 0.01[V]/ [°C]

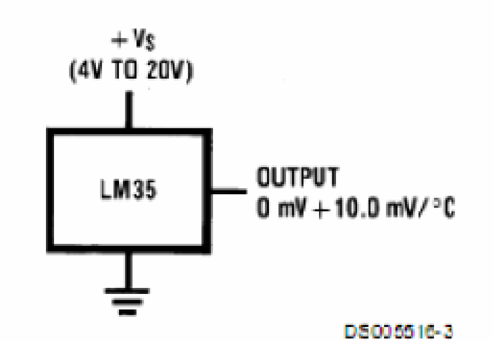
Single ended input mode: $ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$

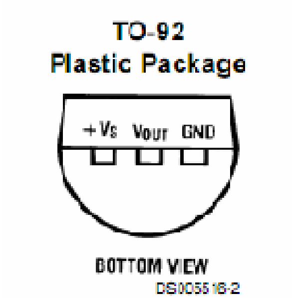
If ADLAR = 1 (low resolution): ADCH=Vin*256/Vref

ADCH = Vout*256/Vref

ADCH = T*2.56/Vref

If Vref = 2.56 V (internal voltage reference ⇒ ADCH = T [ºC]



# ADC

Example (measuring the temperature – ATmega2560):

```
rcall ADC_init
loop:
        rcall start_ADC_conversion ; Starts a conversion
        rcall wait_ADC_complete ; Wait to complete the current conversion
        rcall ADC_read ; Read the result in r16
        rjmp loop

ADC_Init:
        ldi r16, 0b11100011 ; Vref=2,56 V internal, ADLAR=1 (Data Shift left)   ADC3 single ended
        out ADMUX, r16
        ldi r16, 0b10000000 ; Activate ADC, max. speed (clock div. ratio = 2)
        out ADCSRA, r16
ret

start_ADC_conversion:
        sbi ADCSRA, ADSC ; ADC start, set ADSC bit in ADCSRA
ret

wait_ADC_complete:
        sbic ADCSRA, ADSC ; When ADSC=0, conversion is finished
        rjmp wait_ADC_complete
ret

ADC_read:
        in r16, ADCH ; ADCH – temperature on 8 bits
ret
```

# ADC

**Example 2: light brightness measurement**

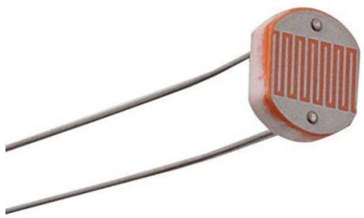
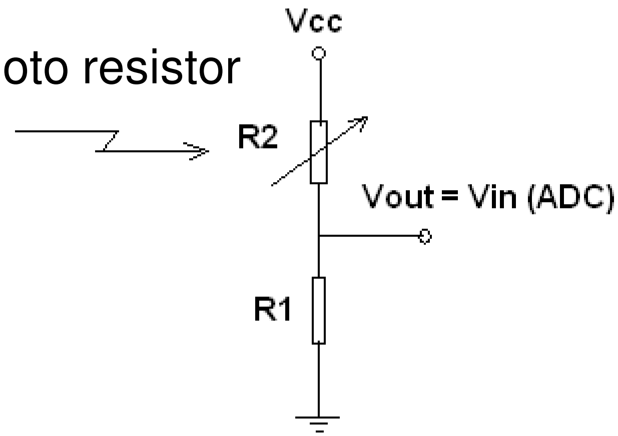


Photo resistor




R2 = 200 Ω (bright light) …. 1.4 MΩ (dark)

R1 = constant (ex: 20 K)

$$V_{OUT} = V_{CC} \frac{R_1}{R_1 + R_2}$$

Single ended input mode, ADLAR = 1 (low resolution): ADCH=Vin*256/Vref

**Sensor calibration**:

- Measure ADCH for lowest light (dark): $ADCH_{MIN}$
- Measure ADCH for brightest light: $ADCH_{MAX}$

**Measurement:**

- Compute the light brightness B [%]: $$B[\%] = \frac{ADC - ADC_{MIN}}{ADC_{MAX} - ADC_{MIN}} * 100$$

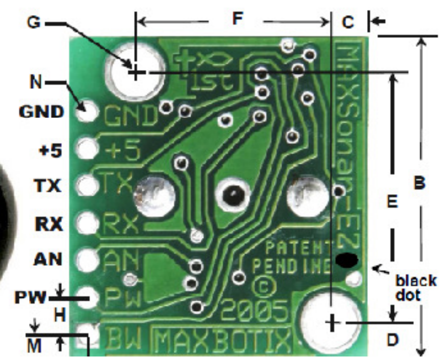
# ADC

**Example 3: distance (depth) measurement with a  US sensor**

LV-MaxSonar®-EZ0™ High Performance Sonar Range Finder (http://maxbotix.com/documents/LV-MaxSonar-EZ_Datasheet.pdf)

**AN –** Outputs analog voltage with a scaling factor of (Vcc/512) per inch. A supply  voltage of 3.3V yields ~6.4mV/in ≈ 2.56 mV/cm

Sonar range: 6-in (15 cm) … 254 in ( 645 cm) with 1-inch resolution. For objects from 0 .. 6in range as 6-inches.

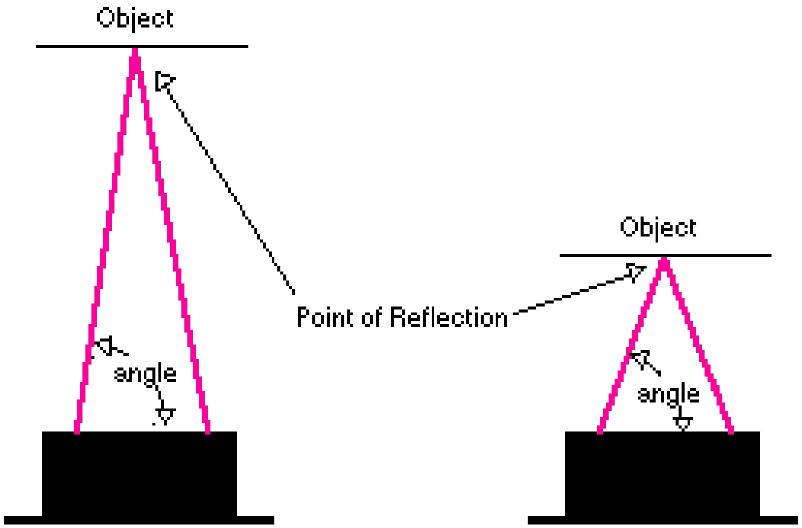

$$ADC = \frac{V_{IN} \times 1024}{V_{REF}} = \frac{2.56mV \times d[cm] \times 1024}{2.56[V]} \approx d[cm]$$

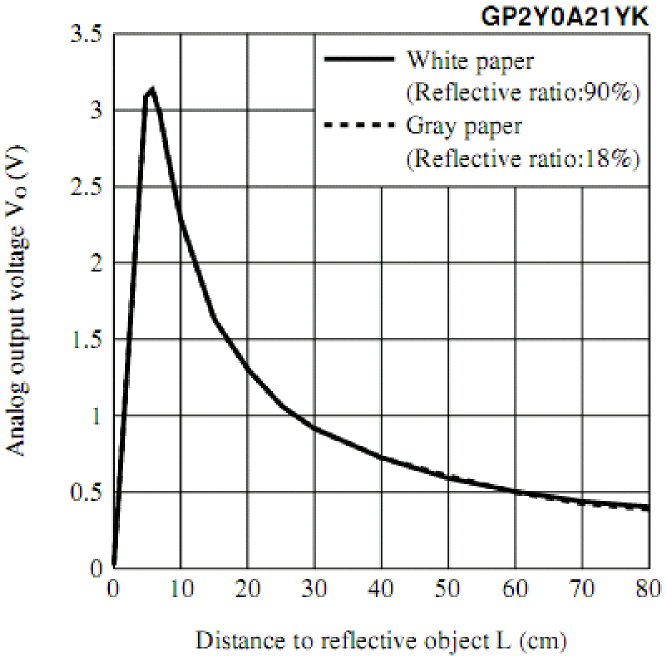
```
ADC_Init:
  ldi r16, 0b11000011 ; Vref=2,56 V internal, ADLAR=0 (Data Shift right - full
1024 bit resolution), ADC3 single ended
  out ADMUX, r16
  ldi r16, 0b10000000 ; Activate ADC, max. speed
  out ADCSRA, r16
ret
ADC_read:
  in r20, ADCL //ADC access to data registers is blocked
  in r21, ADCH //ADC access to the ADCH and ADCL Registers is re-enabled
  // r21:r20 = d[cm] (in r20 range = 15 cm …. 256 cm)
ret
```

# ADC

**SHARP GP2XX,** family of IR distance sensors

- Uses triangulation for distance computation
- Measures the angle for the reflected ray
- Analog output, **non linear**
- Low cost, easy to set up

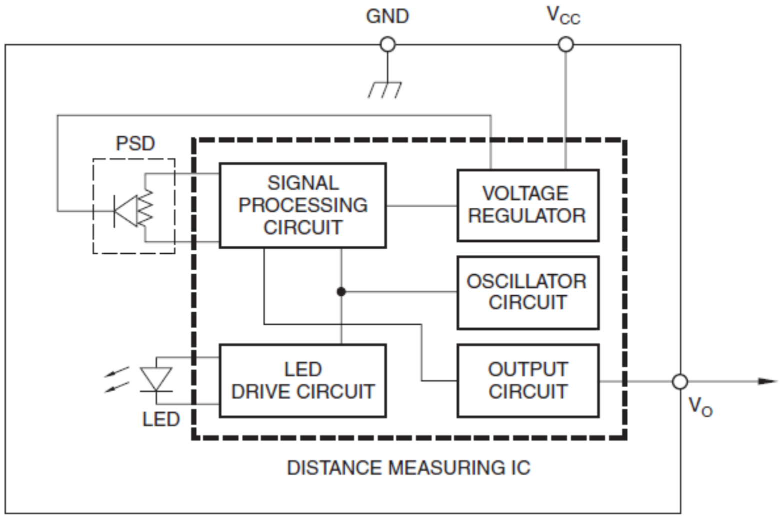



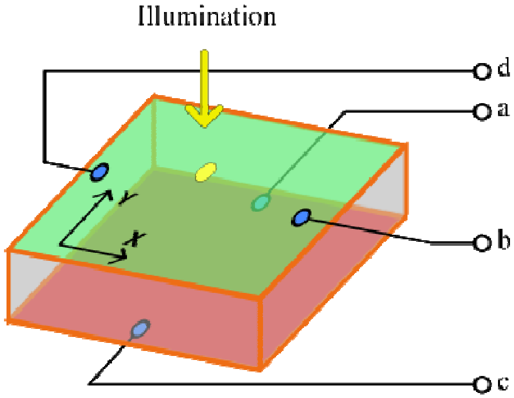



# ADC

**SHARP GP2XX,** family of IR distance sensors

- Based on position sensitive photo diodes
- Can measure the position of the incident light

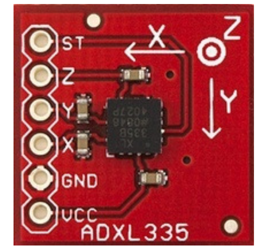

$$x = k_x \cdot \frac{I_b - I_d}{I_b + I_d}$$

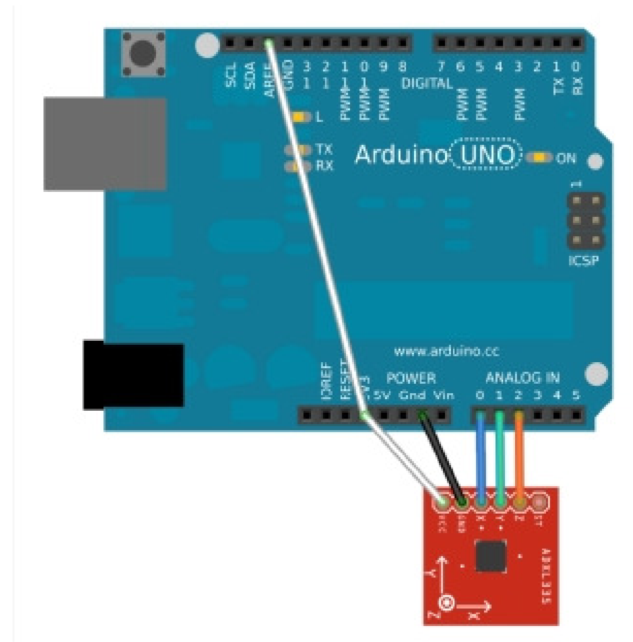
$$y = k_y \cdot \frac{I_a - I_c}{I_a + I_c}$$

# ADC

**ADXL335 Accelerometer**

- Measures acceleration on 3 axes, from -3 to 3 g
- Power at 1.8 … 3.6 V
- Output for 0 G: Vcc/2
- Typical sensitivity for Vcc=3.3V: 300 mv/G



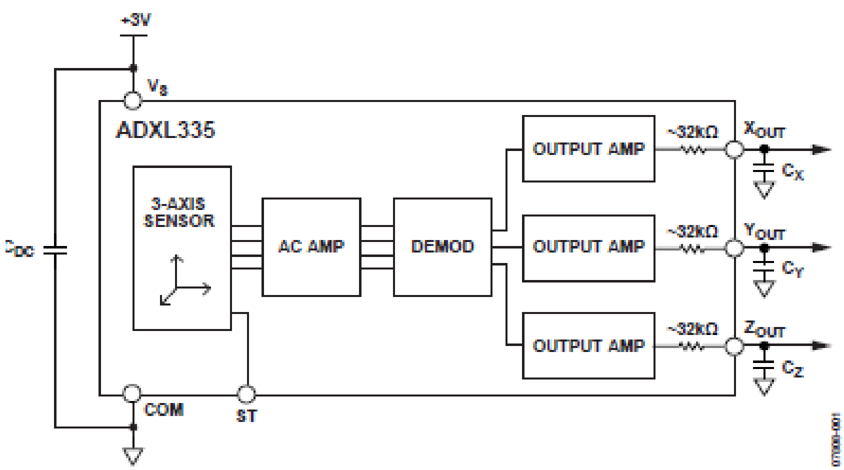
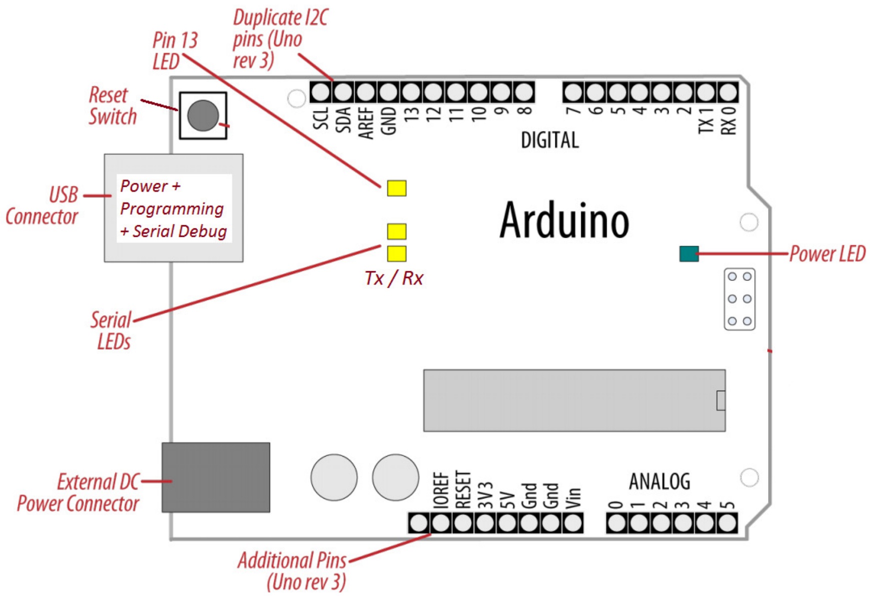




Figure 1.

| | |
|---|---|
| Arduino 3.3 V | ADXL335 VCC |
| Arduino GND | ADXL335 GND |
| Arduino Analog0 | ADXL335 X |
| Arduino Analog1 | ADXL335 Y |
| Arduino Analog2 | ADXL335 Z |
| Arduino 3.3 | Arduino AREF |

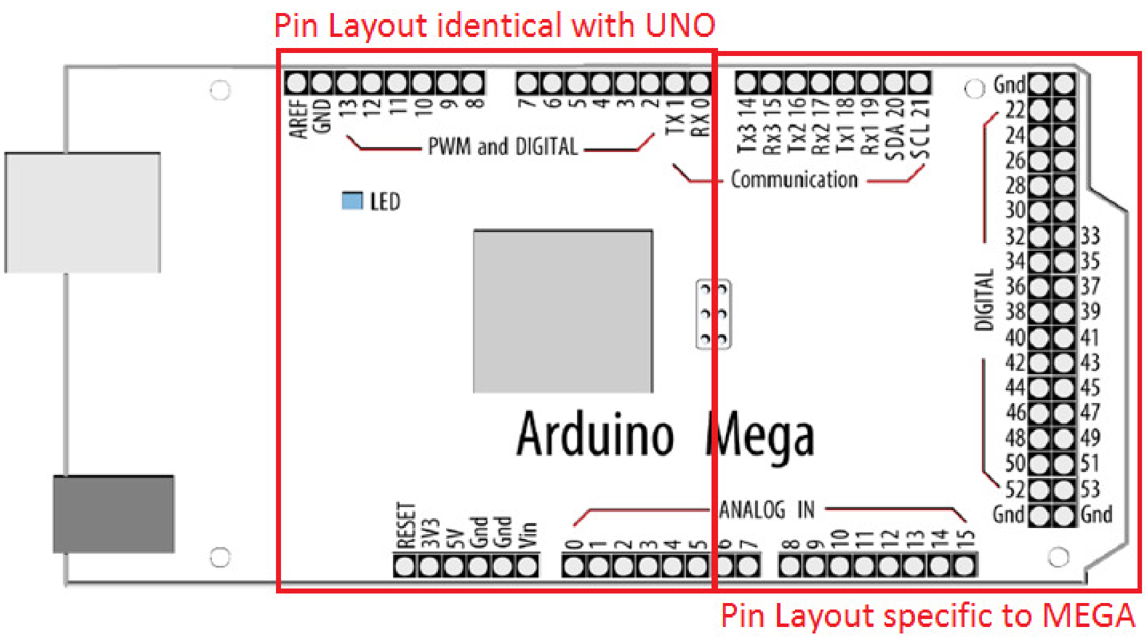
# Analog signal processing with Arduino




**Arduino UNO:** A0 .. A5

**Arduino MEGA:** A0 .. A15

- Analogue pins are inputs for the 10 bit resolution ADC of the µC.
- The ADC has 10 bit resolution, returning integers from 0 to 1023




**Other pins**

**AREF** (in) – external ref. voltage for the ADC

**IOREF** (out) – ref. voltage for shields

# Analog signal processing with Arduino

- **Analog pins main function**: **read analog values**
- Analog pins have also the functionality of general purpose input/output (GPIO) pins (the same as digital pins)

  pinMode(A0, OUTPUT);

  digitalWrite(A0, HIGH);

- Analog pins also have pullup resistors, which work identically to pullup resistors on the digital pins. They are enabled by issuing a command such as:

  digitalWrite(A0, HIGH); // set pullup on A0 while the pin is an *input*.

Turning on a pullup will affect the values reported by analogRead() !!!

**Methods**

**analogRead**(pin) - reads the value from the specified analog pin

**analogReference**(type) - configures the reference voltage used for analog input (i.e. the value used as the top of the input range)

# Analog signal processing with Arduino

**analogReference(type)** – configures the reference voltage used for analog input (i.e. the value used as the top of the input RANGE).

**type** - reference to use:

- DEFAULT: the default analog reference of 5 volts (for UNO & MEGA)
- INTERNAL: a built-in reference, equal to 1.1 volts on UNO (*not available on the Arduino Mega*)
- INTERNAL1V1: a built-in 1.1V reference (*Arduino Mega only*)
- INTERNAL2V56: a built-in 2.56V reference (*Arduino Mega only*)
- EXTERNAL: the voltage applied to the AREF pin (**0 to 5V only**) is used as the reference.

After changing the analog reference, the first few readings from analogRead() may not be accurate !!!

**Don't use anything less than 0V or more than 5V for external reference voltage on the AREF pin! If you're using an external reference on the AREF pin, you must set the analog reference to EXTERNAL before calling analogRead().** Otherwise, you will short together the active reference voltage (internally generated) and the AREF pin, possibly damaging the microcontroller on your Arduino board !!!

# Analog signal processing with Arduino

int *digital_value* **analogRead**(*pin*) - reads the value from the specified analog pin

- This means that it will map input voltages between 0 .. RANGE volts into a integer values ***digital_value*** between 0 and 1023.
- This yields a reading ***resolution*** of: RANGE volts / 1024 units.
- For the DEFAULT reference (5V) this yields:

  ***resolutionADC*** = .0049 volts (4.9 mV) / unit.
- To convert the input *digital_value* to a voltage use:

  ***Voltage*** = ***resolutionADC*** * ***digital_value***
- To convert the *Voltage* to a physical value measured in [X] use:

  ***Measurement*** [X] = ***Voltage*** [V] / ***Sensor_resolution*** [V] / [X]
- It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

If the analog input pin is not connected to anything, the value returned by **analogRead()** will fluctuate based on a number of factors (e.g. the values of the other analog inputs, how close your hand is to the board, etc.) !!!

# Analog signal processing with Arduino

**Example a1** - Read the voltage generated by a potentiometer connected to an analog pin (http://arduino.cc/en/Reference/AnalogRead)

```
int analogPin = 3;                 // potentiometer wiper (middle terminal) connected to analog pin 3
                                   // outside leads to ground and +5V
int val = 0;                       // variable to store the value read
float voltage;                     // value converted to a voltage [mV]
float resolutionADC = 4.9;         // default ADC resolution [mV] / unit (for 5V reference)
void setup()
{
  Serial.begin(9600);     //  setup serial
}

void loop()
{
  val = analogRead(analogPin);     // read the input pin (default settings: 5V reference)
  voltage = val * resolutionADC;   // converts the digital input value into a voltage
  Serial.print(“Digital value = “);
  Serial.println(val);             //  the digital value from the ADC
  Serial.print(“Voltage [mV] = “);
  Serial.println(voltage);
}
```

# Analog signal processing with Arduino

**Temperature sensor using LM50 sensor** http://www.ti.com/lit/ds/symlink/lm50.pdf

The LM50 sensor features

- linear +10.0 mV/℃ = 0.01V/℃  Scale Factor (sensor resolution)
- –40℃ to +125℃ temperature range
- DC offset of +500 mV for reading negative temperatures

The LM50 sensor is included in the brick temperature sensor
http://www.robofun.ro/senzori/vreme/senzor-temperatura-brick

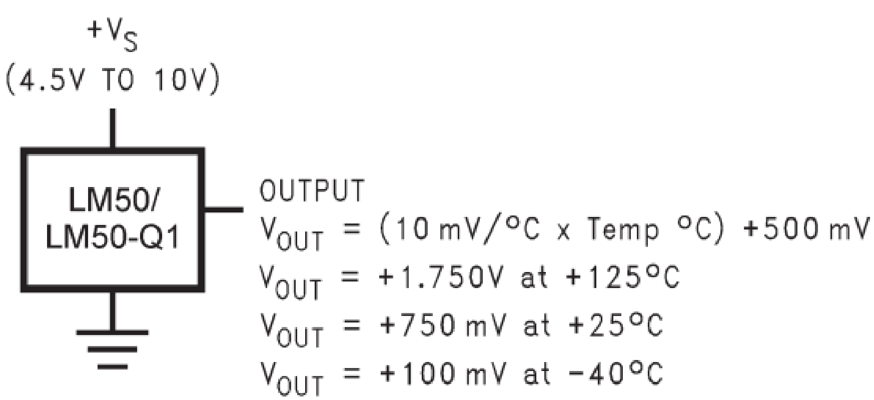
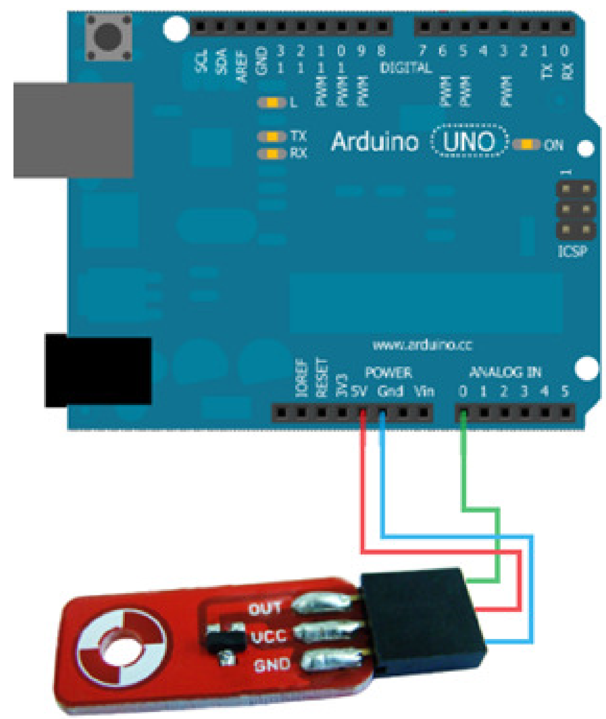



**Figure 2. Full-Range Centigrade Temperature Sensor Application (−40°C to +125°C)**



# Analog signal processing with Arduino

**Example a2** - Read the temperature from the Brick temperature sensor, average 10 consecutive readings and send it to the debug window

```
float resolutionADC  = .0049 ;        // default ADC resolution (for 5V reference) = 0.0049 [V] / unit
float resolutionSensor  = .01 ;       // Sensor resolution = 0.01V/°C

void setup()
{ Serial.begin(9600);
}

void loop(){
  Serial.print("Temp [C]: ");
  float temp = readTempInCelsius(10, 0);  // reads the average temperature over 10 consecutive readings
  Serial.println(temp);
  delay(200);
}

float readTempInCelsius(int count, int pin) {
// reads the average temperature over count consecutive readings from analogue pin
  float sumTemp = 0;
  for (int i =0; i < count; i++) {
     int reading = analogRead(pin);
     float voltage = reading * resolutionADC;
     float tempCelsius = (voltage - 0.5) / resolutionSensor ;  // substract DC offset  and convert to Celsius
     sumTemp = sumTemp + tempCelsius;
     }
  return sumTemp / (float)count;
}
```

# Analog signal processing with Arduino

**Example a3** – Measuring distances with the LV EZ0 sonar (10mV / inch ≅ 0.01V / inch resolution) http://maxbotix.com/documents/LV-MaxSonar-EZ_Datasheet.pdf

```
const int sensorPin = 1;              // Sonar analogue output connected to A1
float resolutionADC  = .0049 ;        // default ADC resolution (for 5V reference) = 0.0049 [V] / unit
float resolutionSensor  = .01 ;       // Sensor resolution = 0.01V/inch

void setup()
{ Serial.begin(9600);
}
void loop(){
  float distance = readDistance(10, sensorPin ); // reads the average distance over 10 readings
  Serial.print("Distance [inch]: ");  Serial.println(distance);
  Serial.print("Distance [cm]: ");  Serial.println(distance*2.54);
  delay(200);
}

float readDistance(int count, int pin) {
// reads the average distance [inch] over count consecutive readings from analogue pin
  float sumDist = 0;
  for (int i =0; i < count; i++) {
      int reading = analogRead(pin);
      float voltage = reading * resolutionADC;
      float distance = voltage / resolutionSensor;  // convert voltage to distance [inch]
      sumDist = sumDist + distance;
      }
  return sumDist / (float)count;
}
```