Building Interactive Devices and Objects

Prof. Dr. Michael Rohs, Dipl.-Inform. Sven Kratz

michael.rohs@ifi.lmu.de

MHCI Lab, LMU München

Today

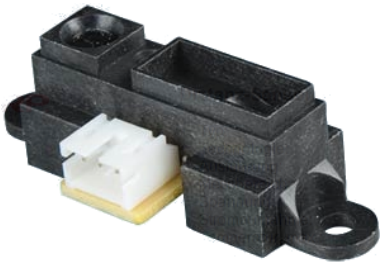
- I/O Ports and Buttons
- C for Microcontrollers
- Controlling Larger Currents
- Operational Amplifiers
- Sensors
- Analog-to-Digital Conversion
- USB-to-Serial Converter
- Exercise 3

Schedule

#	Date	Topic	Group Activity
1	19.4.2012	Session 1: Introduction	Team building
2	26.4.2012	Session 2: Microcontrollers & Electronics	
3	3.5.2012	Session 3: Sensors	Concept development
4	10.5.2012	CHI	Concept development
5	17.5.2012	Christi Himmelfahrt	Concept development
6	24.5.2012	Session 4: Actuators	Concept presentation, Hardware requ.
7	31.5.2012	Session 5: Physical Objects (Sven)	
8	7.6.2012	Frohnleichnam	Project
9	14.6.2012		Project
10	21.6.2012		Project
11	28.6.2012		Project
12	5.7.2012		Project
13	12.7.2012		Evaluation
14	19.7.2012		Evaluation, Presentation

Sessions 3: Sensors, Concept Development

- Sensors, opamps, A/D-conversion
  - Potentiometer, IR distance sensors, capacitive, accelerometer
- Exercises
  1. Read potentiometer level to control LED blink rate
  2. Connect USB breakout board
  3. Concept development

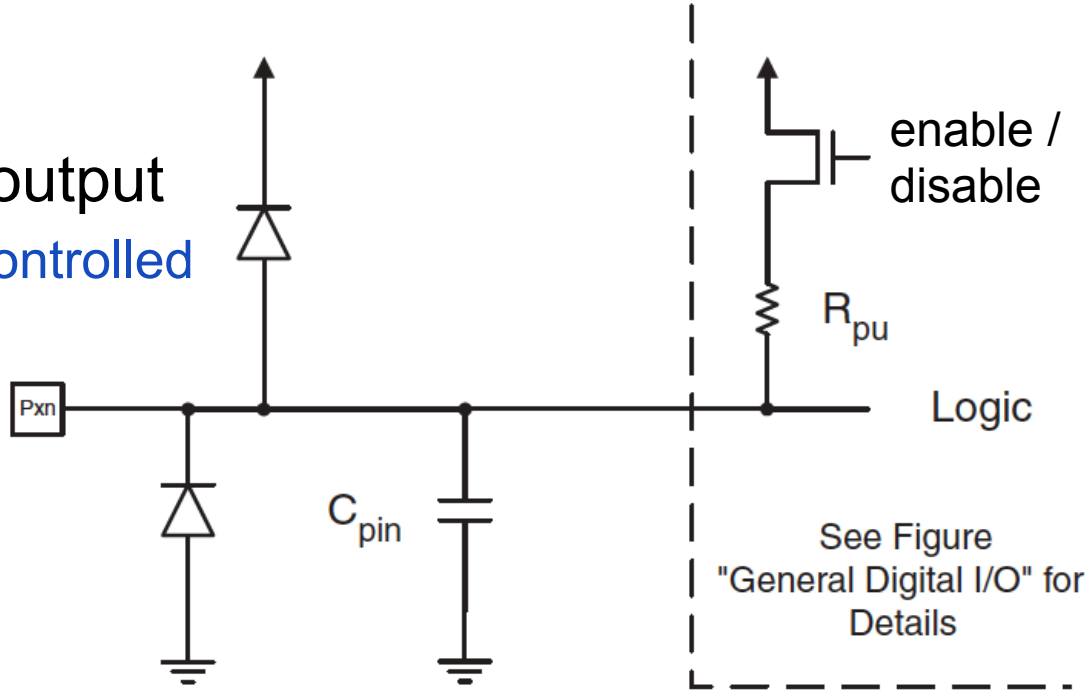


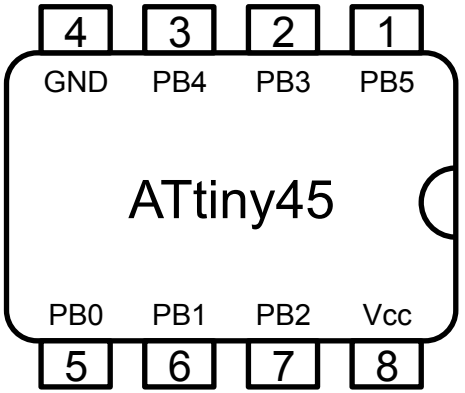
Sharp GP2-1080 distance sensor

I/O PORTS AND BUTTONS

AVR I/O Ports

- I/O pin either input or output
  - Individually software-controlled
- Pin as output
  - States: low, high
  - Can drive 40mA (→ LED)
- Pin as input
  - Internal pull-up resistor (enabled/disabled in software)
  - high resistance state (high-Z) if pull-up disabled

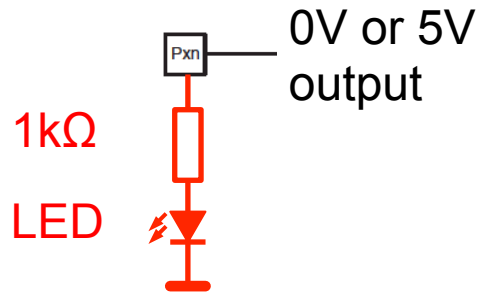




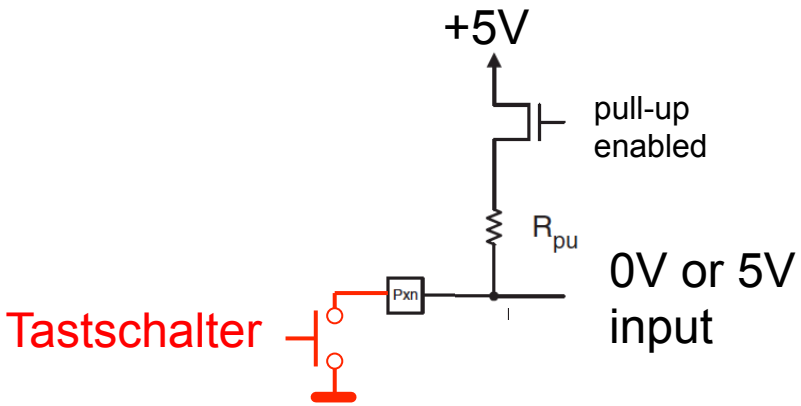
Accessing the I/O Ports

- 3 memory addresses for each I/O port
  - Data Direction Register: DDRx
    - 1 = output
    - 0 = input
  - Data Register: PORTx
    - if output: 1 = PIN driven high, 0 = PIN driven low
    - if input: 1 = pull-up enabled, 0 = pull-up disabled
  - Port Input Pins: PINx
    - read: PIN state (independent of DDRx)
    - write 1: toggles PORTx

Output:



Input (pull-up enabled):



C FOR MICROCONTROLLERS

“The” C Book

- Kernighan/Ritchie:
  The C Programming Language

- Imperative programming language
- Syntax resembles Java
  - .java → .h (declarations), .c (definitions)
  - no classes, no garbage collection, pointers
- C-Compiler compiles each .c-file separately into a .o-file
- C-Linker combines multiple .o-files and libraries into one executable file
- make (and Makefile) controls this process

The Preprocessor

- Before compilation, performs textual replacements

  #<preprocessor directive> [<argument(s)>]

- Insertion of files (no duplicate checking)

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

- Definition of constants

```
#define MY_CONST 10
```

- Conditions (often in header files to avoid duplicates)

```
#ifndef MAIN_H_
#define MAIN_H_
unsigned char myFunction(unsigned char x);
#endif
```

.h: Header files contain declarations
.c: Source files contain definitions

- test.h (declarations)

```
#ifndef TEST_H_
#define TEST_H_
unsigned char test(unsigned char i);
#endif
```

- test.c (definitions, implementation)

```
#include "test.h”
unsigned char test(unsigned char i) {
      return 5 * i;
}
```

C Data Types (for ATtiny, ATmega)

- signed char, 1 byte, -128..127
- unsigned char, 1 byte, 0..255
- signed int, **2 bytes**, -32768..32768
- unsigned int, **2 bytes**, 0..65535
- signed long, 4 bytes, -2^{31}-..2^{31}-1
- unsigned long, 4 bytes, 0..2^{32}-1

- ALU can only process 8-bits in parallel!
- avoid float and double!
  - emulated in software, large & slow

Bitwise Operations

- & and e.g. 0b11 & 0b01 = 0b01
- | or e.g. 0b10 | 0b01 = 0b11
- ^ xor e.g., 3^3 = 0
- << shift left e.g., 5<<3 means $5*2^3$
- >> shift right e.g., 5>>1 means $5/2^1$ (=2)
- ~ not e.g., ~1 means 0xfe

- very inexpensive operations

Declarations and Definitions of Variables

- Definitions reserves memory for the variable

  unsigned char sharedVariable = 3;
  (in main.c)

  static unsigned char hiddenVariable = 2;
  (in main.c, not accessible in other .c-files)

- Declarations describe the type but do not reserve memory

  extern unsigned char sharedVariable;
  (in test.c, refers to the variable in main.c)

- Similar use of **static** to limit accessibility of functions

  static unsigned char test(unsigned char i);
  (in main.c, not accessible in other .c-files)

Pointers

- Refer to a location in memory

```
int x[10];
int y;
int *p;
p = &x; // p points to first element (& = “address of”) x
y = *p; // y gets assigned x[0] (*p = “value at address p”)
*p = 0; // sets first element of x to 0
p = &x[1]; // p points to second element of x
```

- Pointer arithmetics

```
p = p + 1; // p points to next element
           // (pointer value incremented by sizeof(int)=2)
```

- There are also pointers to functions!

Structs

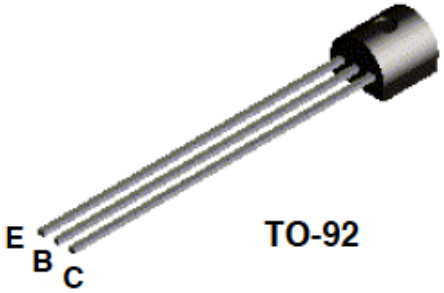
- Example

```
struct point {
        int x;
        int y;
}
```

- Usage

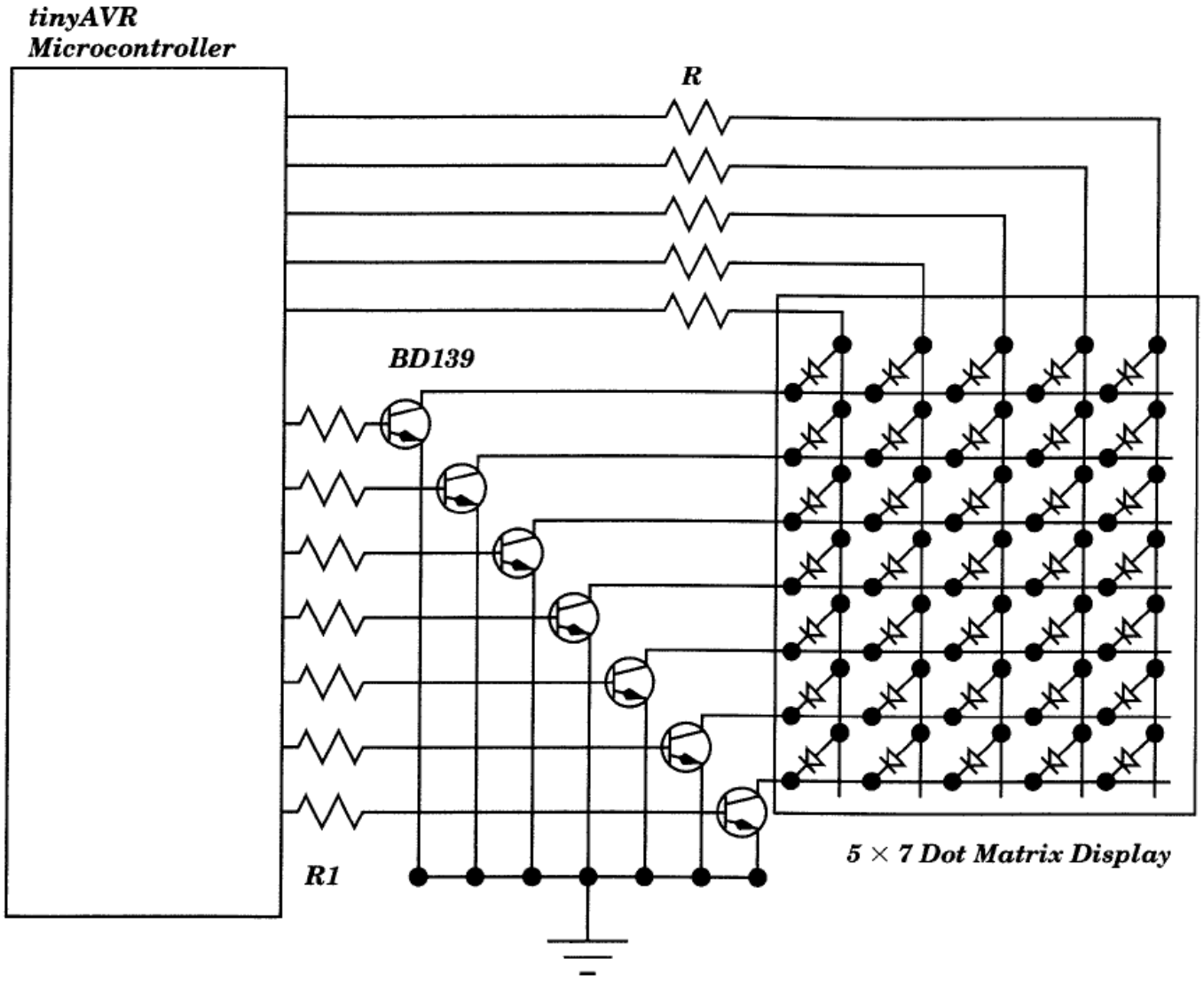
```
struct point pt;
pt.x = 320;
pt.y = 200;
```

- Structs are referenced by-value
  - example: myFunc(struct point pt) …
    puts 4 bytes (2 ints) on call stack



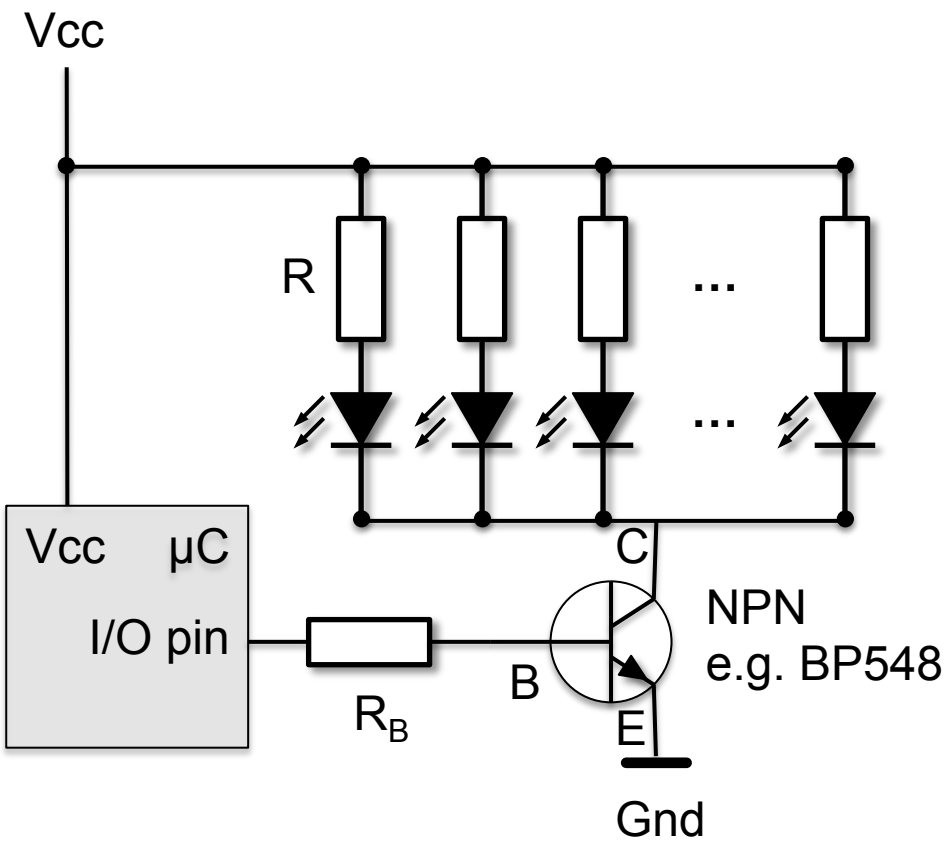
CONTROLLING LARGER CURRENTS

Multiplexing LEDs



Source: Gadre, Malhotra: tinyAVR Microcontroller Projects for the Evil Genios. McGraw-Hill, 2011.

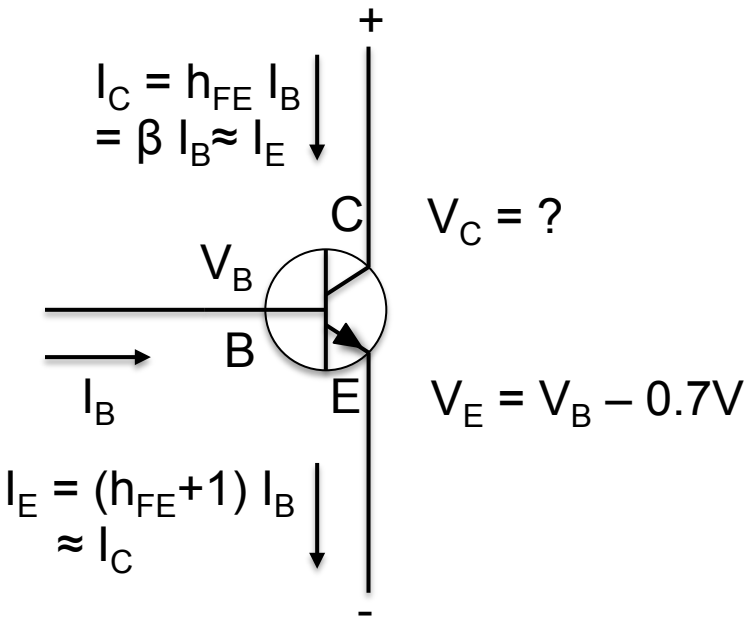
Driving More LEDs with a Transistor



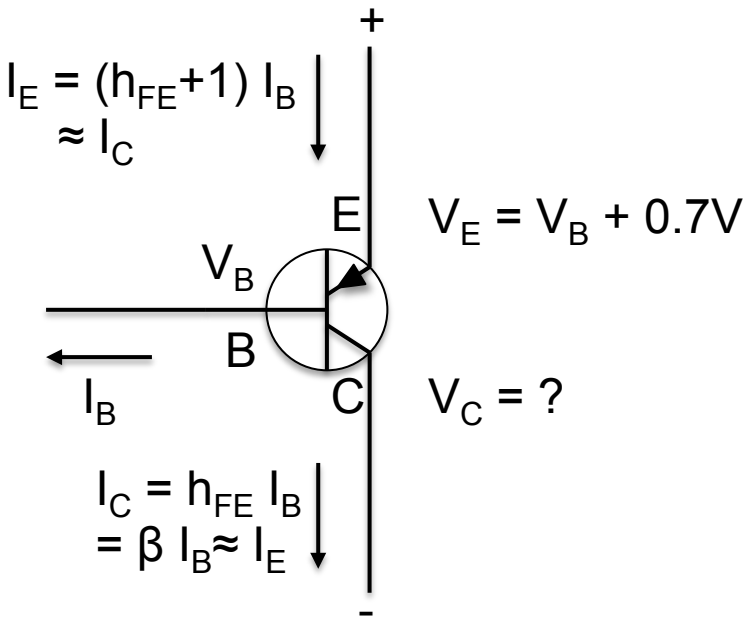
- Given
  - VCC = 5V
  - red LEDs: U_F=2V, I_F=20mA
  - BC548A: $I_{C,max}$ = 100mA, current gain β = h_{FE} = 110..220
- Drive 5 LEDs: 100mA
  - h_{FE} = 120 ($@I_C$=100mA,V_{CE}=5V)
  - $V_{CE(sat)}$ = 0.2V ($@I_C$=100mA,V_{CE}=5V)
  - $V_{BE(sat)}$ = 0.7V ($@I_C$=100mA,V_{CE}=5V)
- R? R_B?
  - U_R = Vcc - $V_{CE(sat)}$ – U_F
  - R = U_R / I_F
  - I_C = h_{FE} I_B

Behavior of Transistors (active region)

NPN (e.g. BC548)



PNP (e.g. BC558)



- $I_C = h_{FE}\ I_B = \beta\ I_B$
- $I_E = I_C + I_B$
- $I_E \approx I_C$

Typical Characteristics of BC548

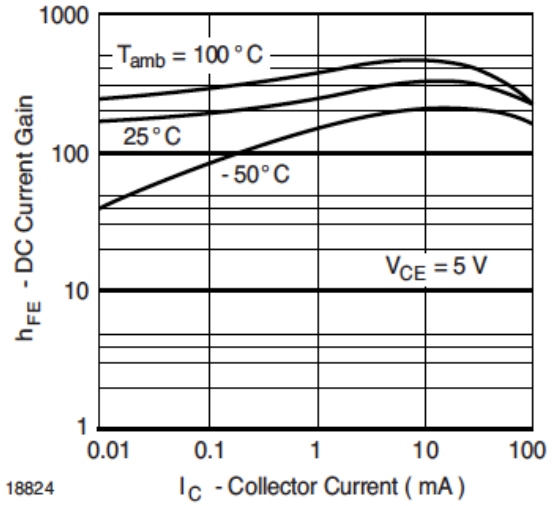


Figure 2. DC Current Gain vs. Collector Current

Source: Vishay Datasheet

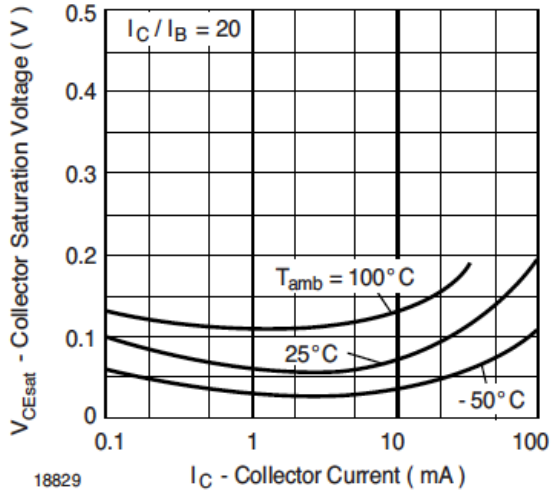
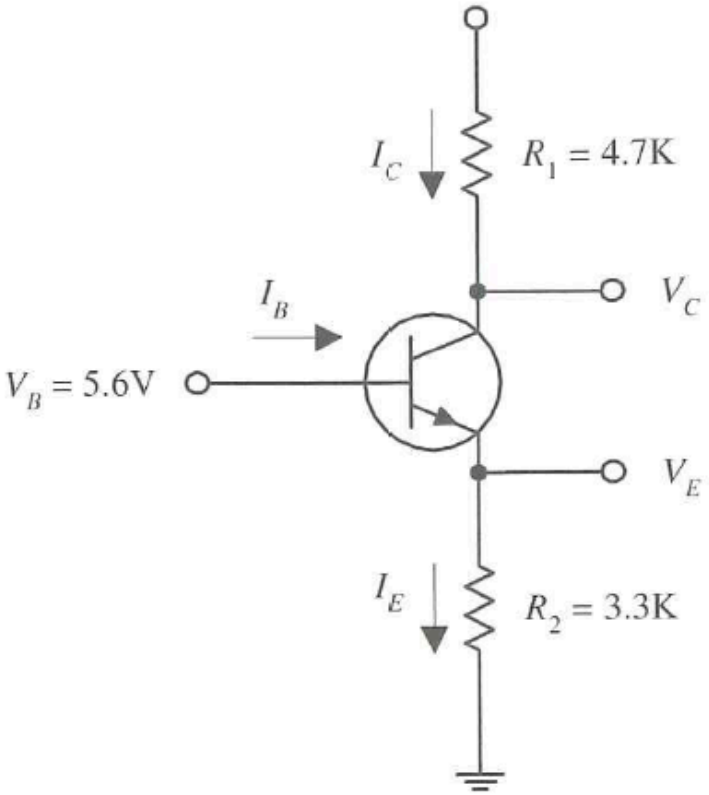


Figure 7. Collector Saturation Voltage vs. Collector Current

Source: Vishay Datasheet

- $I_C = h_{FE}\, I_B = \beta\, I_B$
- $I_E = I_C + I_B$
- $I_E \approx I_C$

EXAMPLE 1 Given $V_{CC} = +20\,\text{V}$, $V_B = 5.6\,\text{V}$, $R_1 = 4.7\ \text{k}\Omega$, $R_2 = 3.3\ \text{k}\Omega$, and $h_{FE} = 100$, find V_E, I_E, I_B, I_C, and V_C.



$$V_E = V_B - 0.6\,\text{V}$$

$$V_E = 5.6\,\text{V} - 0.6\,\text{V} = 5.0\,\text{V}$$

$$I_E = \frac{V_E - 0\,\text{V}}{R_2} = \frac{5.0\,\text{V}}{3300\ \Omega} = 1.5\,\text{mA}$$

$$I_B = \frac{I_E}{(1 + h_{FE})} = \frac{1.5\,\text{mA}}{(1 + 100)} = 0.015\,\text{mA}$$

$$I_C = I_E - I_B \approx I_E = 1.5\,\text{mA}$$

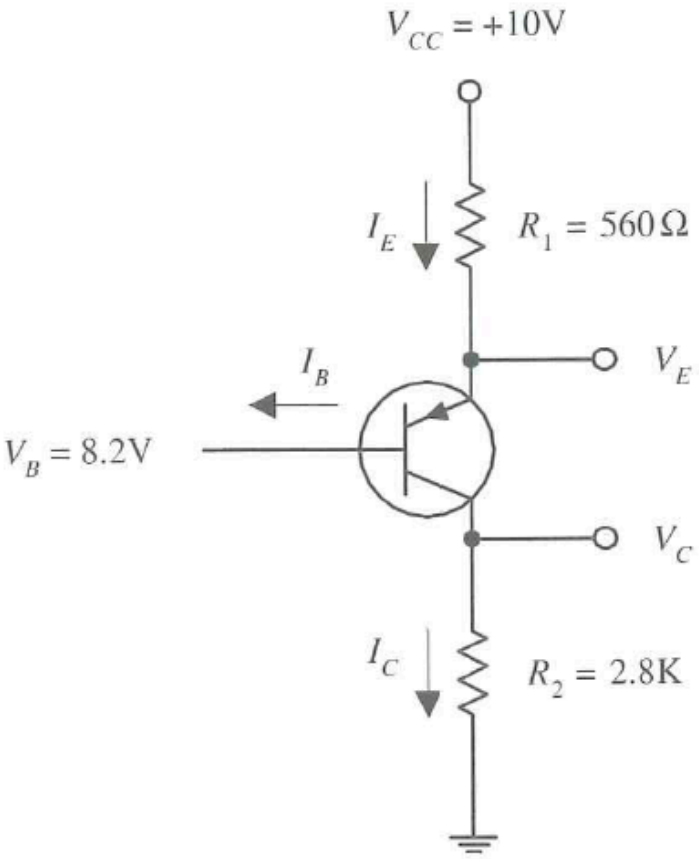
$$V_C = V_{CC} - I_C R_1$$

$$V_C = 20\,\text{V} - (1.5\,\text{mA})(4700\ \Omega)$$

$$V_C = 13\,\text{V}$$

Source: Paul Scherz: Practial Electronics for Inventors. 2nd edition, McGraw-Hill, 2007.

EXAMPLE 2 Given $V_{CC} = +10\,\text{V}$, $V_B = 8.2\,\text{V}$, $R_1 = 560\,\Omega$, $R_2 = 2.8\,\text{k}\Omega$, and $h_{FE} = 100$, find V_E, I_E, I_B, I_C, and V_C.



$$V_E = V_B + 0.6\,\text{V}$$

$$V_E = 8.2\,\text{V} + 0.6\,\text{V} = 8.8\,\text{V}$$

$$I_E = \frac{V_{CC} - V_E}{R_1} = \frac{10\,\text{V} - 8.8\,\text{V}}{560\,\Omega} = 2.1\,\text{mA}$$

$$I_B = \frac{I_E}{(1 + h_{FE})} = \frac{2.1\,\text{mA}}{(1 + 100)} = 0.02\,\text{mA}$$

$$I_C = I_E - I_B \approx I_E = 2.1\,\text{mA}$$

$$V_C = 0\,\text{V} + I_C R_2$$

$$V_C = 0\,\text{V} + (2.1\,\text{mA})(2800\,\Omega)$$

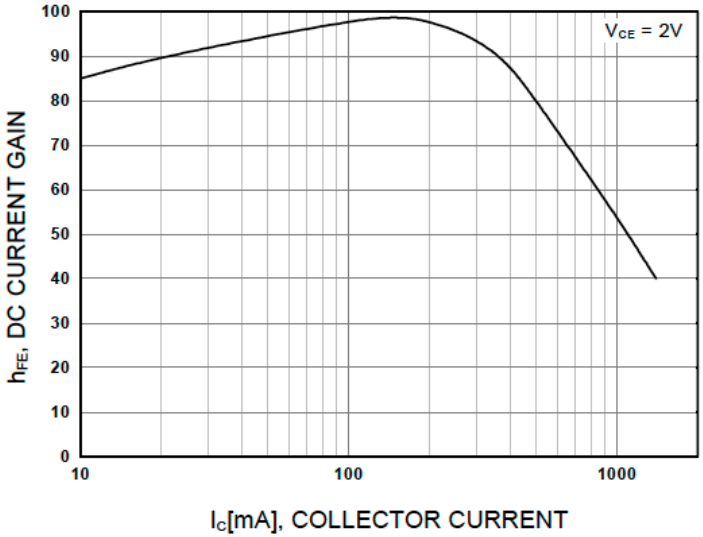
$$V_C = 5.9\,\text{V}$$

Source: Paul Scherz: Practial Electronics for Inventors. 2nd edition, McGraw-Hill, 2007.

Transistors (larger current)

- BD139 – NPN
  - maximum ratings: I_C = 1.5A, I_B = 0.5A
  - $V_{CE(sat)}$ = 0.5V, V_{BE} = 1V
  - h_{FE} = 63..160 @ I_C = 150mA, V_{CE} = 2V

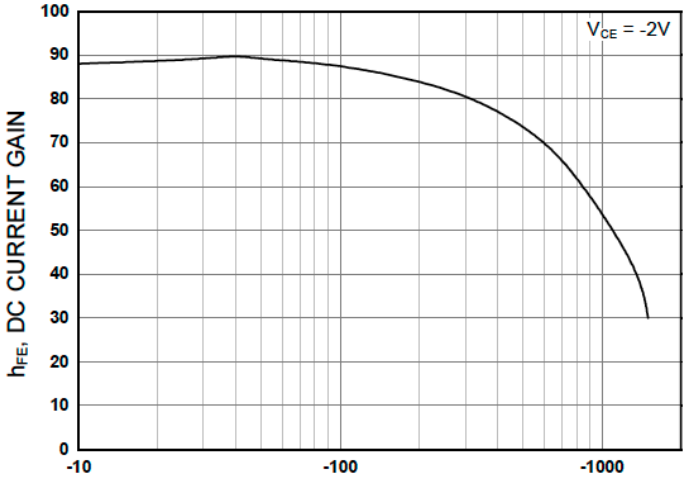
BD139 h_{FE}



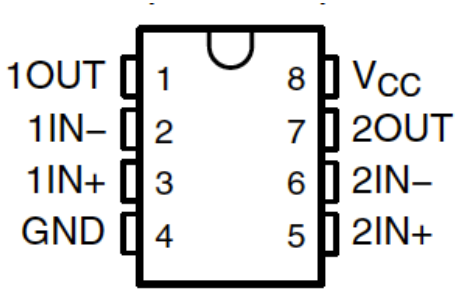
Source: Fairchild Datasheet

- BD140 – PNP
  - maximum ratings: I_C = -1.5A, I_B = -0.5A
  - $V_{CE(sat)}$ = -0.5V, V_{BE} = -1V
  - h_{FE} = 40..250 @ I_C = -150mA, V_{CE} = -2V

BD140 h_{FE}

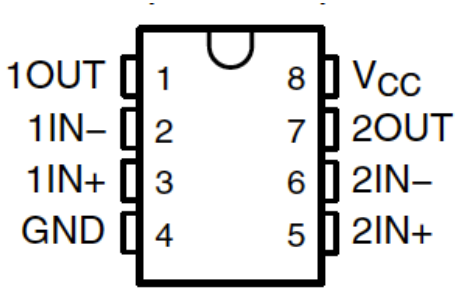


Source: Fairchild Datasheet



OPERATION AMPLIFIERS

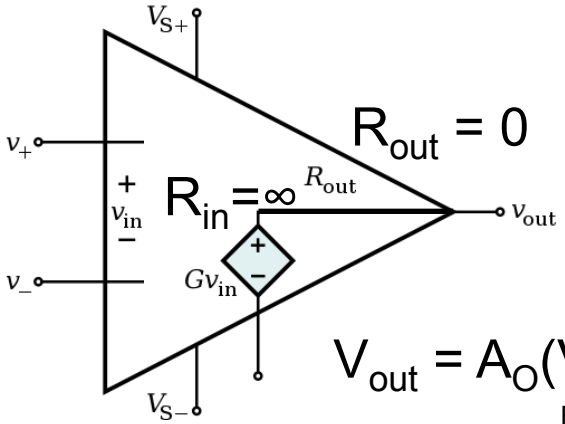
Operational Amplifiers



real op-amp

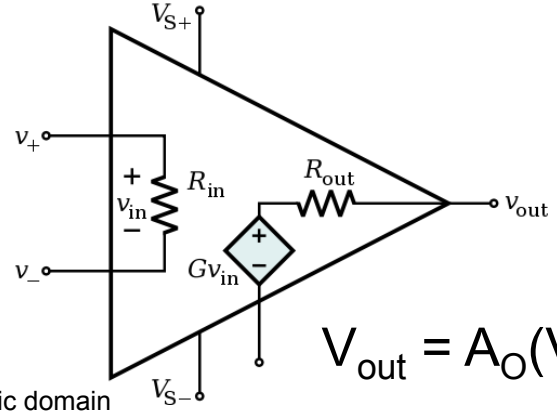
- Amplify signals (e.g. from sensors)
  - Differential amplifier
- Ideal op-amps
  - **Rule 1:** open-loop voltage gain: $A_O = \infty$
  - **Rule 2:** Infinite input impedance: $R_{in} = \infty$
  - **Rule 3:** Input terminals draw no current
  - *If negative feedback:* **Rule 4:** $V_+ - V_- = 0$

Ideal op-amp:



$V_{out} = A_O(V_+ - V_-)$

Real op-amp:



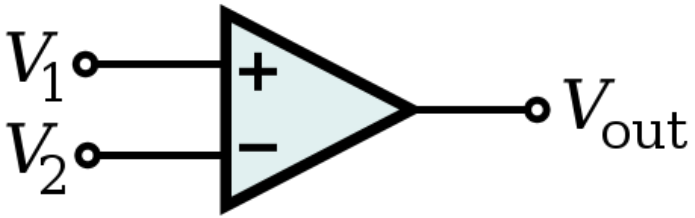
$V_{out} = A_O(V_+ - V_-)$

Figure author: Inductiveload, public domain

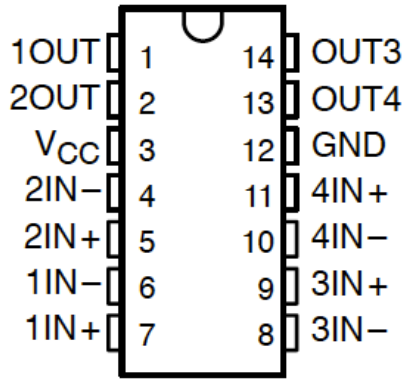
Op-Amps as Comparators

- Op-amp in open-loop setup (no negative feedback)
  - Ideal op-amp has open-loop voltage gain: $A_O = \infty$
  - Real op-amp voltage limited by supply voltage (V_{S-}, V_{S+})

$$V_{out} = \begin{cases} V_{S+} & V_1 > V_2 \\ V_{S-} & V_1 < V_2 \end{cases}$$



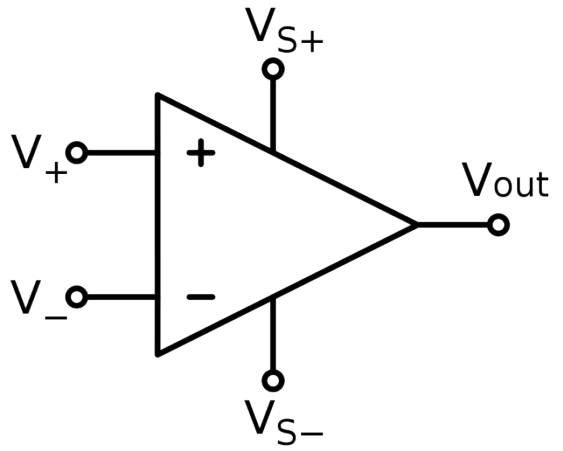
- Used for setting voltage thresholds
  - Example: V_2 set to fixed reference voltage (voltage divider, potentiometer), V_1 attached to sensor, V_{out} either 0 or V_{CC}
- Dedicated comparator chips
  - Example: LM339

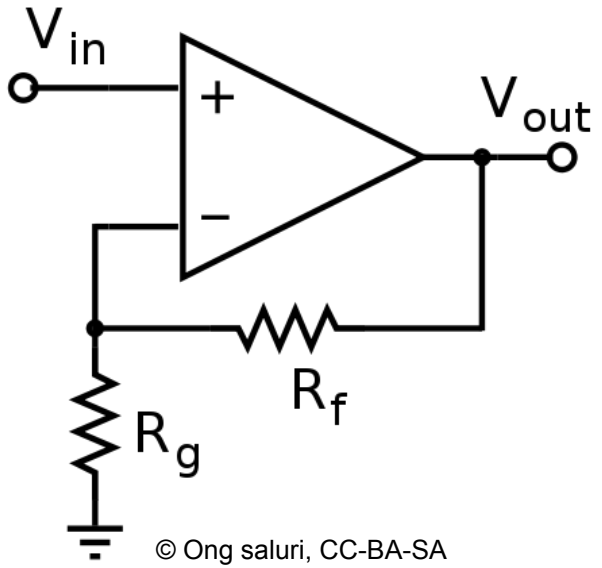


LM339, Source: TI Datasheet

Operational Amplifiers

- Op-amp circuit diagram
  - V_+: non-inverting input
  - V_-: inverting input
  - V_{out}: output
  - V_{S+}: positive power supply
  - V_{S-}: negative power supply
- Negative feedback
  - Rule 4: $V_+ - V_- = 0$
  - Closed-loop operation
  - Used to control gain (V_{out}/V_{in})





Unity Gain Buffer

- Gain: V_{out} / V_{in} = 1
- Benefits
  - high input impedance
  - low output impedance

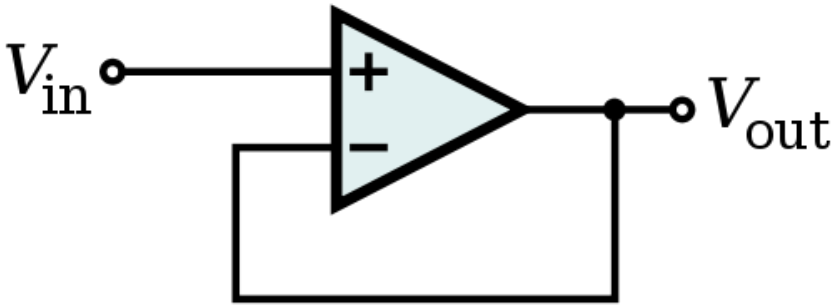


Figure author: Inductiveload, public domain

- High input impedance
  - does not draw much current from source
- Low output impedance
  - drives load like a good voltage source

Inverting Amplifier

- $V_+ = V_- = 0V$ (rule 4)
- $I = V_{in} / R_{in}$ (Ohm's law)
- $-I = V_{out} / R_f$ (Ohm's law, rule 3)
- Gain:
  $V_{out} / V_{in} = - R_f / R_{in}$

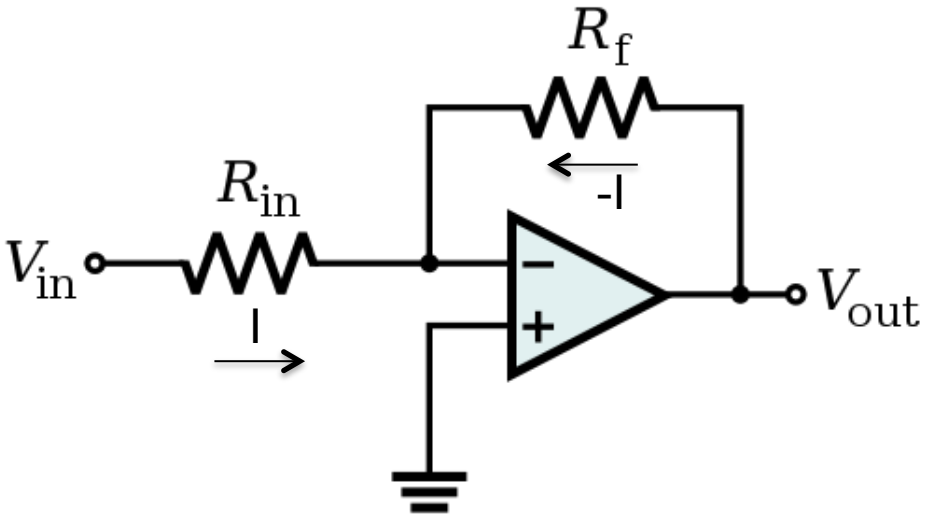


Figure author: Inductiveload, public domain

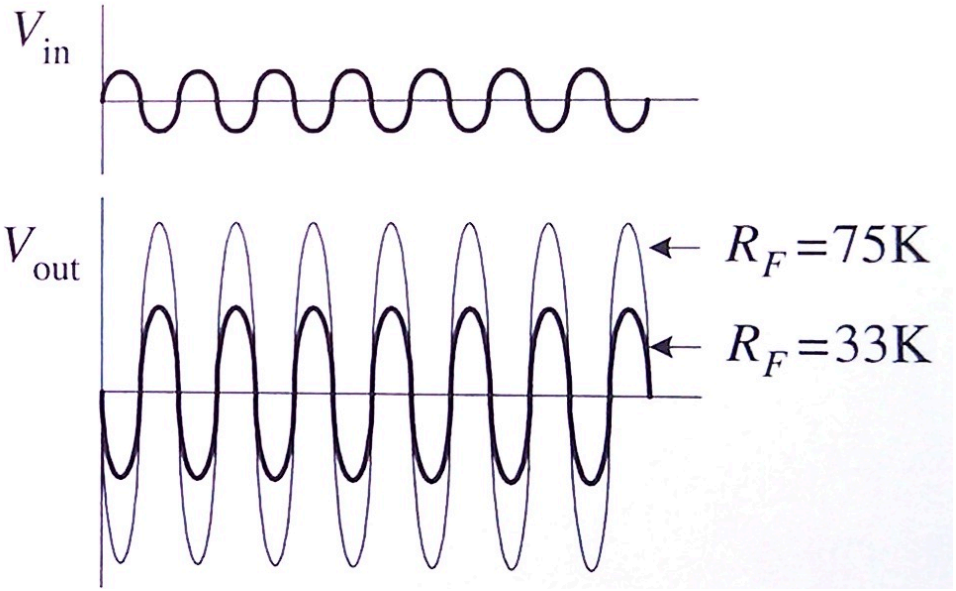


Figure: Paul Scherz: Practial Electronics for Inventors. 2nd edition, McGraw-Hill, 2007.

Non-Inverting Amplifier

- $V_{in} = V_+ = V_-$ (rule 4)
  $= R_1 / (R_1 + R_2) * V_{out}$
  (Ohm's law, rule 3)
- Gain:
  $V_{out} / V_{in} = (R_1 + R_2) / R_1$
  $= 1 + R_2 / R_1$

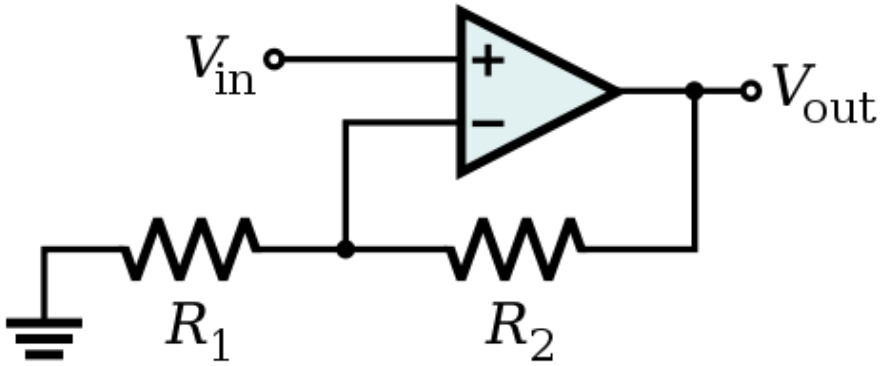


Figure author: Inductiveload, public domain

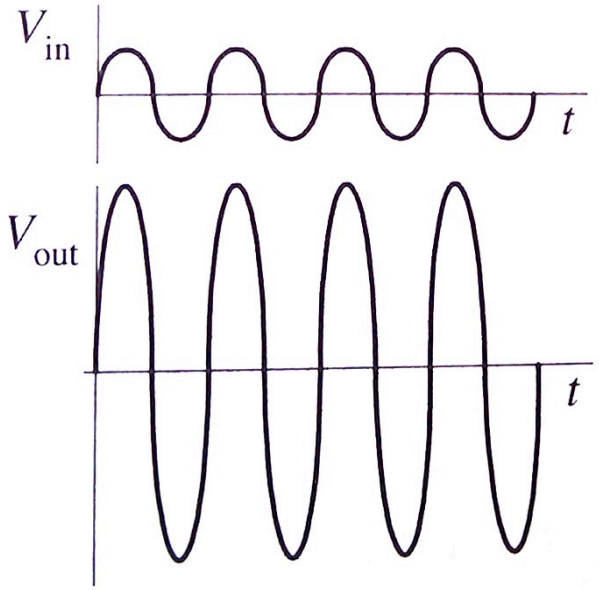
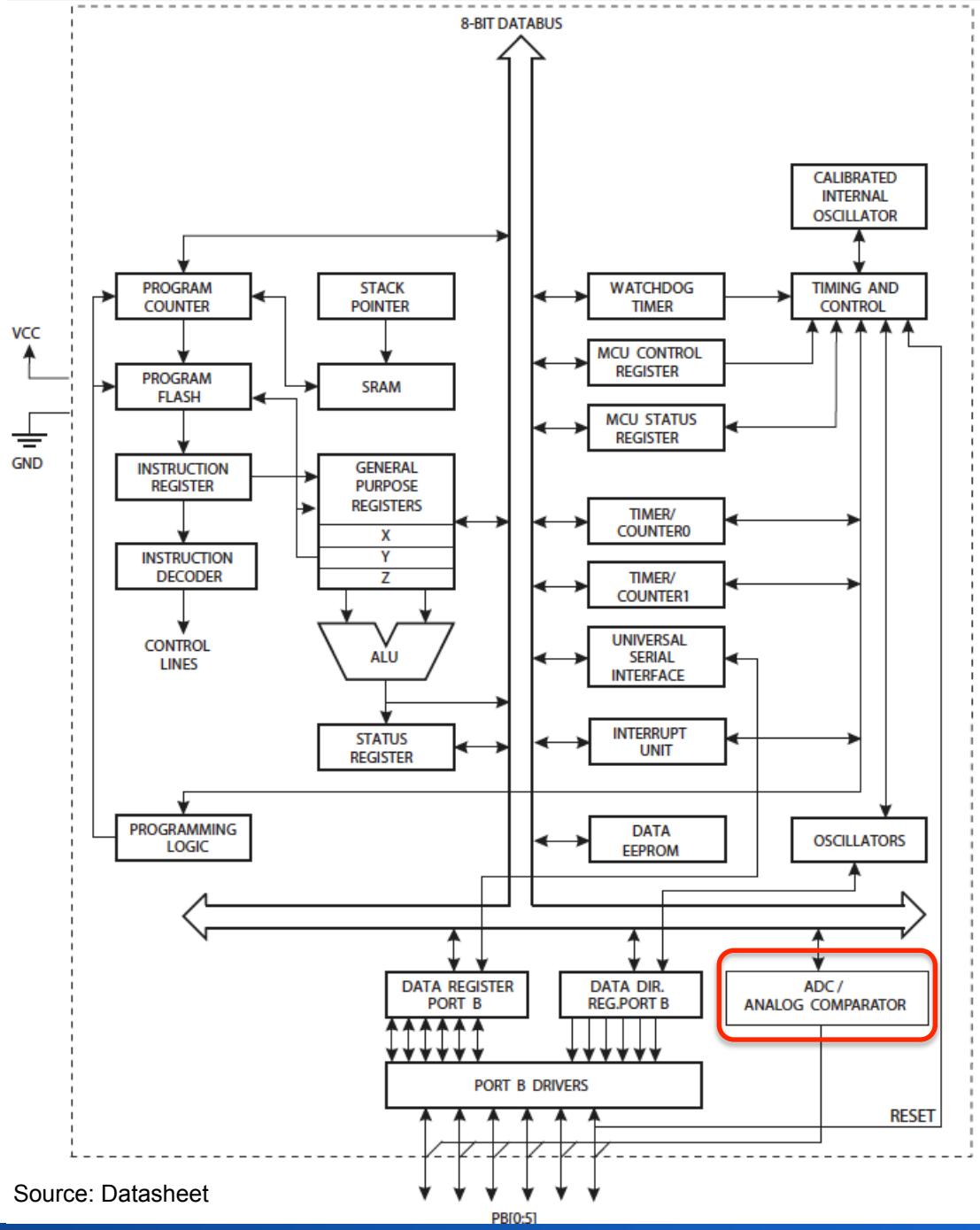


Figure: Paul Scherz: Practial Electronics for Inventors. 2nd edition, McGraw-Hill, 2007.

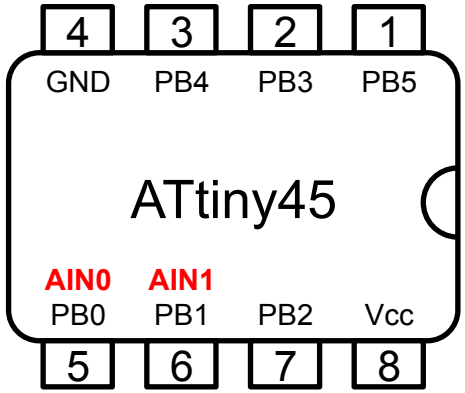
AVR ATtiny45 Architecture

- Analog Comparator

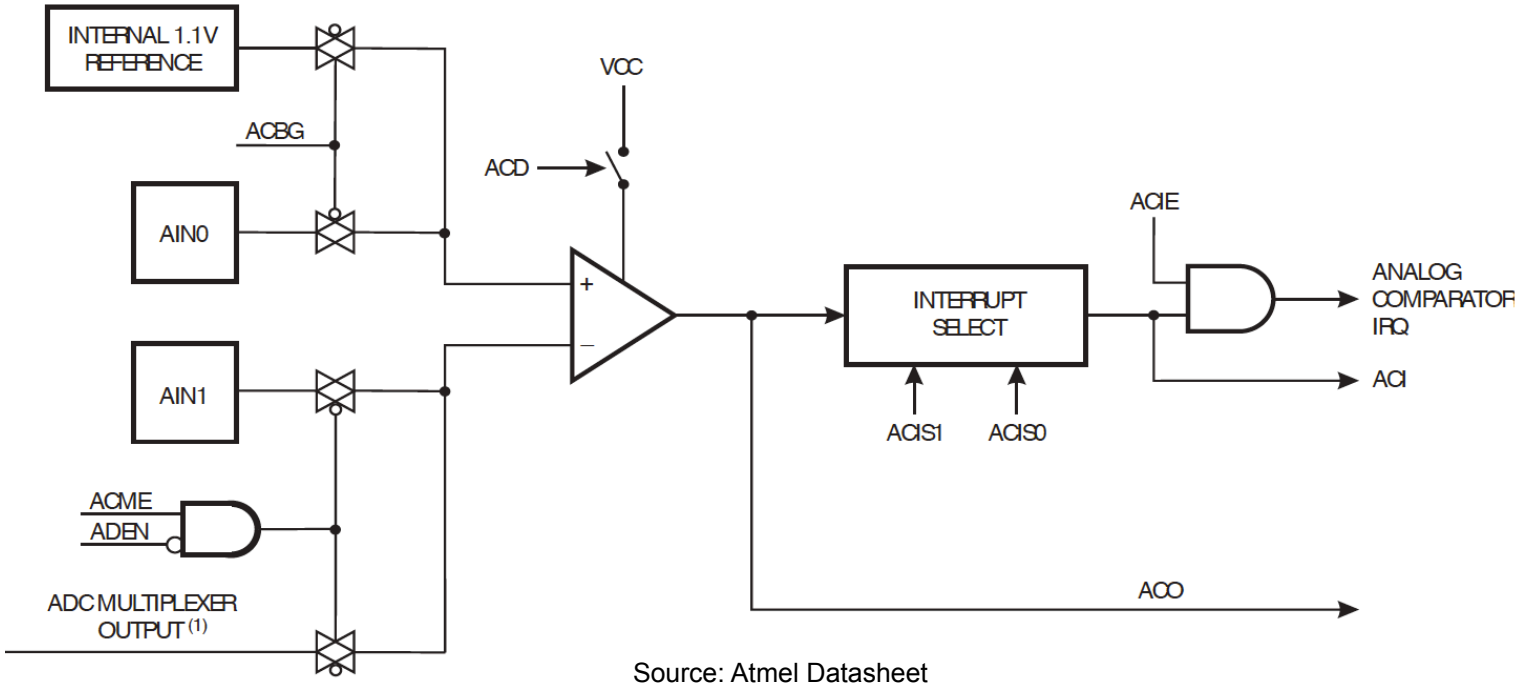


Source: Datasheet

ATtiny45 Analog Comparator



- Compares inputs AIN0 (+) and AIN1(-)
- Sets ACO if AIN0 > AIN1
  - Can trigger interrupt (on ACO rise, fall, or toggle)
- Registers control comparator: ADCSRB, ACSR, DIDR0



Source: Atmel Datasheet

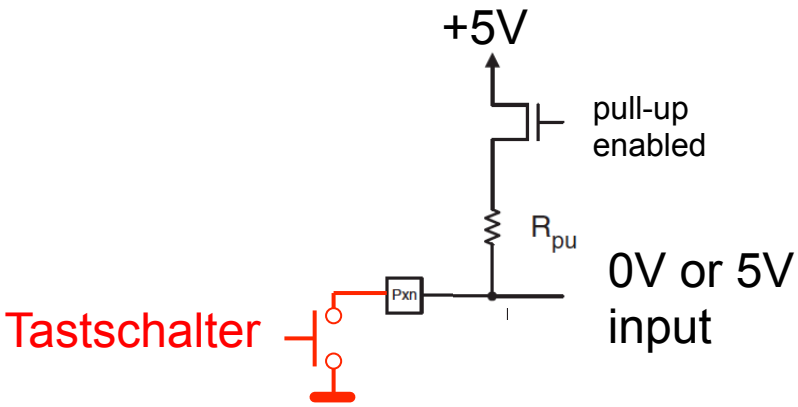
SENSORS

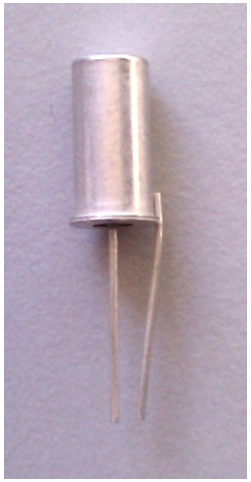
Encoding of Sensor Values

- Binary: GND-level or V_{CC}
  - simple
- Continuous voltage: 0..V_{CC}
  - requires A/D conversion
- Frequency
  - requires counters
- I2C bus
  - requires implementation of bus protocol
- etc.

Buttons and Switches

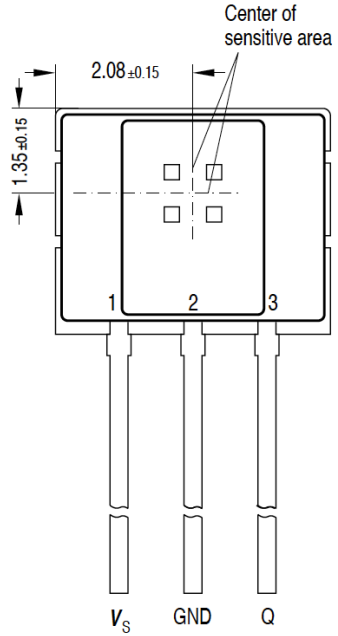
- Push buttons
  - Closed while pressed
- Toggle buttons
  - Change state when pressed
- Switches
  - Off position, on position
- Tilt Switches
  - Contains conducting ball contact
  - Contact closes based on gravity





Hall Sensor (TLE 4905)

- Senses magnetic field: switch on, when magnet nearby
- Used in bike computers to count wheel revelations
- Principle: Magnetic field perpendicular to Hall sensor induces voltage
- V_S = 3.8..24V
- I_{out} = 100mA



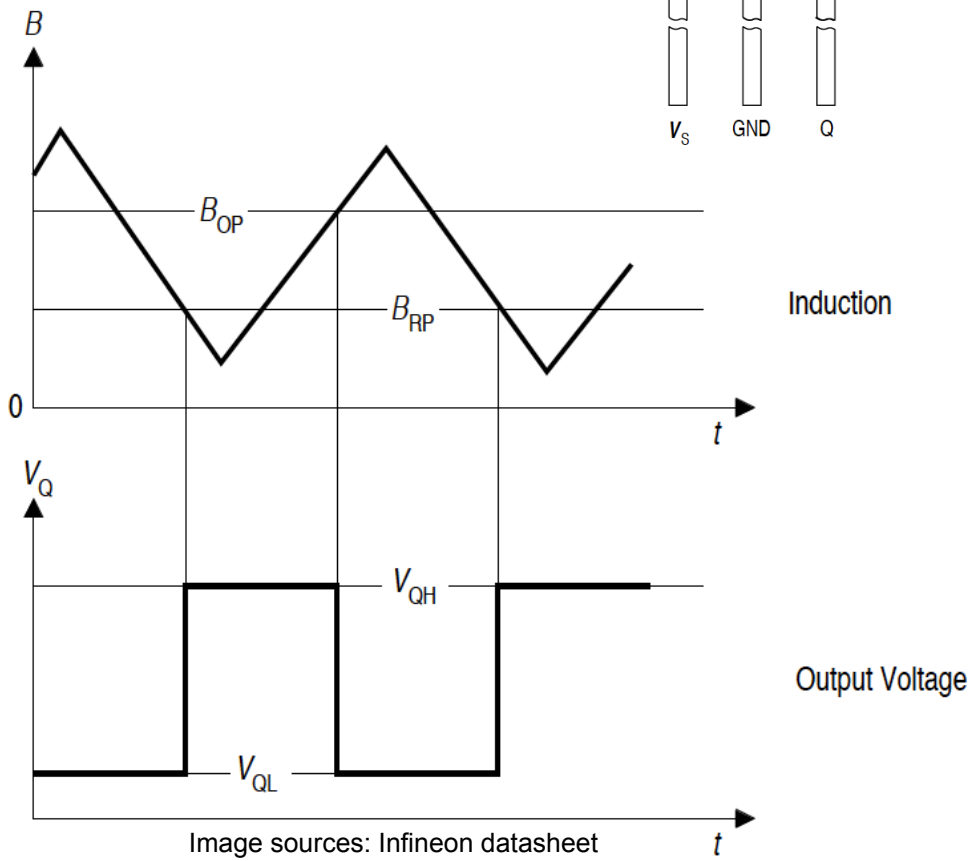
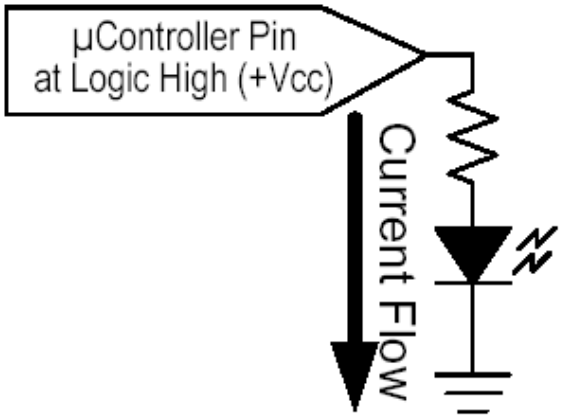


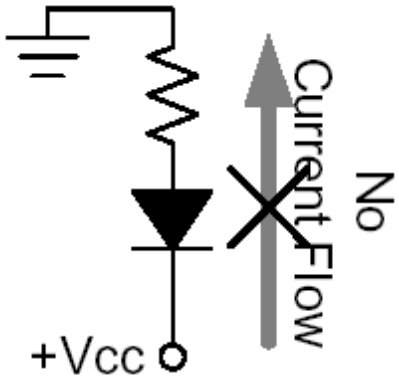
Image sources: Infineon datasheet

LEDs as Light Sensors

- LEDs are photo diodes
- Sensitive to light at the wavelength they emit
- Indirectly measure photo current
- Use LED under reverse bias conditions
  - Anode to ground
  - Cathode to +VCC



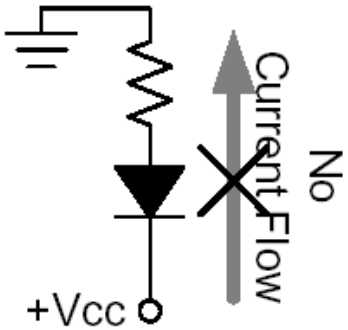
Normal operation



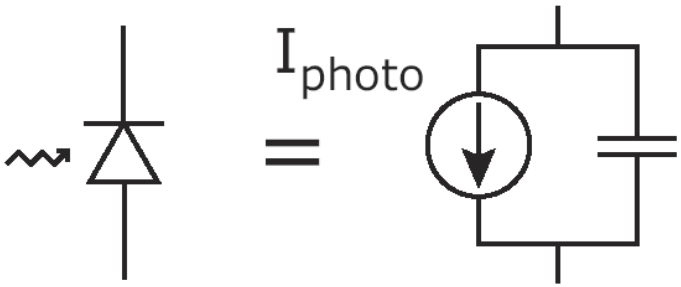
Reverse-biased LED

- Dietz, Yerazunis, Leigh: Very Low-Cost Sensing and Communication Using Bidirectional LEDs. UbiComp 2003, pp. 175-191.
- Hudson: Using Light Emitting Diode Arrays as Touch-Sensitive Input and Output Devices. UIST 2004, pp. 287-290.

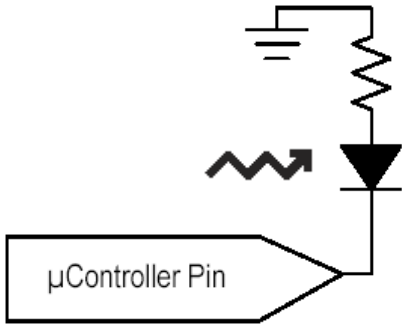
LEDs as Light Sensors



Reverse biased LED



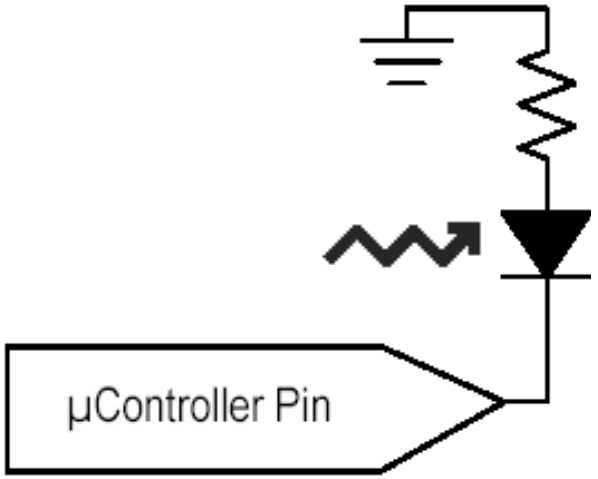
Modeled as capacitor in parallel with current source (photocurrent)



Circuit for light sensing

- Reverse biased LED = Capacitor parallel to photocurrent
- Sensing algorithm
  - Set microcontroller pin to high → capacitor charges
  - Switch pin to input mode → photo current discharges capacitor
  - Measure time until pin is low

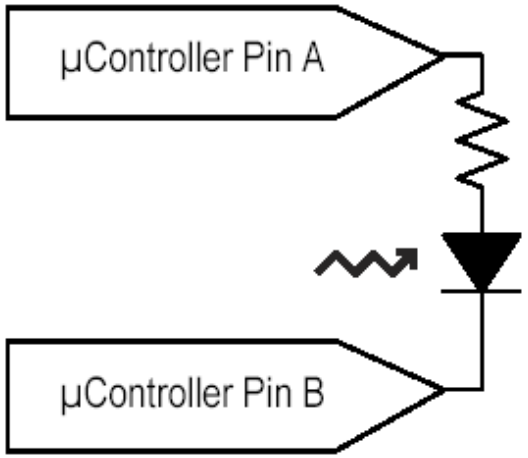
LEDs as Light Sensors




```
unsigned int senseLight() {
    // reverse bias LED
    DDRB  |= 0b00000001; // PB0 output
    PORTB |= 0b00000001; // PB0 high
    _delay_ms(10);

    // put in high-Z state
    DDRB  &= 0b11111110; // PB0 0 input
    PORTB &= 0b11111110; // PB0 low (turn off pull-up resistor)

    // count until pin drops to low
    unsigned int time = 0;
    while ((PINB & 1) == 1 && time < 0xffff) {
        time++;
    }
    return time;
}
```

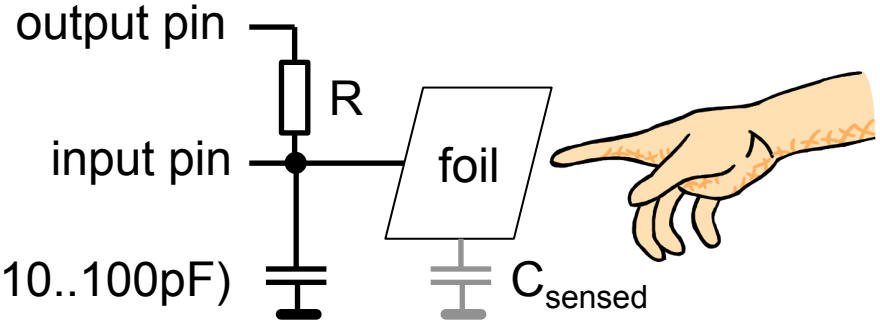
Exercise: Alternate between Emitting and Sensing Light

```
unsigned int senseAndEmitLight() {

   // …

    return time;
}
```



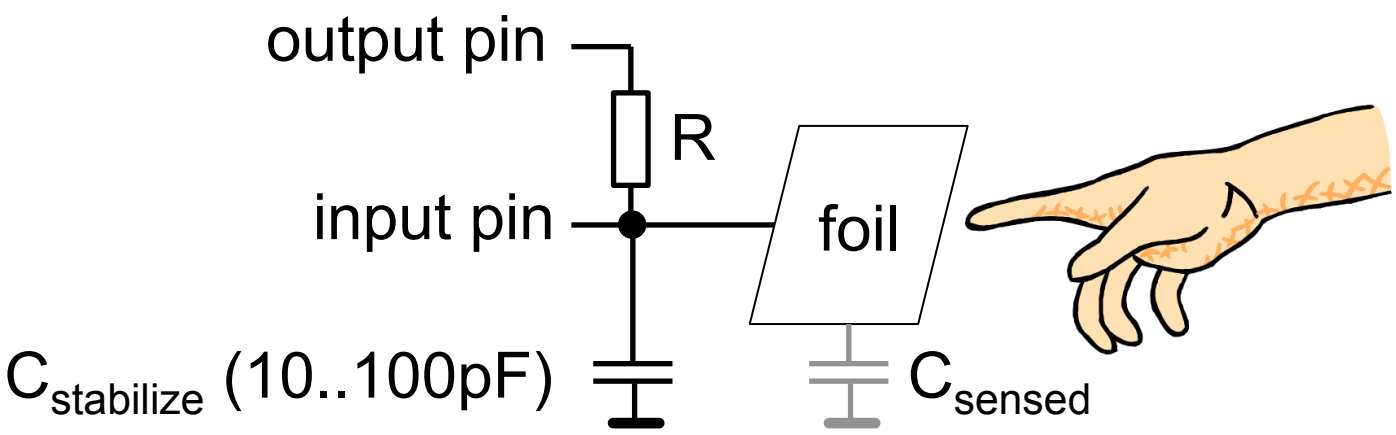
Capacitive Sensing

- Sense change in capacity of a capacitor
  - Modified by presence of hand/finger
  - Also depends on air humidity
- Senses hovering hand/finger
  - Zero force
  - Through (thin) insulating materials
- Components
  - Aluminum or copper foil
  - Large resistor (100kΩ..10MΩ)
- Algorithm
  - Change state of output pin
  - Measure delay in time of input pin (loop)



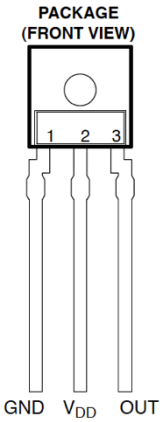
Capacitive Sensing

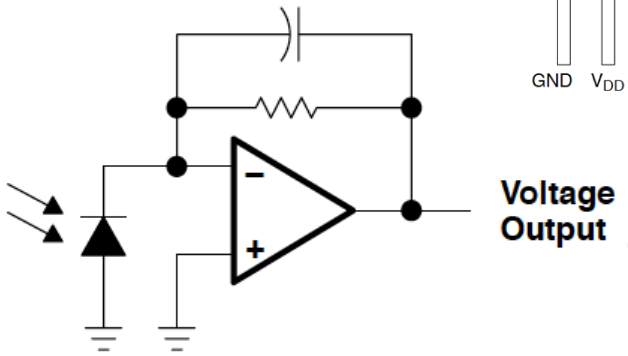
- Algorithm
  - Change state of output pin
  - Measure delay in time of input pin (loop)
- Principle
  - Capacitor is charged / discharged over time
  - Time depends on R and C_{sensed}

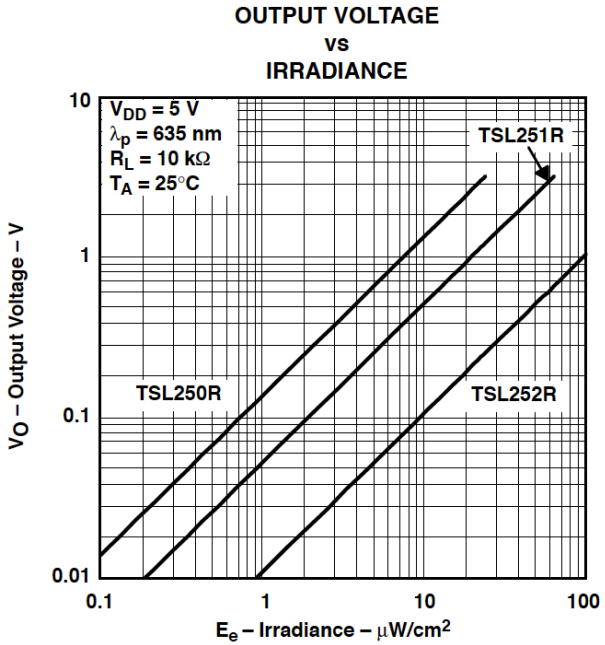


Light-to-Voltage Sensors (TSL250R)

- Converts light intensity to voltage
- Photodiode + OpAmp
- Supply-voltage range: 2.7..5.5V
- Supply current: 1.1 mA







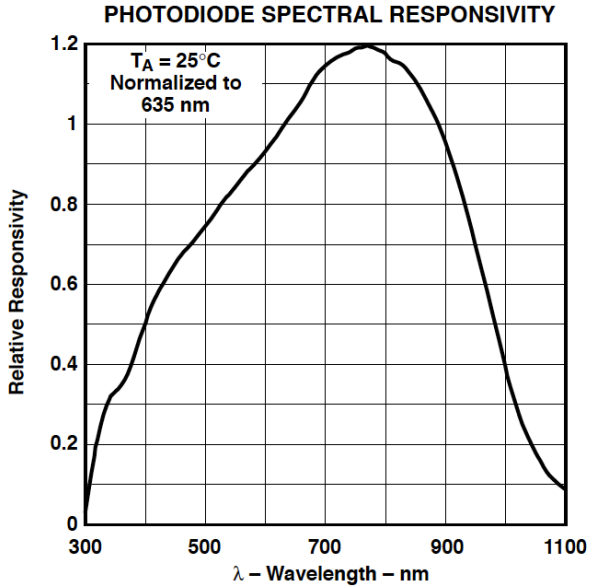
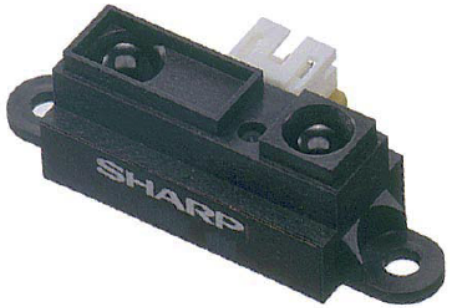
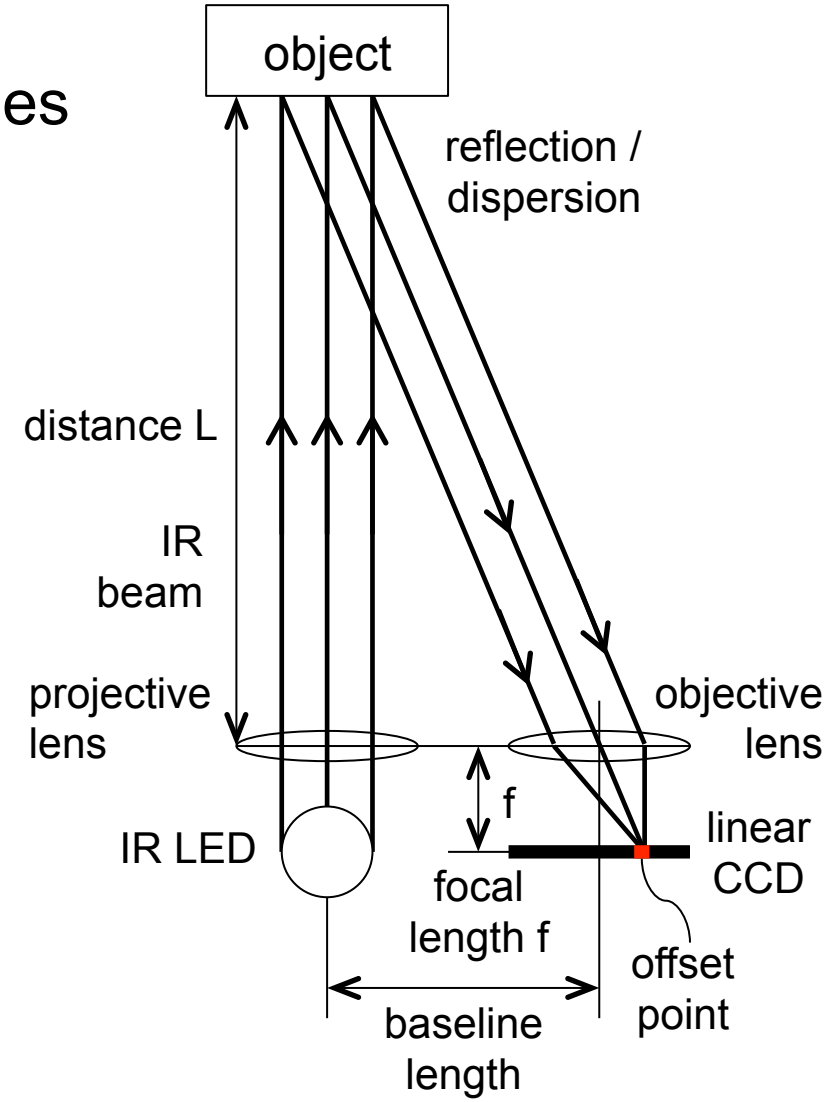


Figure sources: TAOS Datasheet

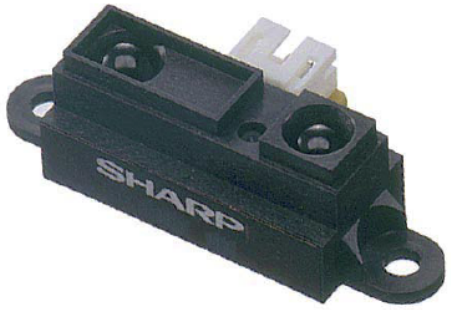
Infrared Distance Sensors

- An emitter sends out light pulses
- A linear CCD array receives reflected light
- Distance corresponds to position on CCD array





Infrared Distance Sensors



- Sharp GP2Y0A21YK0F
  - Distance: 10 to 80 cm
  - Near infrared: λ = 870 nm
  - Operation: 5V, 30mA
  - Connectors: Vcc, GND, Vout
- Needs significant current
  - Put large capacitor between Vcc and GND
- Linearizing analog output
  - distance = 27.93 * $V_{out}^{-1.2}$
  - piecewise linear interpolation

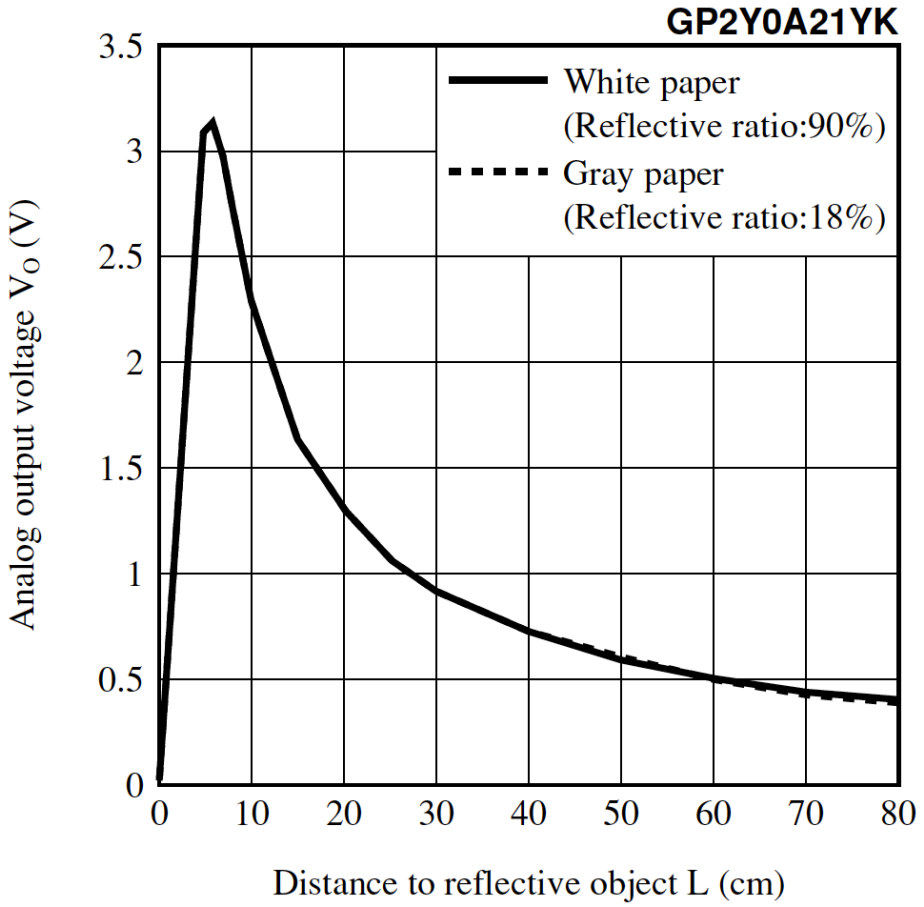
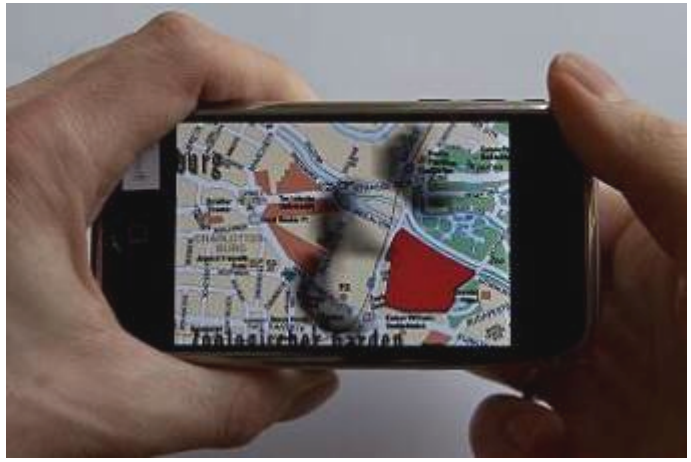
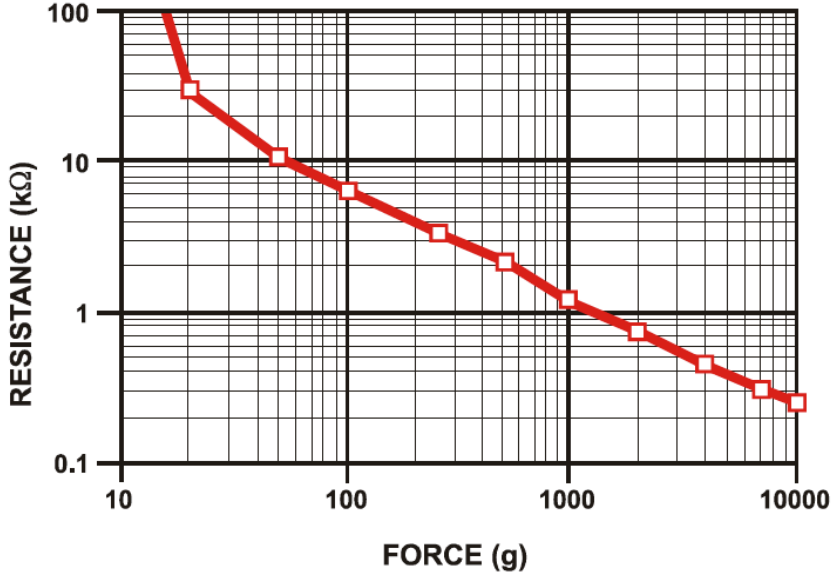


Figure sources: Sharp Datasheet

Force Sensing Resistors (FSRs)

- Force Sensing Resistors
  - Composed of multiple layers
  - Flat, sensitive to bend
  - Force changes resistance
  - Non-linear response curve
- Example: Interlink FSR-402





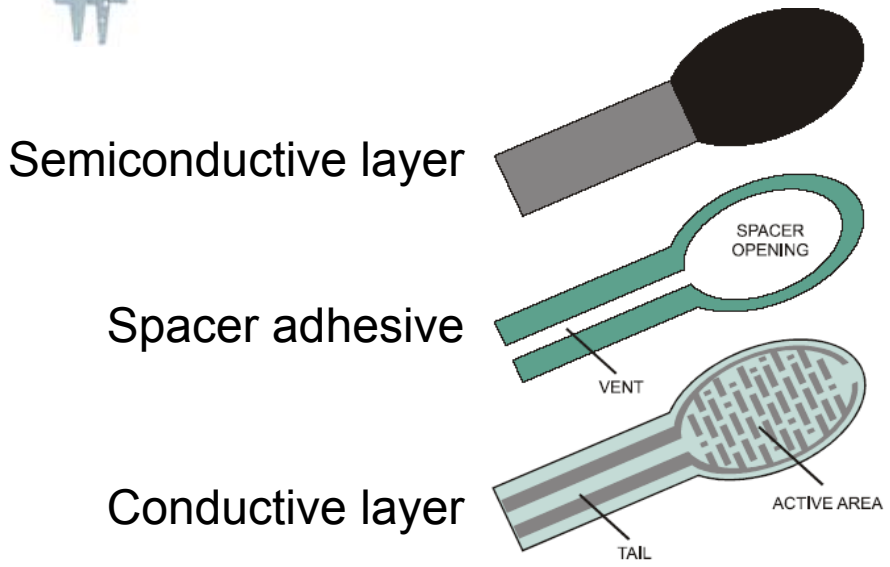
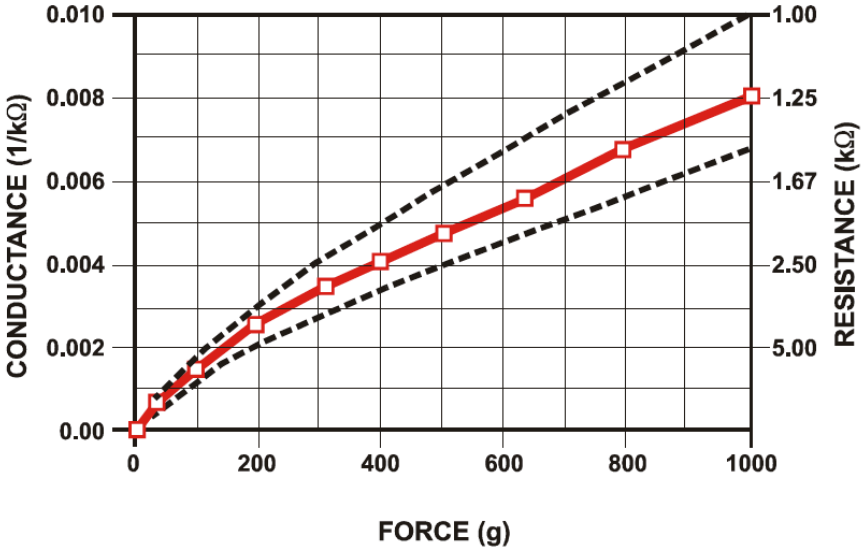


Figure 1: FSR Construction

FSR Characteristics

- Low force range 0..1kg important for human interaction
- Not very precise: force accuracy 5..25%
- But humans are even worse in judging pressure



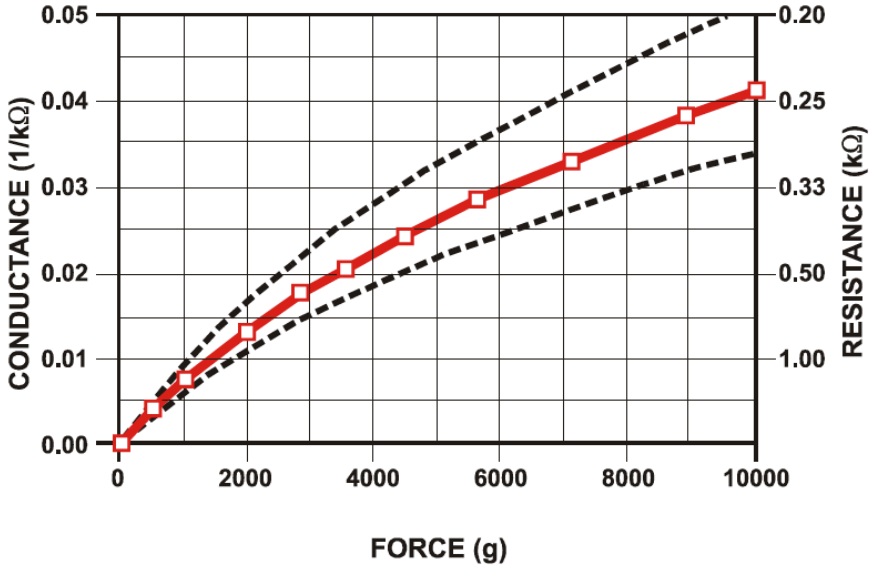
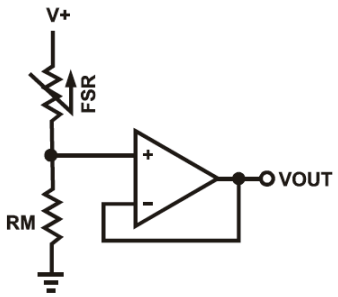
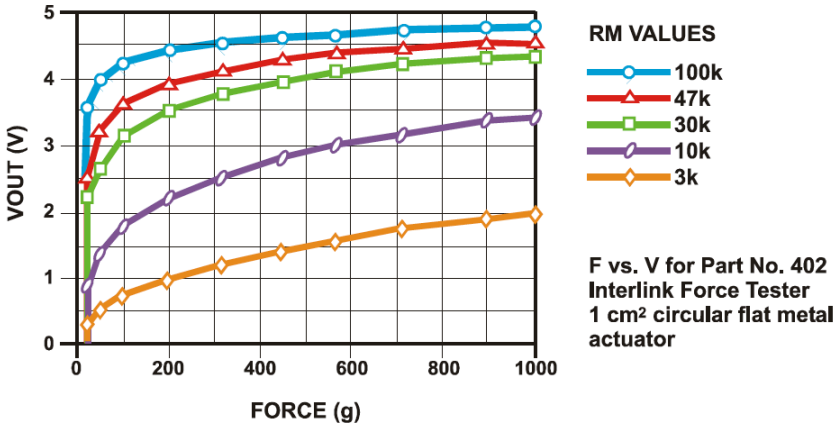


Figure sources: Interlink

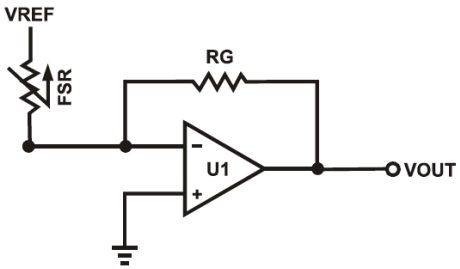
FSR Interface Circuits

- Voltage divider
  - Very nonlinear
  - Behaves like switch





- Current-to-voltage converter
  - Better dynamic range



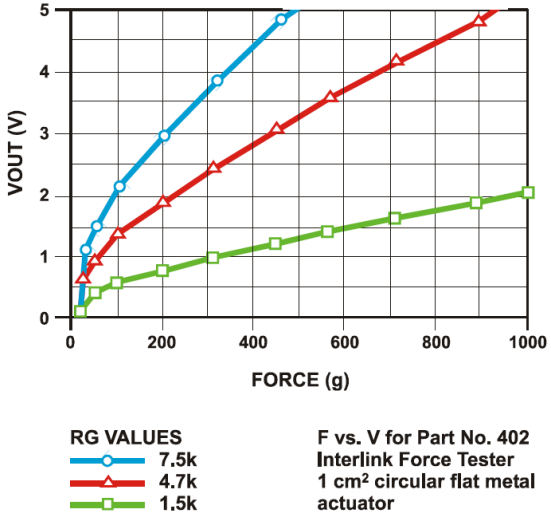


Figure sources: Interlink

FSR Shapes

- Examples from Interlink

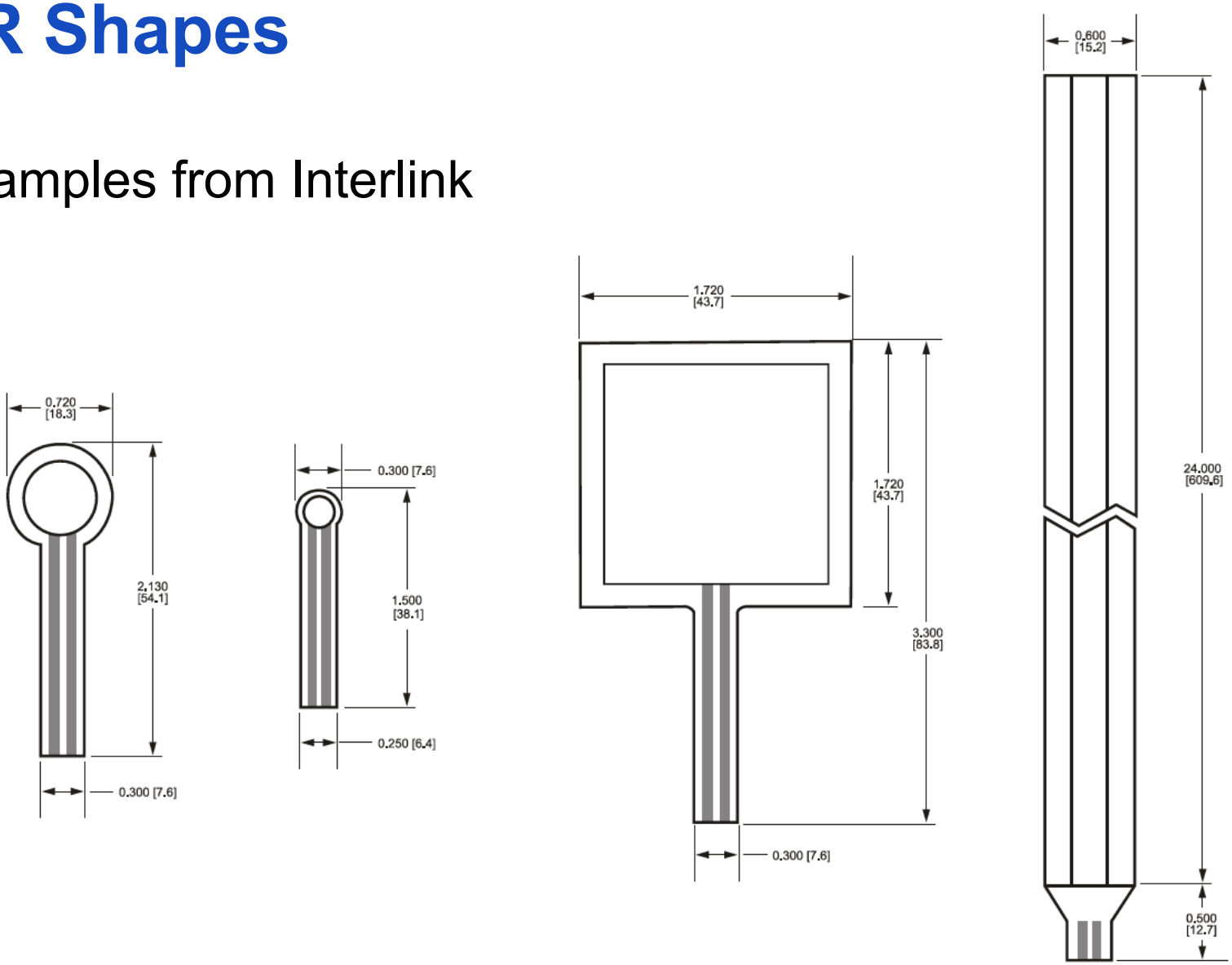
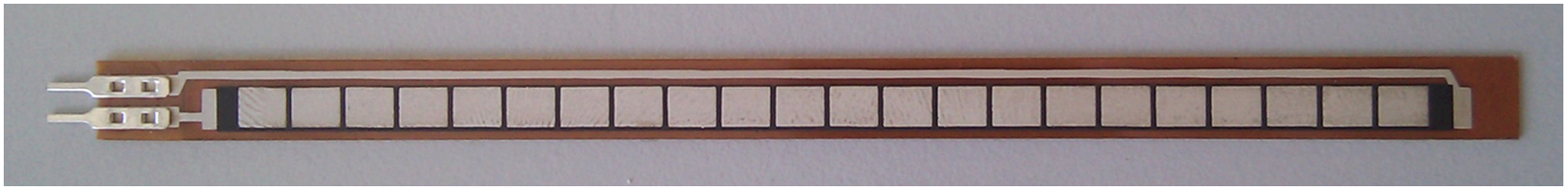


Figure sources: Interlink

FSR Bend Sensors

- Change resistance when bent
- Un-flexed: 10kΩ Flexed / bent 90°: 30..40kΩ

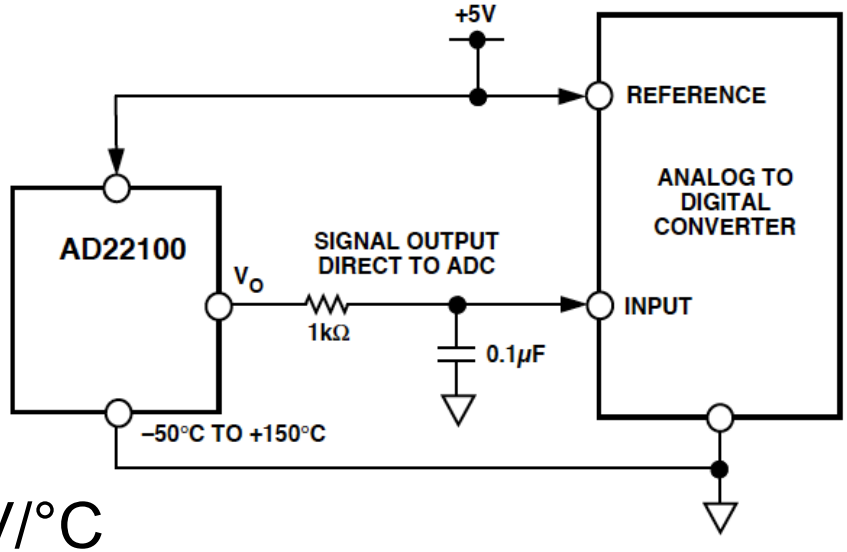


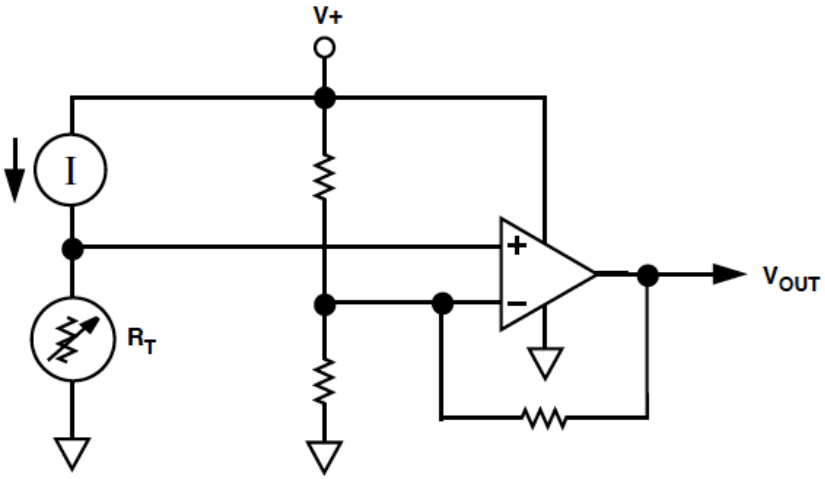
- Sensor glove
  - http://www.tufts.edu/programs/mma/emid/projectreportsS04/moerlein.html

http://www.tufts.edu/programs/mma/emid/projectreportsS04/moerlein.html

Temperature Sensor (Analog Devices AD22100)

- Temperature range: –50°C to +150°C
- Accuracy: < ±2%
- Linearity: < ±1%
- Output: V_{OUT} = 1.375 V + T * 0.0225 V/°C





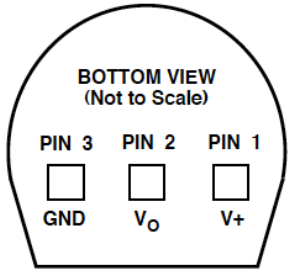
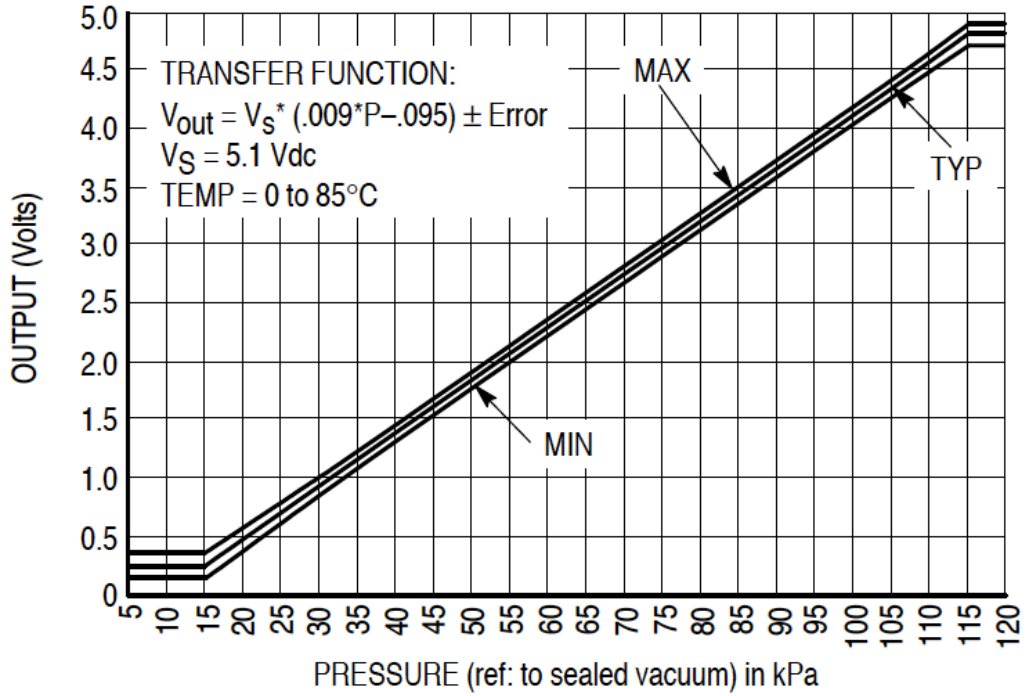


Figure sources: AnalogDevices Datasheet

Air Pressure Sensors (MPX 4115A)

- Senses absolute air pressure in altimeter or barometer applications
- High level analog output signal
- Temperature compensation



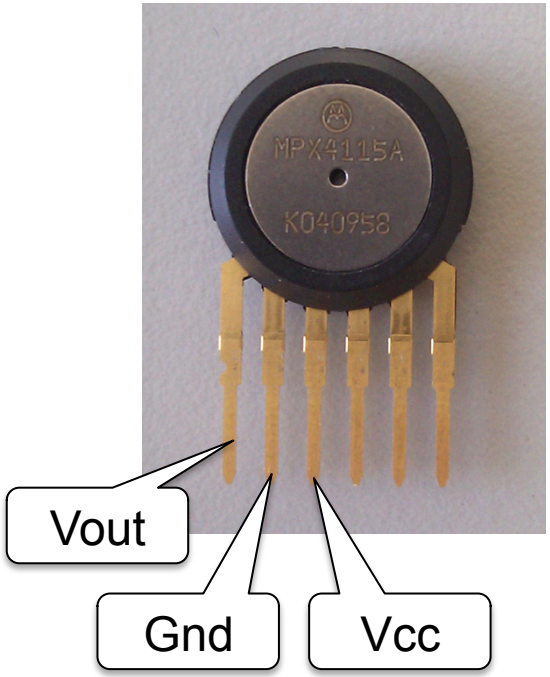
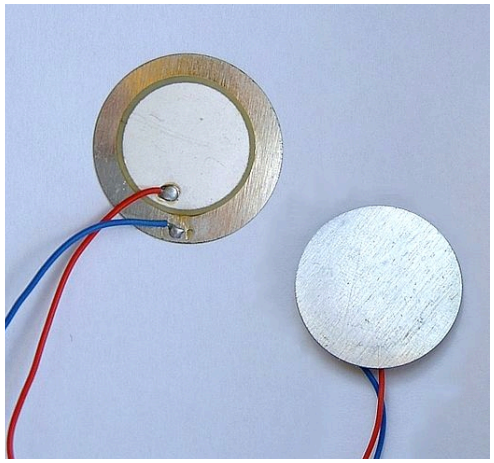


Figure sources: Motorola Datasheet

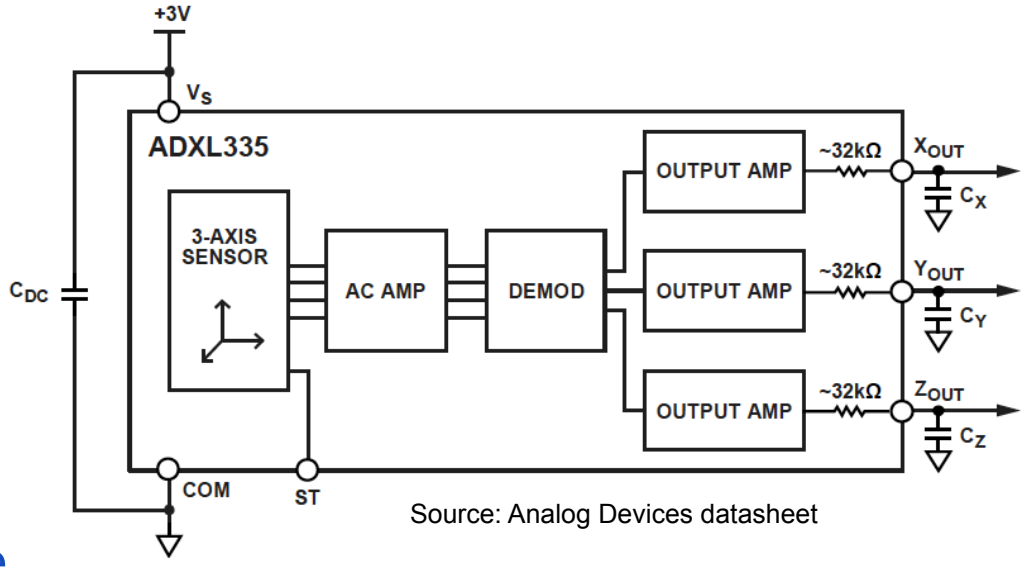
Piezo Elements as Sensors



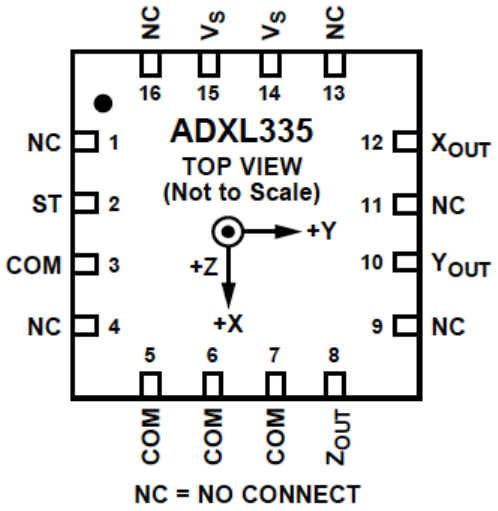
- Piezo elements can be used for output, but also for sensing vibration, e.g. knocks
  - Generates voltage when deformed by vibration, sound wave, mechanical strain
  - Generates vibration (a sound), when voltage is applied
- Directly usable as sensor by reading analog value with AVR's ADC
- Piezos are polarized (red = V_{cc}, blue= ground)
- Need a current-limiting resistor (1MΩ)
- Glue against sensing surface

Accelerometer ADXL335

- Polysilicon surface-micromachined sensor
- 3-axis sensing, ±3g
  - Gravity as static reference
- Low power (350µA @ V_{cc} = 3V)
- Output voltages X_{out}, Y_{out}, Z_{out} proportional to acceleration
- Selectable bandwidth / filtering
  - Capacitors C_X, C_Y, C_Z
  - $F_{-3\,dB} = 1 / (2\pi\,(32k\Omega)\,C_{(X,\,Y,\,Z)})$
  - Max: 1600 Hz (x,y axes), 500 Hz (z axis)



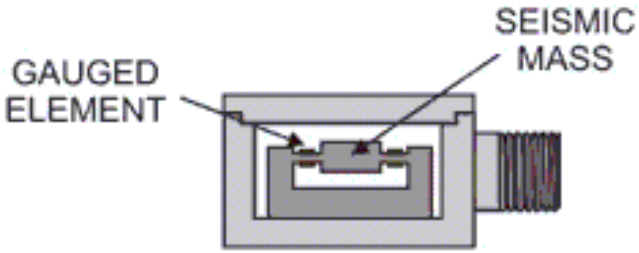
Source: Analog Devices datasheet



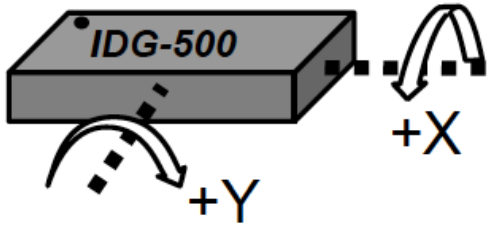
Source: Analog Devices datasheet

How do Accelerometers work?

- Causes of acceleration
  - Gravity, vibration, human movement, etc.
- Operating principle
  - Conceptually: damped mass on a spring
  - Typically: silicon springs anchor a silicon wafer to controller
  - Movement to signal: Capacitance, induction, piezoelectric etc.
- For ADXL335
  - Polysilicon surface-micromachined structure containing mass
  - Polysilicon springs suspend mass
  - Deflection of mass measured with differential capacitor: one plate fixed, other attached to moving mass
  - Square waves drive plates
  - Deflection unbalances differential capacitor

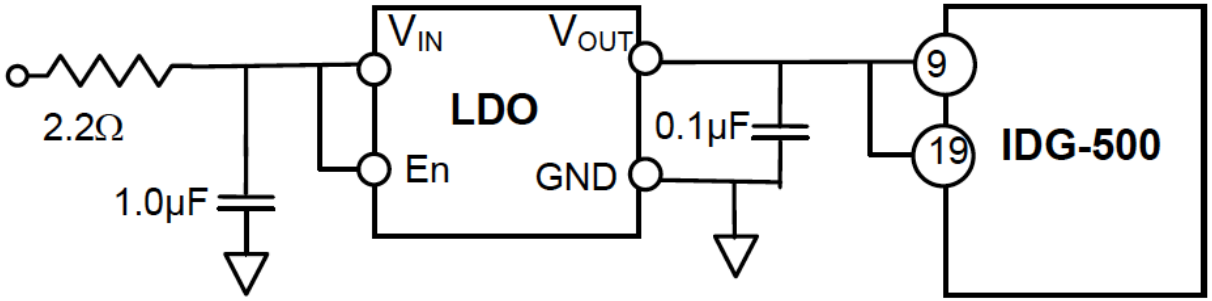


Gyroscope IDG500



Source: InvenSense datasheet

- Dual-axis angular rate sensor (gyroscope)
  - Senses rate of rotation about X- and Y-axis (in-plane sensing)
  - Factory-calibrated
  - Low-pass filters
- V_{CC} = 3V



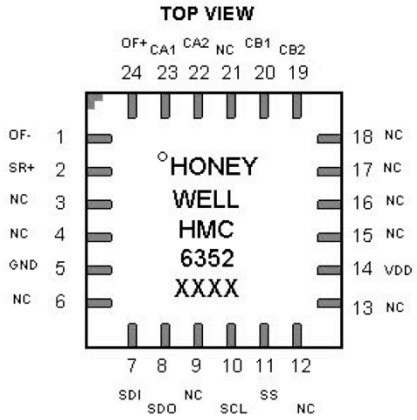
Source: InvenSense datasheet

- Output voltage proportional to the angular rate
- Separate outputs for standard and high sensitivity
  - X-/Y-Out Pins: 500°/s full scale range, 2.0mV/°/s sensitivity
  - X/Y4.5-Out Pins: 110°/s full scale range, 9.1mV/°/s sensitivity

I2C Magnetometer / Compass Honeywell HMC6352

- Compass module
  - 2-axis magneto-resistive sensors
  - Support circuits
  - Algorithms for heading computation
- Parameters
  - V_{CC} = 2.7..5.2V, typ. 3.0V
  - Update rate: 1..20Hz
  - Heading resolution: 0.5°
  - I2C interface

used for complex sensors (e.g., allows sending configuration commands)



Source: Honeywell datasheet

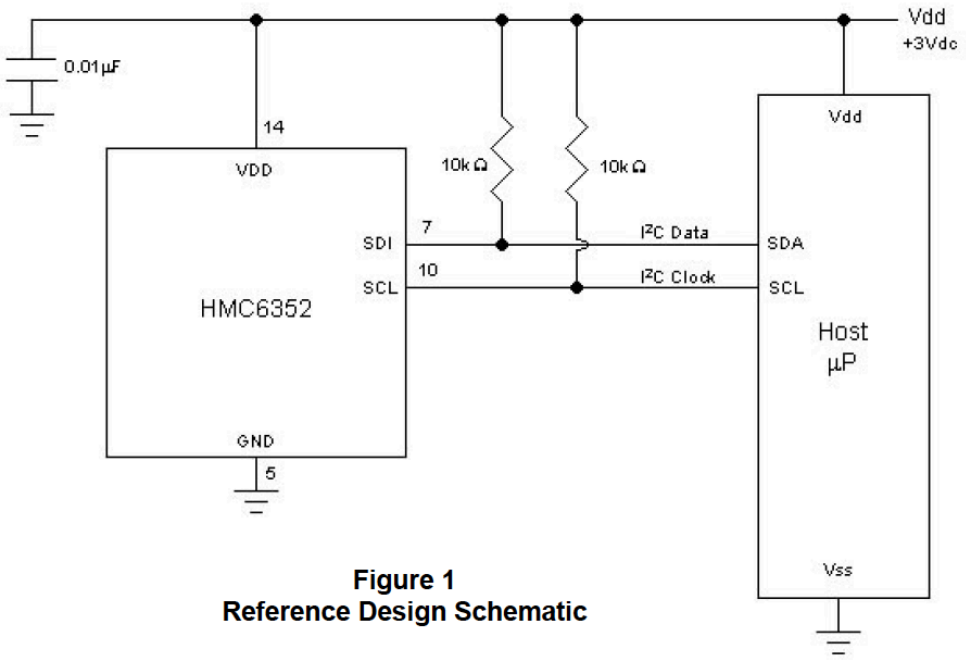


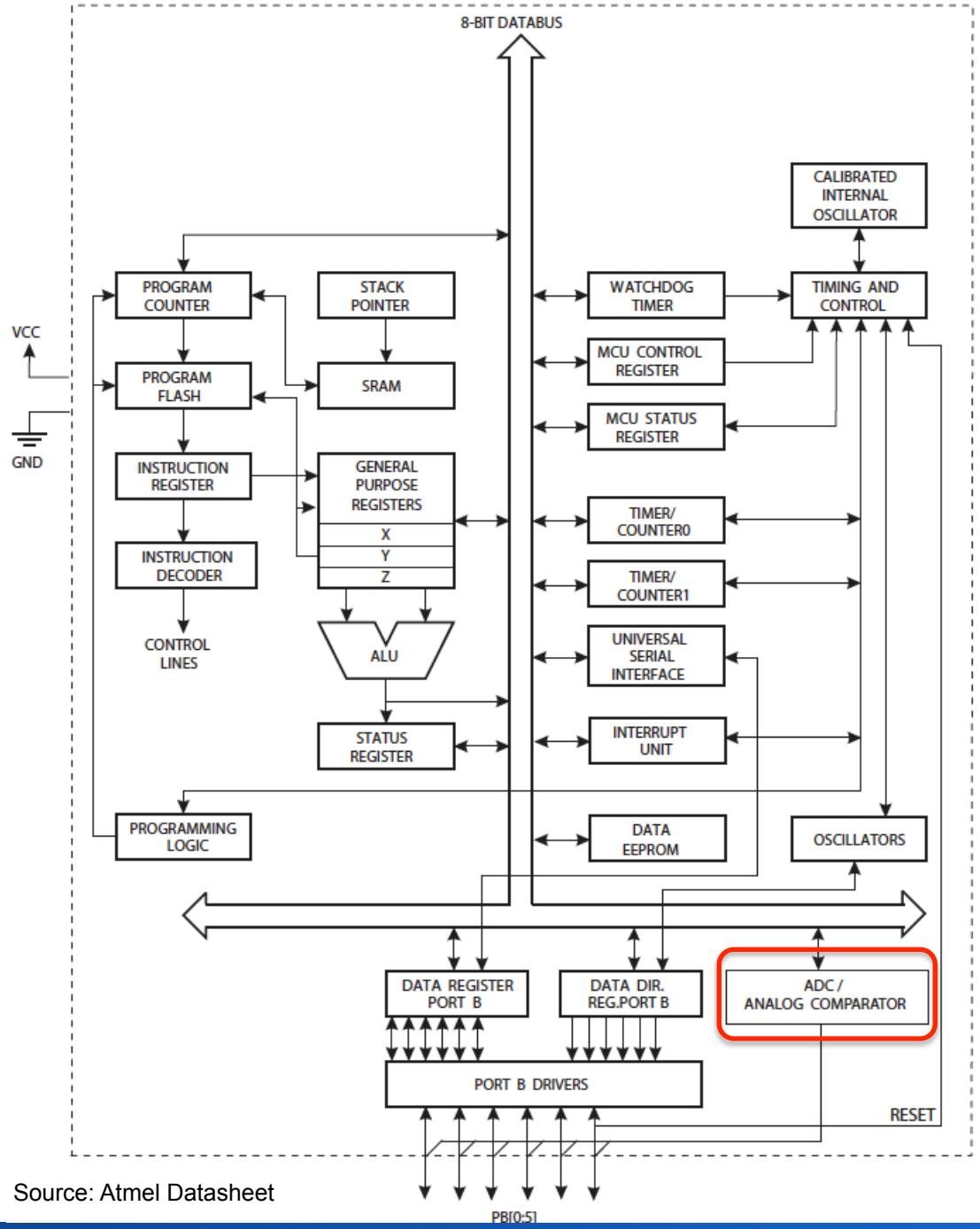
Figure 1
Reference Design Schematic

Source: Honeywell datasheet

ANALOG TO DIGITAL CONVERSION

AVR ATtiny45 Architecture

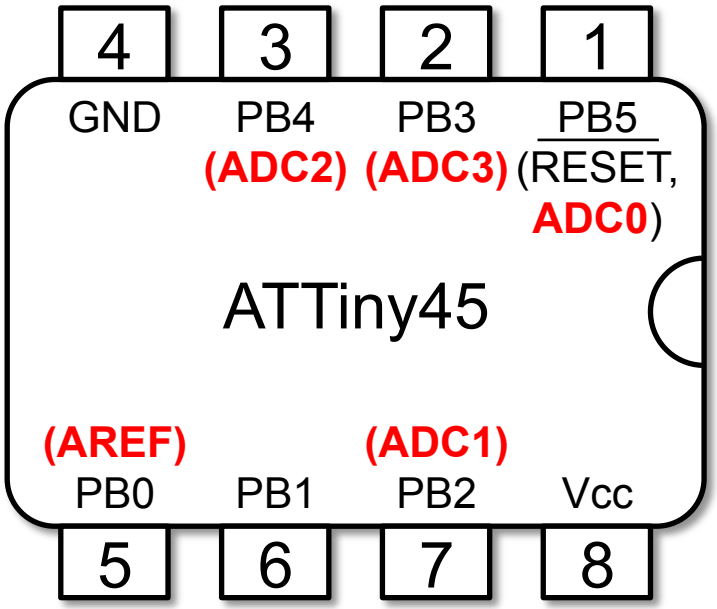
- Analog to Digital Converter



Source: Atmel Datasheet

Analog to Digital Converter

- Converts analog input voltage to 10-bit digital value
  - value = 1024 * V_{IN} / V_{REF}
  - Min analog value = ground
  - Max analog value = V_{CC}
  - Reference voltage: 1.1V or 2.56V or external reference or V_{CC}
  - 0x000 = ground
  - 0x3ff = ref. voltage - one LSB
- Characteristics
  - 4 input channels
  - 65-260µs conversion time
  - 15kSPS conversion rate (SPS = samples per second)



Analog to Digital Converter Registers

- ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
0x07	REFS1	REFS0	ADLAR	REFS2	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	

Source: Atmel Datasheet

- MUX0..1: Selecting input channel
  (00=ADC0, … , 11=ADC3)
- REFS0..1: Internal reference
  (0: V_{CC}, 1: external ref. at PB0, 2: internal ref. 1.1V)
- ADLAR: ADC left adjust result
  (1 = left adjust result)

Analog to Digital Converter Registers

- ADCL and ADCH – The ADC Data Register, ADLAR = 0

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	

Source: Atmel Datasheet

- ADCL and ADCH – The ADC Data Register, ADLAR = 1

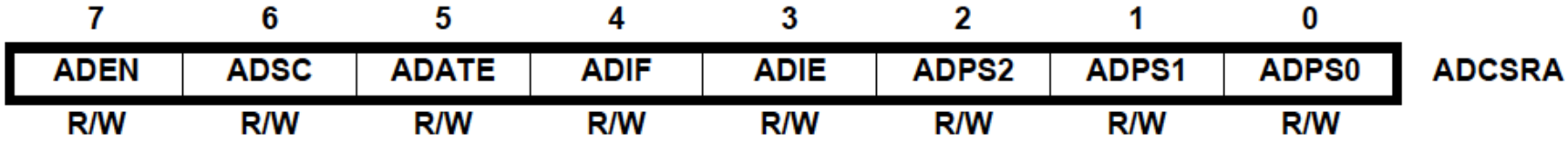
15	14	13	12	11	10	9	8	
ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
ADC1	ADC0	–	–	–	–	–	–	ADCL
7	6	5	4	3	2	1	0	

Source: Atmel Datasheet

- If left adjusted: read ADCH to get 8-bit precision

Analog to Digital Converter Registers

- ADCSRA – ADC Control and Status Register A



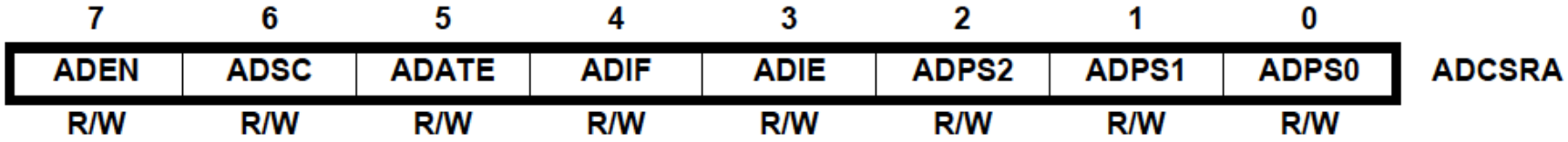
7	6	5	4	3	2	1	0	
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Source: Atmel Datasheet

- ADEN: ADC enable (1 = on, 0 = off)
- ADSC: ADC start conversion, write 1 to start conversion
  - Reads as 1 while conversion in progress
- ADATE: ADC auto trigger enable, write 1 to enable triggering on positive edge of trigger signal
- ADIF: ADC interrupt flag, set when conversion complete
- (continued on next slide…)

Analog to Digital Converter Registers

- 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	

Source: Atmel Datasheet

- …
- ADIE: ADC interrupt enable, write 1 (and set I-bit in SREG) to enable “ADC complete interrupt”
- ADPS2:0: ADC prescaler select bits, division factor of system clock to ADC clock

  0..7: 2, 2, 4, 8, 16, 32, 64, 128

Analog to Digital Converter Registers

- ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
0x03	BIN	ACME	IPR	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R/W	R/W	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

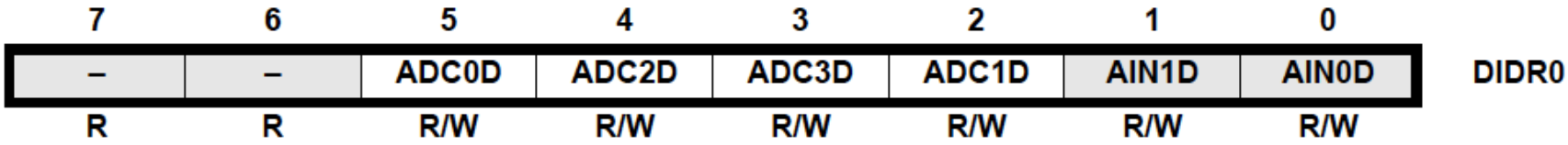
Source: Atmel Datasheet

- ADTS2..0: ADC auto trigger source, triggers conversion on rising edge of selected interrupt

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/Counter Compare Match A
1	0	0	Timer/Counter Overflow
1	0	1	Timer/Counter Compare Match B
1	1	0	Pin Change Interrupt Request

Analog to Digital Converter Registers

- DIDR0 – Digital Input Disable Register 0



Source: Atmel Datasheet

- Write 1 to disable unneeded ADC input channels
- Reduces power consumption

Starting an Analog-to-Digital Conversion

- Starting a single conversion
  - Write 1 to start conversion bit (ADSC) in control register (ADCSRA)
  - ADSC reads 1 during conversion
- Auto triggering
  - Various sources (e.g. timer for fixed intervals)
  - Interrupt flag for source will be set and conversion triggered, even if interrupt is disabled, flag must be cleared to trigger subsequent conversion
  - Free-running-mode: start new conversion as soon as ongoing conversion has finished

Conversion Timing

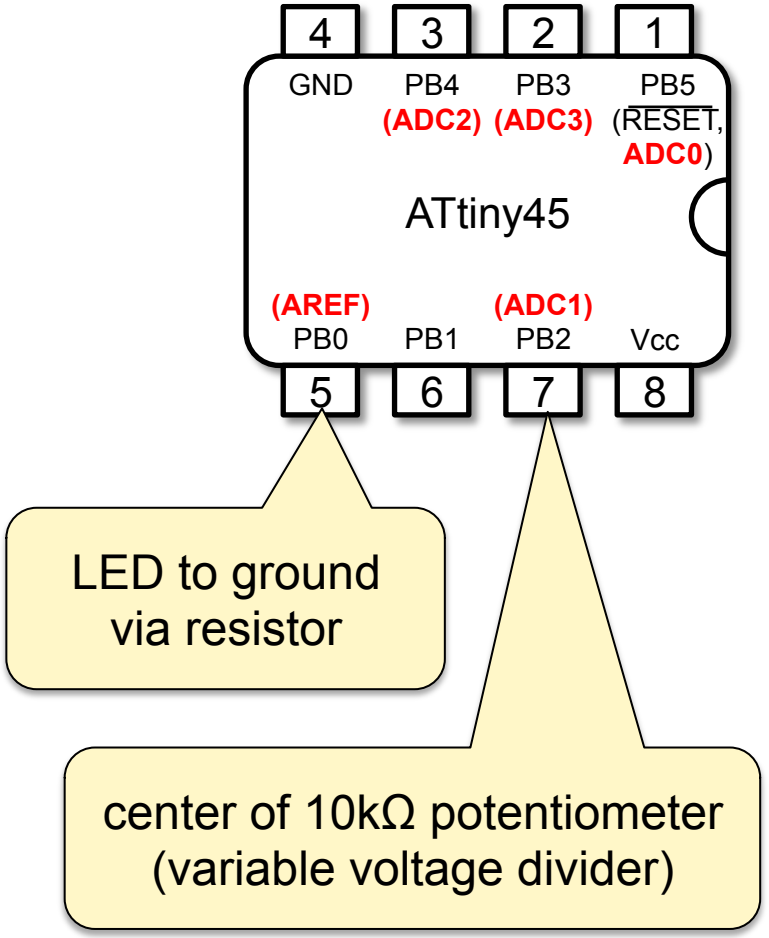
- Input clock frequency
  - 50..200 kHz ADC for maximum resolution
  - >200kHz if lower resolution sufficient
- Duration
  - Normal conversion takes 13 ADC clock cycles
  - Initial conversion 25 ADC clock cycles
- Discard first result after ADC activation or chg. ref.
  - Result may be wrong → discard

Analog Input Signal

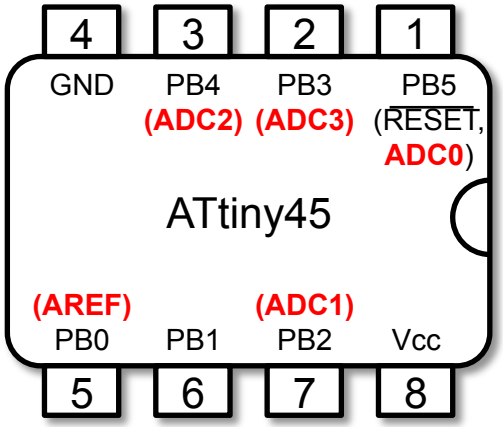
- Signal should be slowly varying
- Signal components higher than the Nyquist frequency (f_{ADC} / 2) should not be present
- Recommendations
  - Keep signal paths short
  - Separate analog tracks from digital tracks
  - Do not switch digital output pin during conversion
  - Place bypass capacitors close to V_{CC} and GND pins
  - Use ADC Noise Reduction Mode for higher accuracy

Example: 10kΩ Poti on ADC1 (PB2)

```
#include <avr/io.h>
#include <util/delay.h>
static unsigned char ledState = 0;

void blinkLed(unsigned int d) {
        ledState = 1 - ledState;
        if (ledState) PORTB |= 0x01;
        else PORTB &= 0xfe;
        _delay_ms(d);
}

int main() {
        // next slide...
}
```



Example: 10kΩ Poti on ADC1 (PB2)

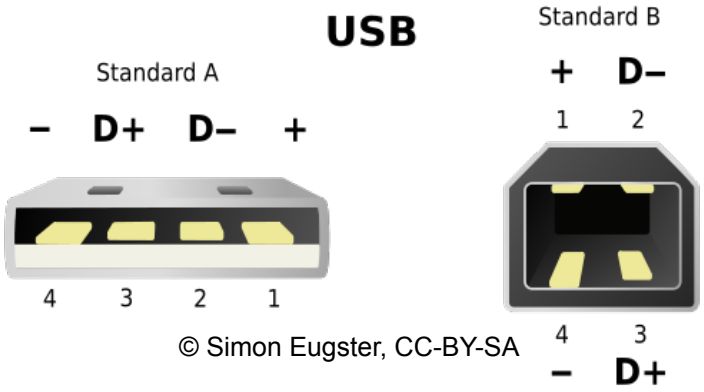



```
int main() {
    DDRB  &= 0b11111011; // port 2 as input
    PORTB &= 0b11111011; // port 2 disable pull-up

    ADMUX = (0 << REFS0) | (1 << ADLAR) | 1; // Vcc, left adj., ADC1 = PB2
    ADCSRA = (1 << ADEN) | 6; // enable, prescaler 1:64 (150kHz)
    ADCSRB = 0; // free running mode
    DIDR0 = (1 << ADC0D) | (1 << ADC2D) | (1 << ADC3D);
                // disable ADCs 0, 2, 3
    while (1) {
        ADCSRA |= (1 << ADSC); // start conversion
        while (ADCSRA & (1 << ADSC)); // wait until conversion complete
        unsigned int d = ADCH;
        blinkLed(d); // use value…
    }
```

UNIVERSAL SERIAL BUS (USB)

Universal Serial Bus (USB)

- High data rates (difficult to process with ATtinys)
  - USB 1.0: 1.5 Mbit/s (Low-Speed) and 12 Mbit/s (Full-Speed)
  - USB 2.0: 480 Mbit/s (High-Speed)
  - USB 3.0: 5 Gbit/s (Super-Speed)



Pin 1	V_{CC} (+5 V)
Pin 2	Data-
Pin 3	Data+
Pin 4	Ground

- 500mA max. (at V_{CC} = 5V)

- Some AVRs have built-in USB modules
  - the ones we use don't

USB-to-Serial Converter

- Serial side (RS232)
  - RTS (request-to-send, green)
  - RX (receive, yellow)
  - TX (transmit, orange)
  - 5V (red)
  - CTS (clear-to-send, brown)
  - GND (black)

- Virtual COM Port Drivers
  - www.ftdichip.com/Drivers/VCP.htm

FTDI FT232RQ Virtual COM Port Drivers

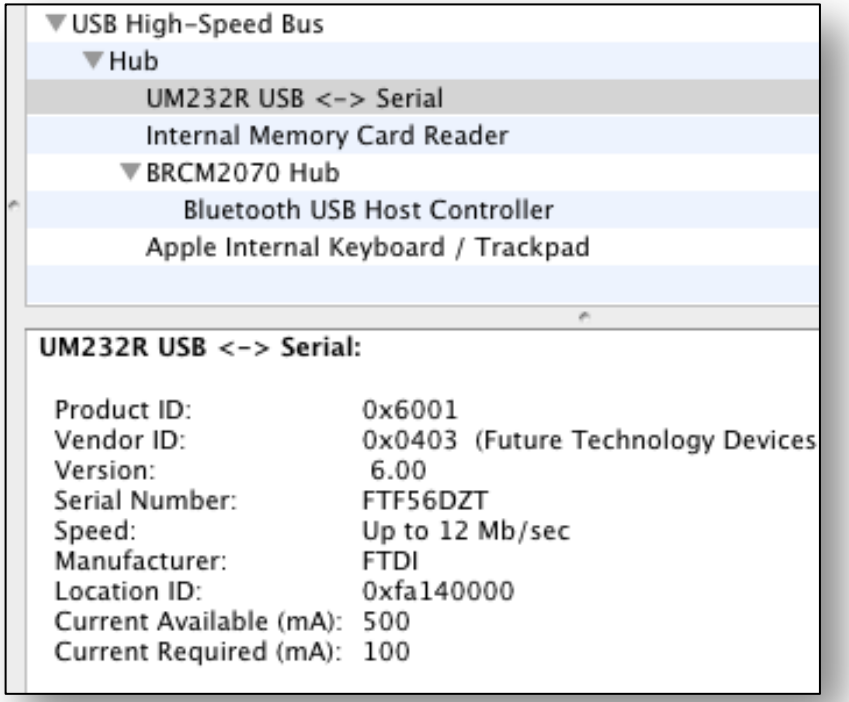
- Virtual COM port drivers
  - http://www.ftdichip.com/Drivers/VCP.htm
- USB device appears as virtual COM port

  cd /dev

  ls -l | grep usb

  cu.usbserial-A100OXPZ

  tty.usbserial-A100OXPZ
- Shows up in System Profiler

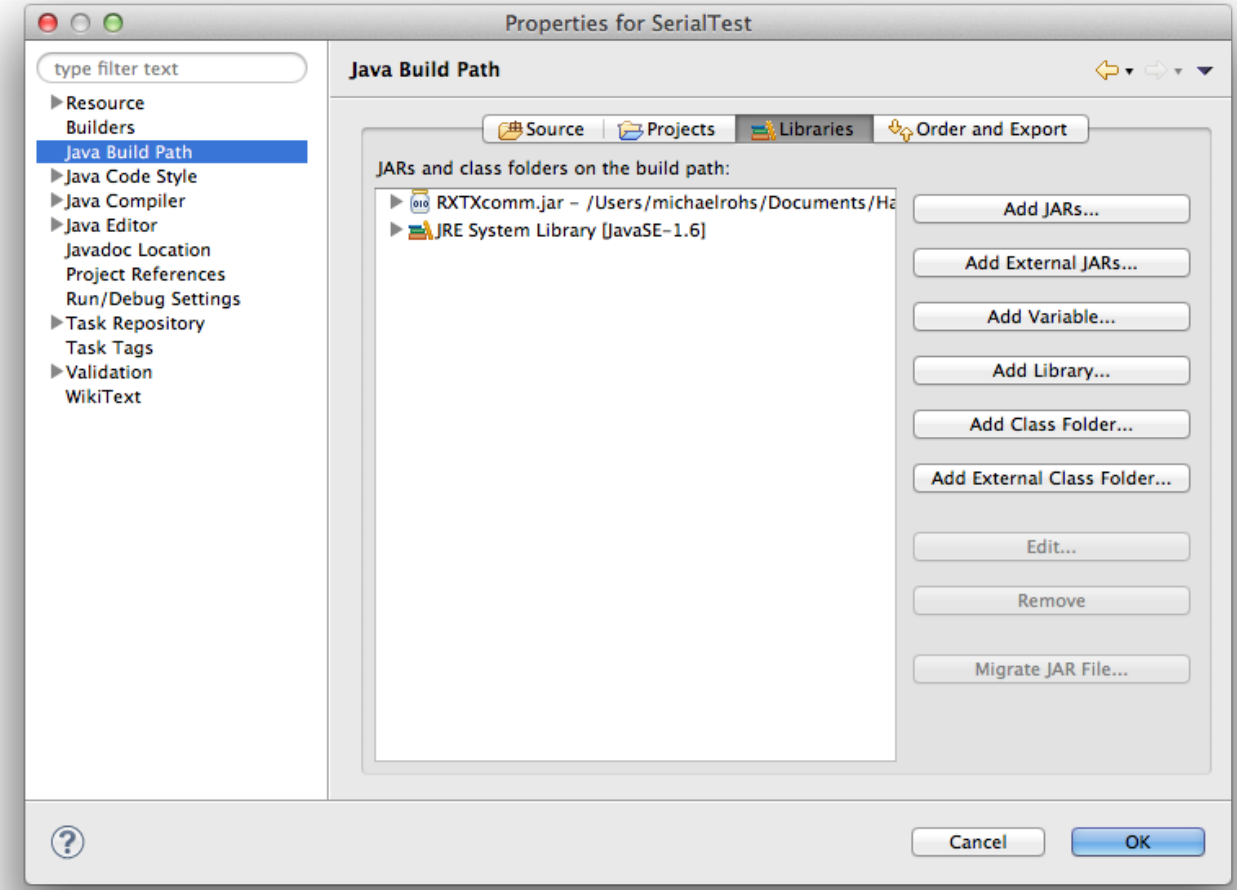


Using Virtual COM Port in Java

- Requires Java Serial Communications API
  - http://download.oracle.com/docs/cd/E17802_01/products/products/javacomm/reference/api/index.html
  - various implementations
- Windows
  - http://www.oracle.com/technetwork/java/index-jsp-141752.html
  - javacomm20-win32.zip
- Mac OS X, Linux
  - http://rxtx.qbang.org/wiki/index.php/Download
  - http://rxtx.qbang.org/wiki/index.php/Using_RXTX
- Add jar to Eclipse project
  - Properties, Java Build Path, Add External JARs…
  - RXTXcomm.jar (or other implementation)

Using Virtual COM Port in Java, Eclipse

- Properties, Java Build Path, Add External JARs…



Example: Sending to Virtual Com Port in Java

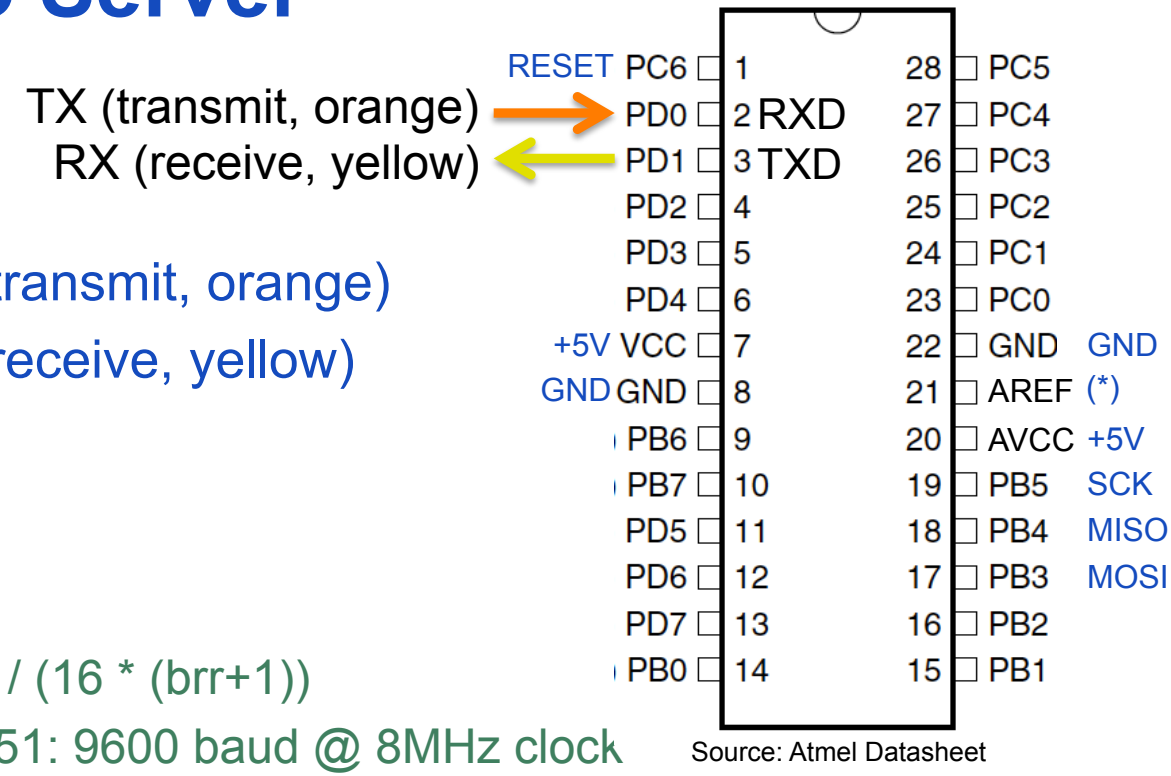
```
import java.io.*;        import java.util.*;        import gnu.io.*;

public class Usb2Serial {
public static void main(String[] args) throws Exception {
   Enumeration ports = CommPortIdentifier.getPortIdentifiers();
   while (ports.hasMoreElements()) {
      CommPortIdentifier portId = (CommPortIdentifier) ports.nextElement();
      if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
         if (portId.getName().equals("/dev/tty.usbserial-A100OXPZ")) {
            SerialPort sp = (SerialPort) portId.open("Usb2Serial", 2000);
            OutputStream os = sp.getOutputStream();
            sp.setSerialPortParams(9600,
                                   SerialPort.DATABITS_8,
                                   SerialPort.STOPBITS_1,
                                   SerialPort.PARITY_NONE);
            os.write("hello world".getBytes());
            try { Thread.sleep(2000); } catch (InterruptedException ex) {}
            sp.close();
         }
} } }
```

replace by actual device name

ATmega8 Echo Server

- PINs
  - PD0 (RXD) to TX (transmit, orange)
  - PD1 (TXD) to RX (receive, yellow)
- Source

```
#include "usart.h"
int main() {
        // baud rate = clock / (16 * (brr+1))
        USART_Init(51); // 51: 9600 baud @ 8MHz clock
        while (1) {
                unsigned char x = USART_Receive();
                USART_Transmit(x);
        }
        return 0;
}
```

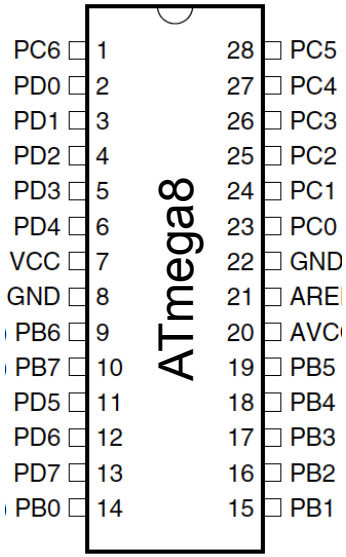


Source: Atmel Datasheet

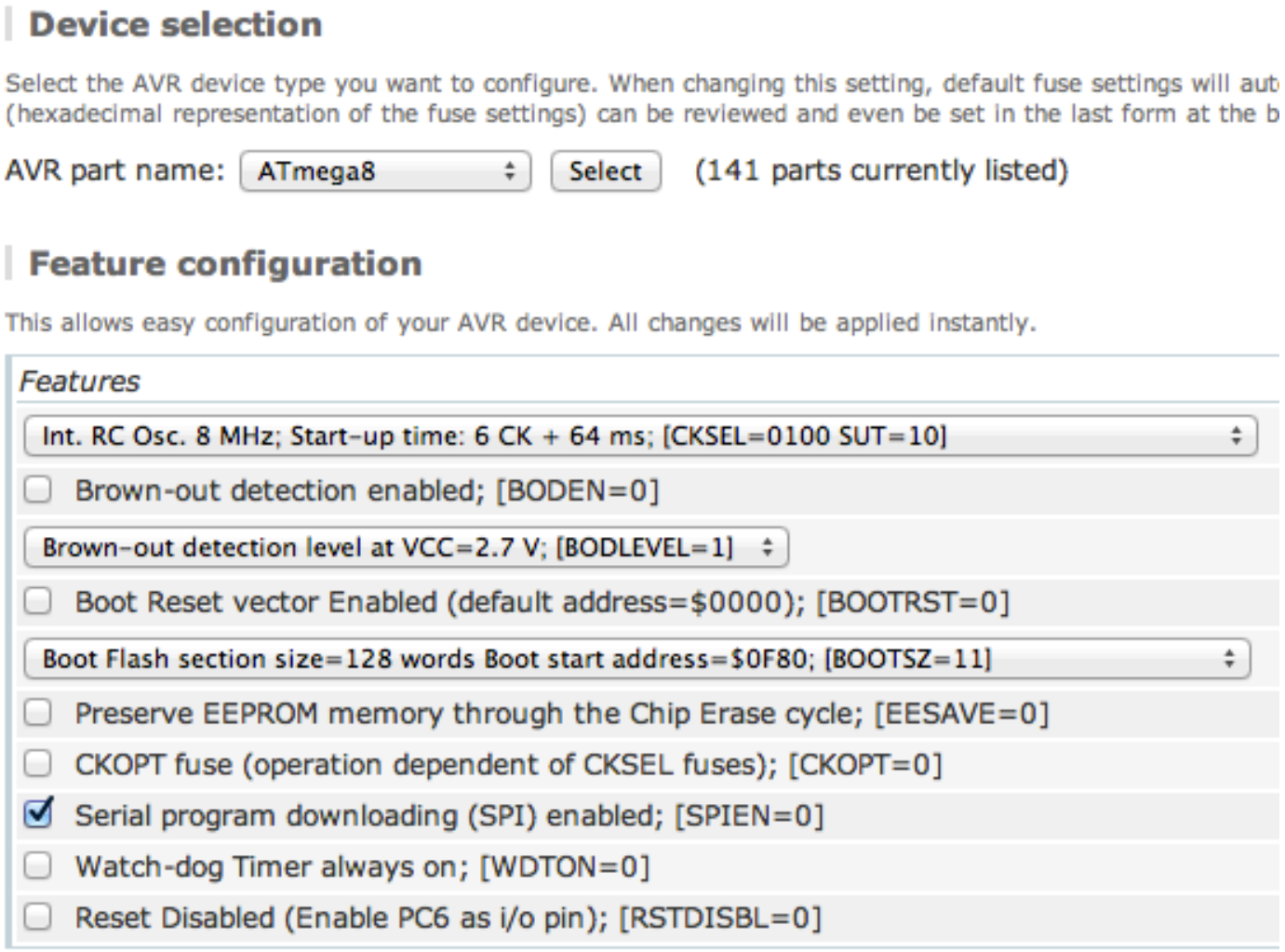
(*) a small capacitor (100nF) between AREF and GND improves noise immunity of the A/D converter

ATmega8 Fuses for EchoServer

- low: 0xE4
- high: 0xDF



Source: Atmel Datasheet



Source: www.engbedded.com/fusecalc/

ATmega8 USART Initialization

```
void USART_Init(unsigned int brr)
{
   // set baud rate: baud = clock / (16 * (brr+1)) <=> brr = clock / (16 * baud) - 1
   UBRRH = (brr >> 8) & 0x0f;
   UBRRL = brr;

   // enable receiver and transmitter
   UCSRB = (1 << RXEN) | (1 << TXEN) | (0 << UCSZ2);

   // set frame format: 8 data bits, no parity, 1 stop bit
   UCSRC = (1 << URSEL) | (0 << UMSEL) | (0 << UPM0) |
           (0 << USBS) | (3 << UCSZ0) | (0 << UCPOL);
}
```

ATmega8 USART Transmission

```
void USART_Transmit(unsigned char data) {
    // wait until data register empty
    while (!(UCSRA & (1<<UDRE)));
    // send the data
    UDR = data;
}

unsigned char USART_Receive() {
    // wait for data to be received
    while (!(UCSRA & (1<<RXC)));
    // return the data
    return UDR;
}
```

ATmega8 USART Transmission

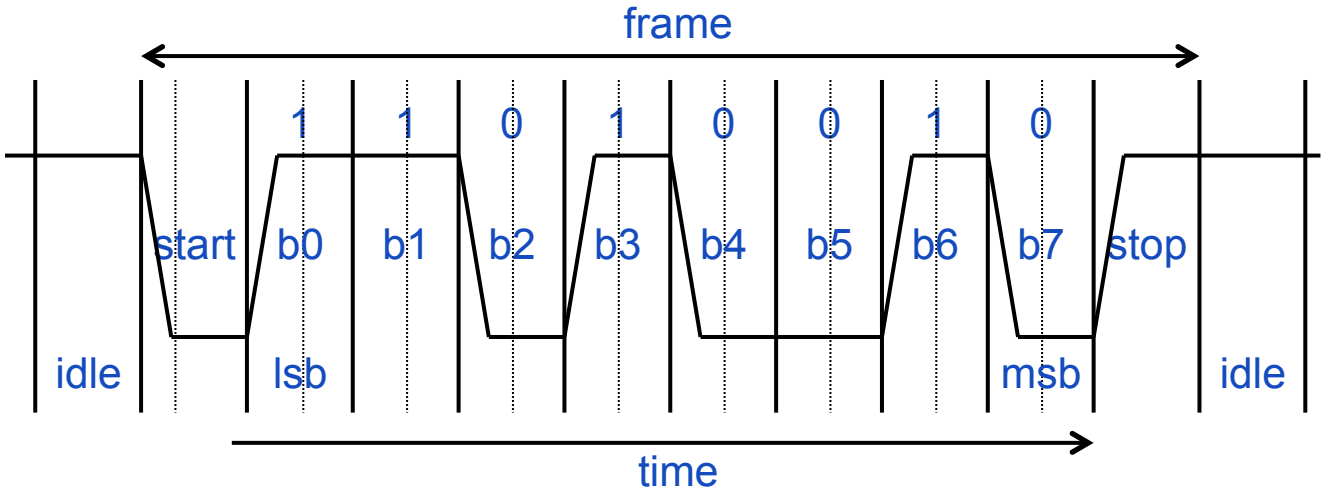
```
void USART_Transmit_String(unsigned char* data)
{
    unsigned char c;
    while ((c = *data++) != 0) {
        USART_Transmit(c);
    }
}
```

Universal Asynchronous Receiver/ Transmitter (UART)

- Sequential transmission/reception of a sequence of bits
- Framing (start bit = 0, stop bit = 1; 8 data bits, no parity bit)

Start (0)	Bit 0 (lsb)	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7 (msb)	Stop (1)

- Timing diagram

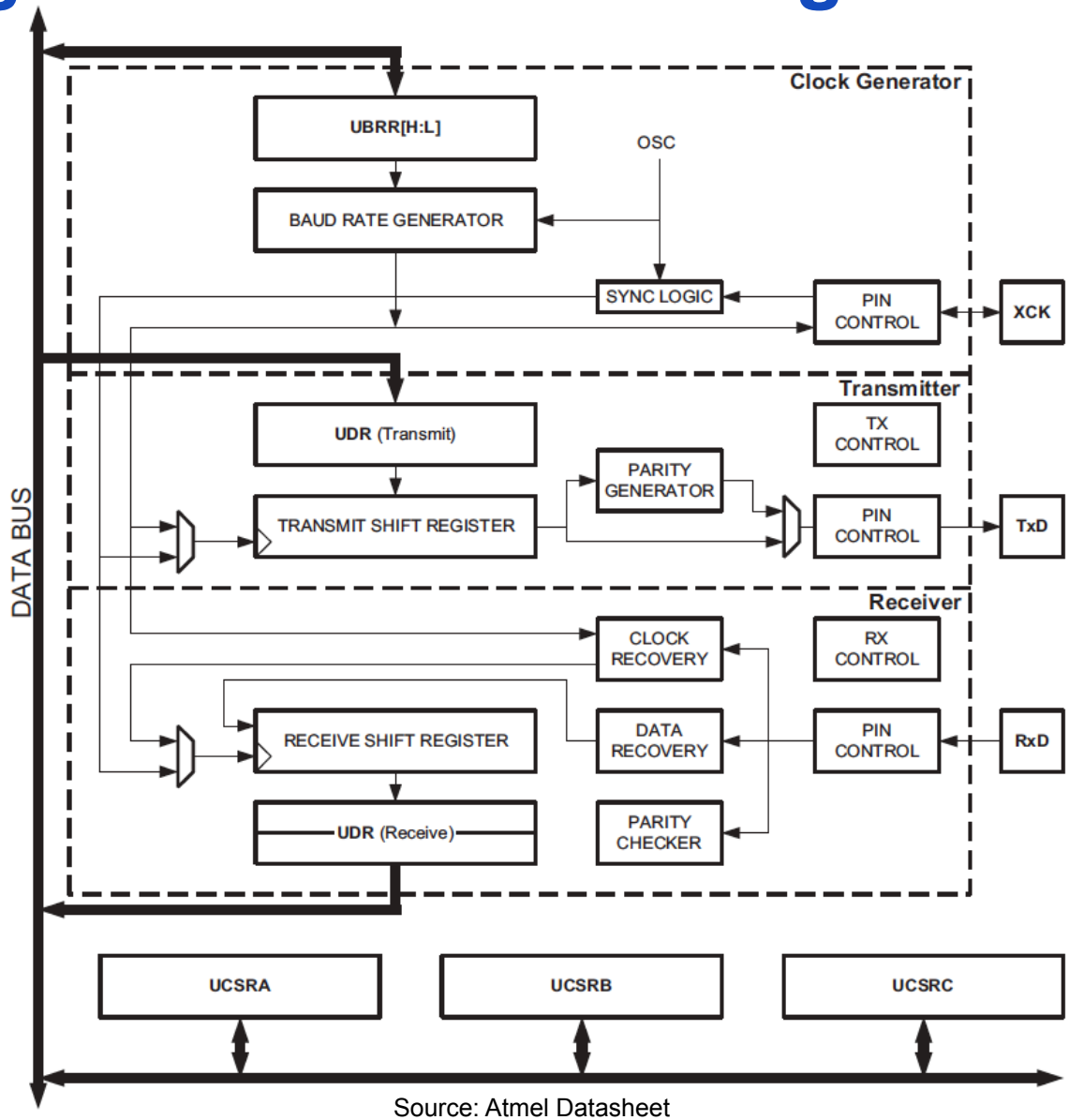


- Hardware handshake: Request-to-send, clear-to-send

ATmega8 USART

- USART = Universal Synchronous and Asynchronous serial Receiver and Transmitter
  - asynchronous: no clock signal, common baud rates, each frame synchronized
  - synchronous: clock signal
  - full duplex: simultaneous transmission and reception
  - frames formats: 5..9 data bits, 1..2 stop bits
  - parity generation / check (even: xor of data bits, odd: *not* even)
  - data overrun and framing error detection
  - interrupts: TX complete, TX data register empty, RX Complete
  - multi-processor communication

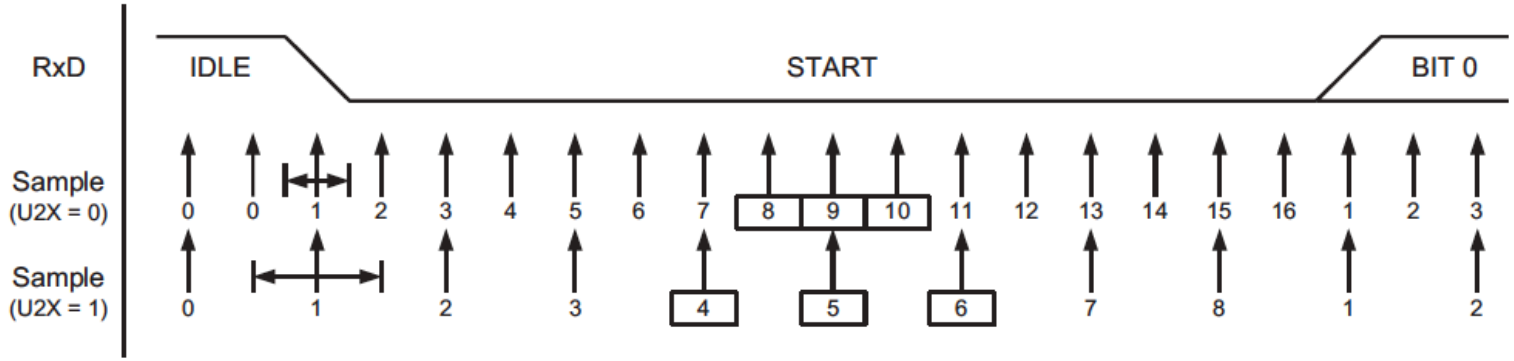
ATmega8 USART Block Diagram



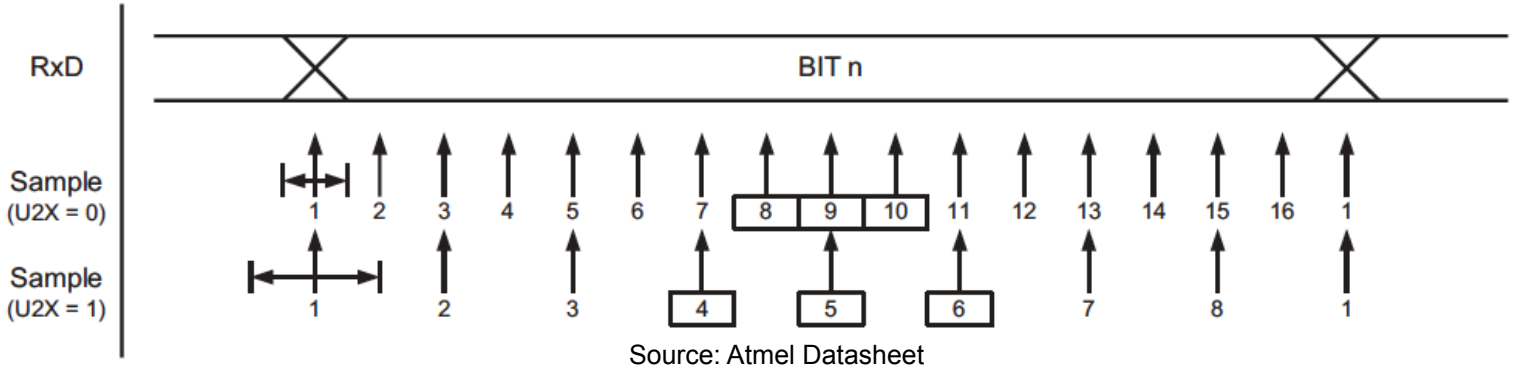
Source: Atmel Datasheet

ATtiny2313 USART Bit Sampling

- Single (U2X = 0) and double speed (U2X = 1) modes
- Double speed: majority vote on three samples
- Start bit:



- Data bits:



Source: Atmel Datasheet

ATmega8 USART Baud Rates

- UBRR = baud rate register (12 bits)
- Baud rate computation
  - asynchronous, normal: $baud = \frac{f_{osc}}{16\,(UBRR+1)} \Leftrightarrow UBRR = \frac{f_{osc}}{16\,baud} - 1$
  - asynchronous, double: $baud = \frac{f_{osc}}{8\,(UBRR+1)} \Leftrightarrow UBRR = \frac{f_{osc}}{8\,baud} - 1$
  - synchronous master: $baud = \frac{f_{osc}}{2\,(UBRR+1)} \Leftrightarrow UBRR = \frac{f_{osc}}{2\,baud} - 1$
- Maximum baud rate error should be ±2.0% (8 data bits)
- Example: f_{osc} = 16MHz, *baud* = 115200:

$$UBRR = \frac{16 \cdot 10^6}{16 \cdot 115200} - 1 \approx 7.68,\ baud = \frac{16 \cdot 10^6}{16 \cdot (8+1)} \approx 111111.11$$ ➔ -3.5% error

BRAINSTORMING

Brainwriting (in your group)

- Repeat 5 rounds
  - 3 minutes: On paper, fill one row with 3 ideas
  - Pass on paper clockwise
  - Read other ideas, fill next line with 3 more ideas
- Select the 3 best ideas
  - 10 minutes
  - Present selected ideas
