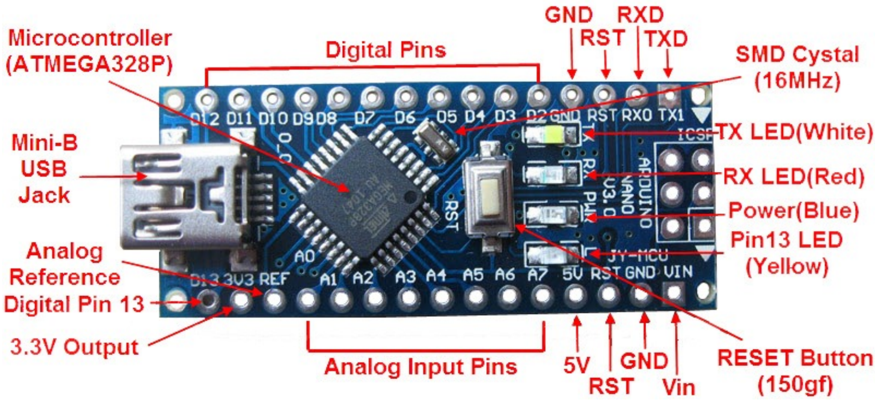
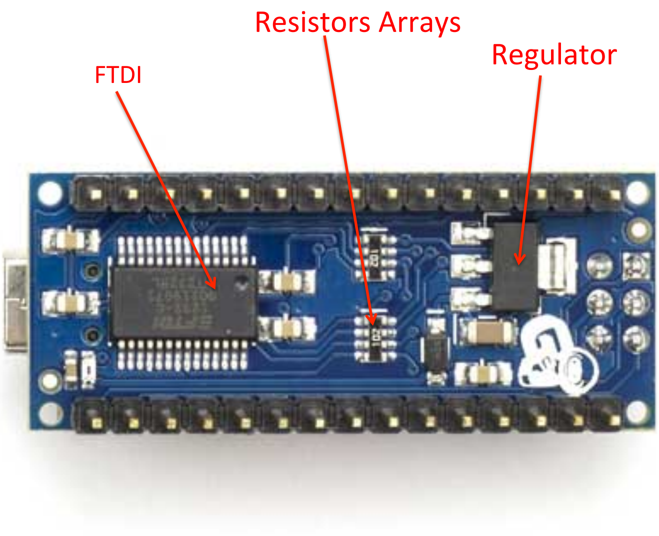
ETH Course 402-0248-00L: Electronics for Physicists II (Digital)

- **1: Setup uC tools, introduction**
- **2: Solder SMD Arduino Nano board**
- **3: Build application around ATmega328P**
- **4: Design your own PCB schematic**
- **5: Place and route your PCB**
- **6: Start logic design with FPGAs**

The ATmega168P / 328P

- AT = Atmel: Big microcontroller company
- mega: microcontroller family
- 16: 16KB Flash memory / 32: 32KB Flash
- 8: 8-bit architecture
- P: PicoPower Technology. Optional. For low power battery-based applications.

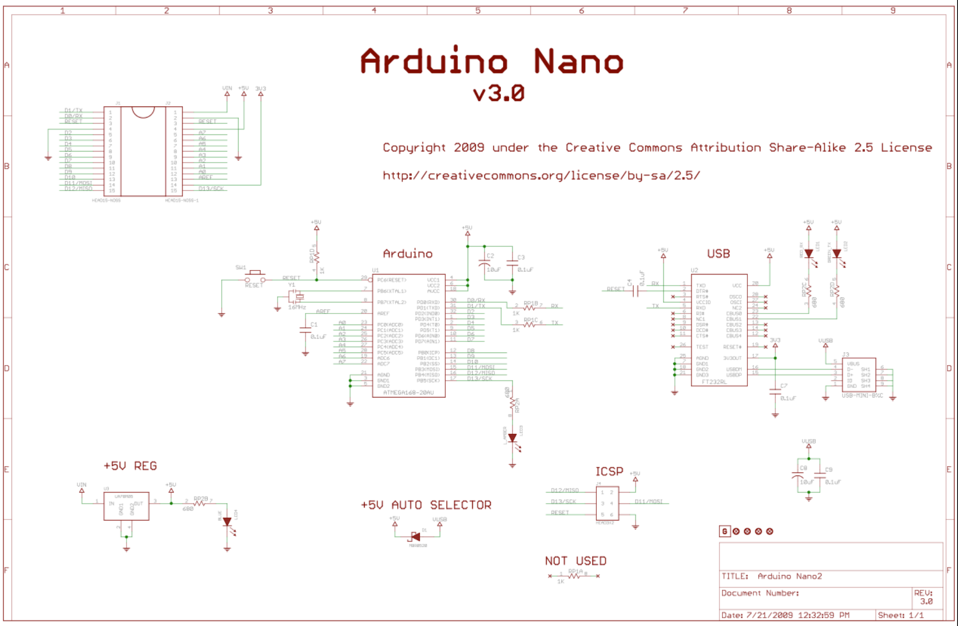

Microcontroller
(ATMEGA328P)
Digital Pins
GND
RST
RXD
TXD
SMD Cystal
(16MHz)
Mini-B
USB
Jack
TX LED(White)
RX LED(Red)
Power(Blue)
Pin13 LED
(Yellow)
Analog
Reference
Digital Pin 13
3.3V Output
Analog Input Pins
5V
RST
GND
Vin
RESET Button
(150gf)


Resistors Arrays
FTDI
Regulator

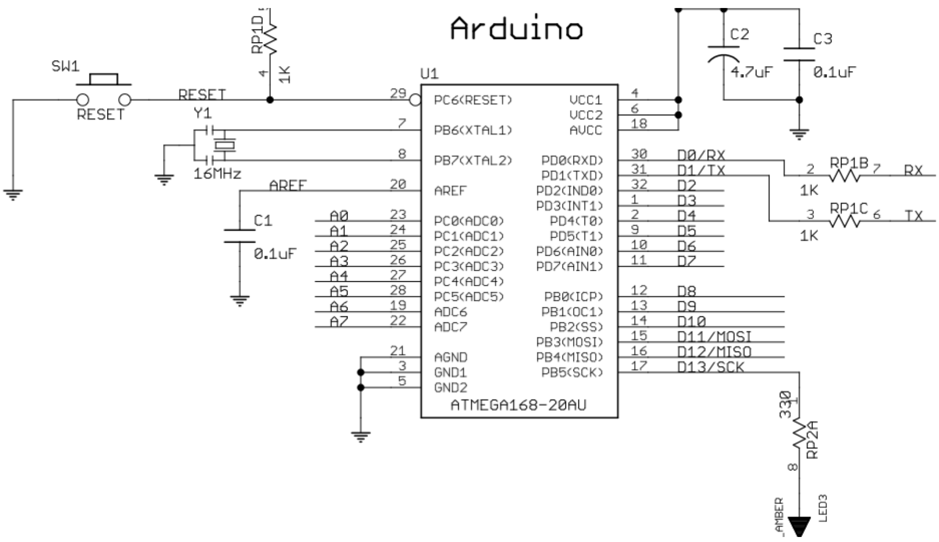
ATmega16/328P capabilities (Ex. 3)

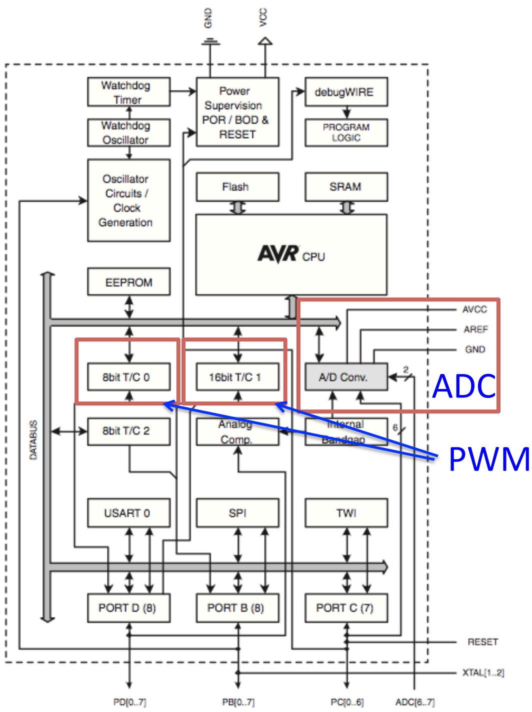
- **System Functions**
  - Power and Clock Manager
  - Low Freq Internal Oscillator
  - Watchdog Timer
  - Real-Time Clock Timer
- **Interrupt Controller**
  - Fixed priority. One level of interruption. Interruptions with flag (can remember) or without. Global Interrupt Enable (I-bit) is disabled during an interrupt service.
- **NO Universal Serial Bus (USB)**
  - This micro hasn't USB. The nano board provide an USB-USART interface from FTDI company.

- One 16-bit **Timer/ Counter (TC)** with Auto-Reload and PWM
- Two 8-bit **Timer / Counter (TC)** with AR and PWM
- One 8-channel 10-bit **Analog-To-Digital Converter (ADC)**, 76.9ks/s
- SPI, USART, I2C



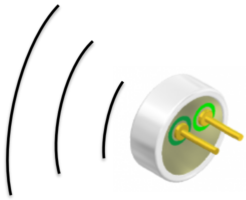
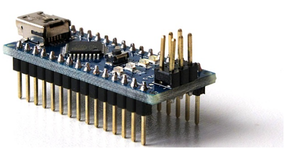
Arduino Nano ATmega328P schematics





Exercise 3: "Sound volume robot"

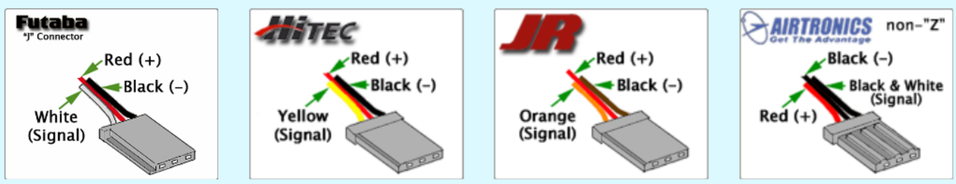
- measures sound volume and moves arm to indicate loudness
- microphone -> preamp -> ADC -> uC -> PWM output

("debugging", programming)

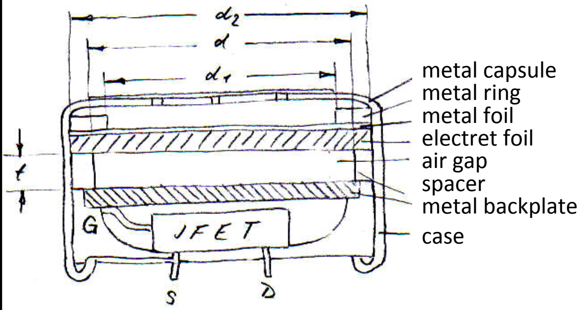
"RC" servos (Radio-Control Servo-Motors)

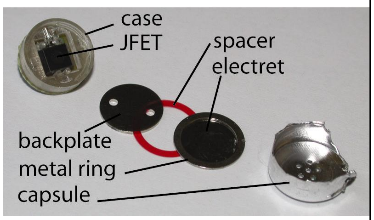
- Position controlled – Servo has internal position measurement and controller
- Rotation angle 120 degrees
- Pulse width from 1-2ms sets desired position
- Pulses must be sent at frequency 50-200Hz
- Pulse height >2V



Electret Microphone

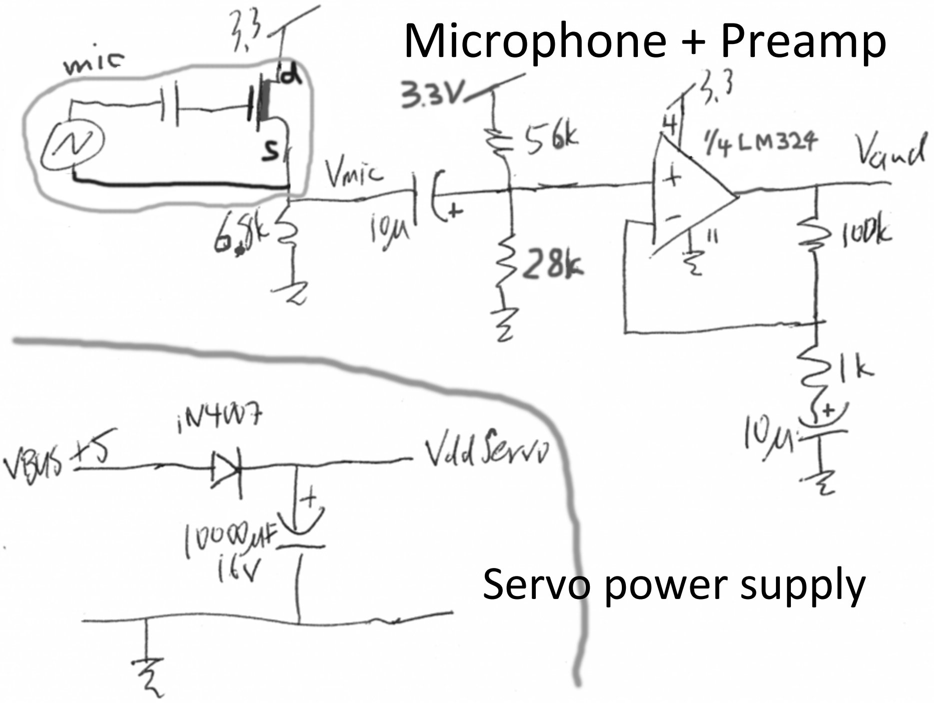
- Cheap (< 1$)
- Electret material, no polarization voltage is required
- Low-noise JFET buffer
- Metal foil is connected to source of the JFET through metal capsule





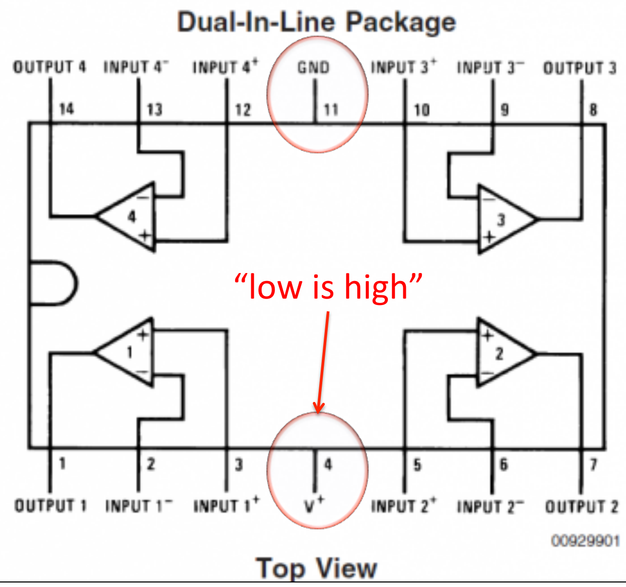
11

Microphone + Preamp



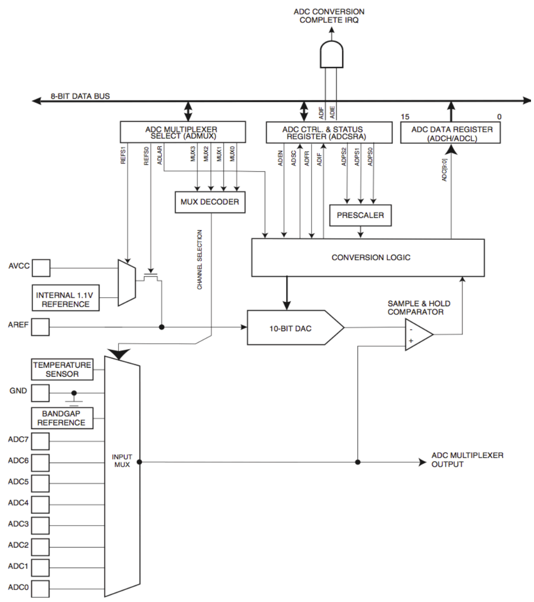
Servo power supply

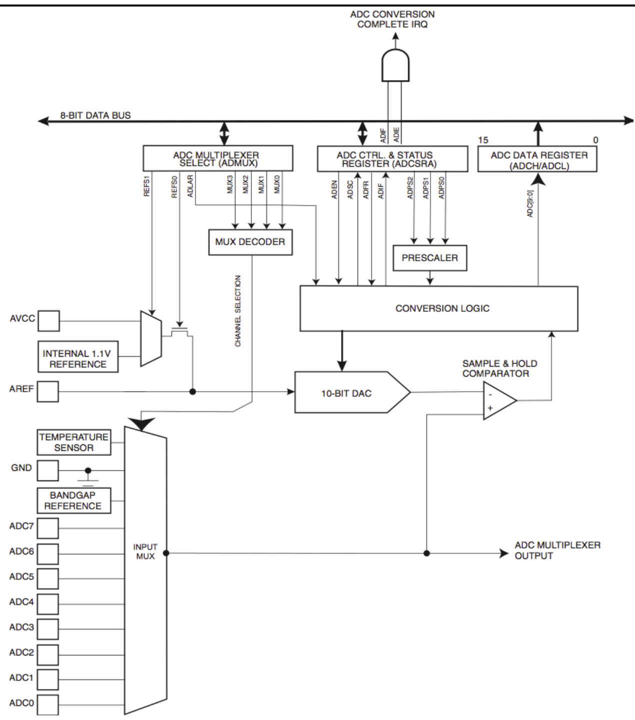
LM324 quad JFET opamp

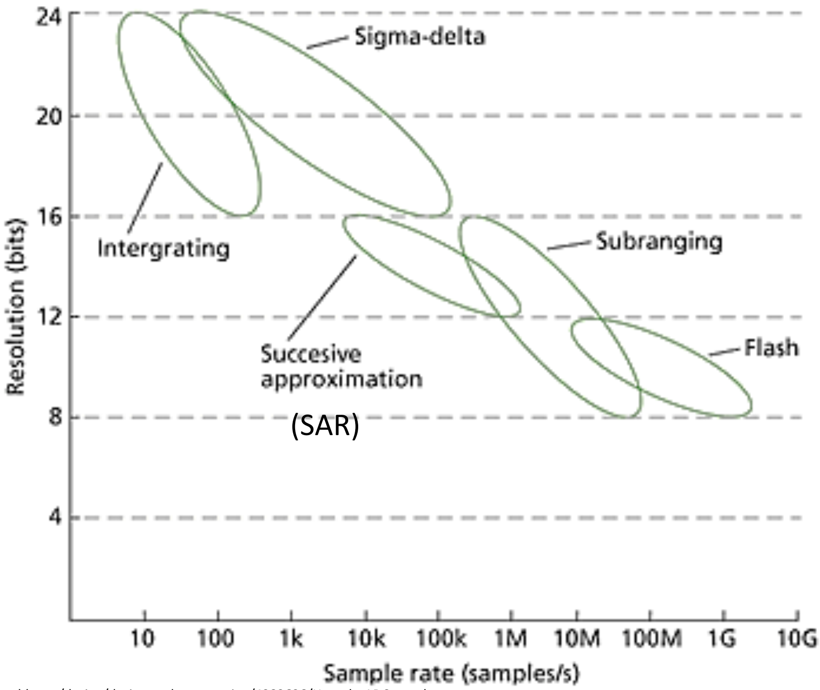


ATmega328P Analog to Digital converter

- 10-bit Successive approximation register (SAR) type
- 8 multiplexed single-ended input channels
- Internal Temp sensor
- Max combined sample rate 79.6ks/s
- Interrupt on End of Conversion.
- Triggered by:
  - External Interrupt Request 0
  - Timer 0
  - Timer 1
  - Analog Comparator

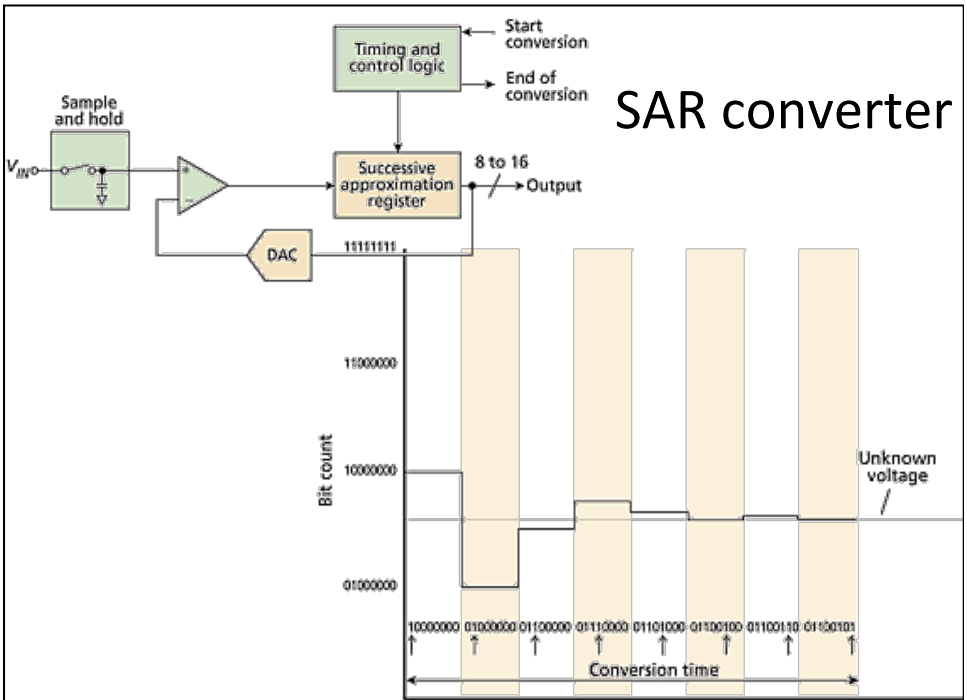






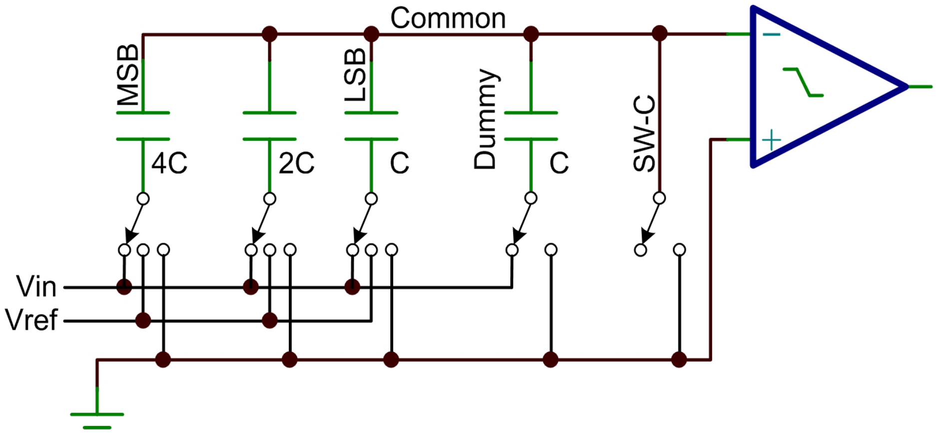
http://tmworld.com/design/design-and-prototyping/4383626/How-do-ADCs-work-

SAR converter

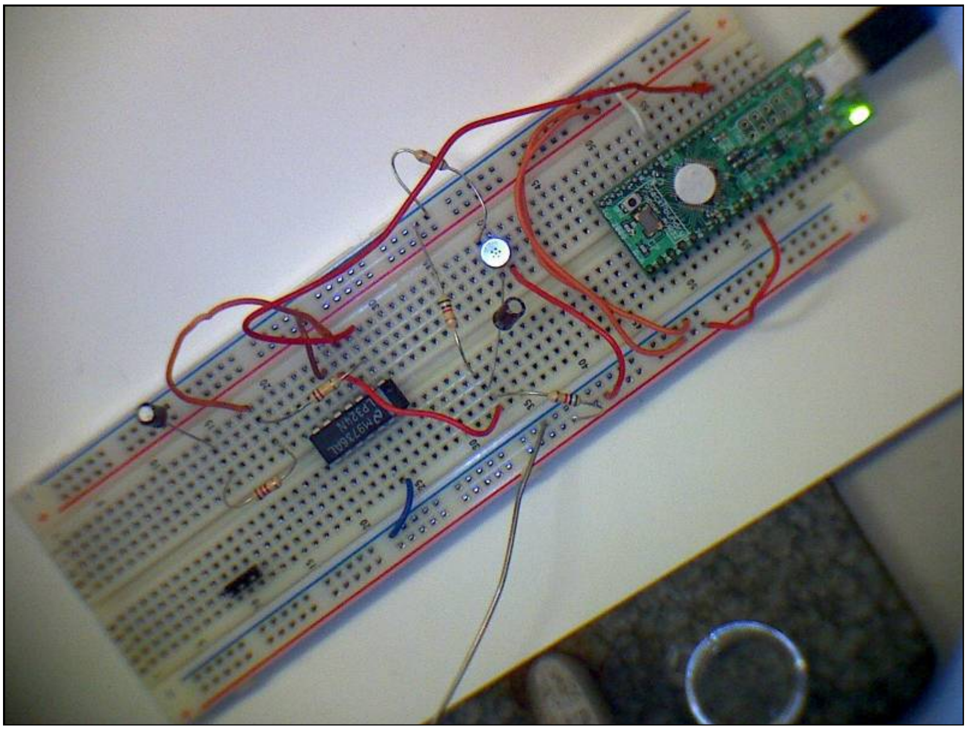


http://tmworld.com/design/design-and-prototyping/4383626/How-do-ADCs-work-

SAR DAC + comparator



http://i.cmpnet.com/planetanalog/2008/03/C0283-Figure2.gif



How to work in Arduino with ADC

- adc_data= analogRead(pin);

It takes time to convert the analog signal and this time is wasted.

- Interruptions: Allows you to execute other code while waiting for the ADC to finish:
  - Initialize the ADC on setup(): next slice
  - Capture the ADC end of conversion interrupt:

```
ISR (ADC_vect) {
        //your code
}
```

Example: declarations and setup()

```
// Testing interrupt-based analog reading
// ATMega328p

// Note, many macro values are defined in <avr/io.h> and
// <avr/interrupts.h>, which are included automatically by
// the Arduino interface

// High when a value is ready to be read
volatile int readFlag;

// Value to store analog result
volatile int analogVal;
```

```
// Initialization
void setup(){

  // clear ADLAR in ADMUX (0x7C) to right-adjust the result
  // ADCL will contain lower 8 bits, ADCH upper 2 (in last two bits)
  ADMUX &= B11011111;

  // Set REFS1..0 in ADMUX (0x7C) to change reference voltage to the
  // proper source (01)
  ADMUX |= B01000000;

  // Clear MUX3..0 in ADMUX (0x7C) in preparation for setting the analog
  // input
  ADMUX &= B11110000;

  // Set MUX3..0 in ADMUX (0x7C) to read from AD8 (Internal temp)
  // Do not set above 15! You will overrun other parts of ADMUX. A full
  // list of possible inputs is available in Table 24-4 of the ATMega328
  // datasheet
  ADMUX |= 8;
  // ADMUX |= B00001000; // Binary equivalent

  // Set ADEN in ADCSRA (0x7A) to enable the ADC.
  // Note, this instruction takes 12 ADC clocks to execute
  ADCSRA |= B10000000;

  // Set ADATE in ADCSRA (0x7A) to enable auto-triggering.
  ADCSRA |= B00100000;

  // Clear ADTS2..0 in ADCSRB (0x7B) to set trigger mode to free running.
  // This means that as soon as an ADC has finished, the next will be
  // immediately started.
  ADCSRB &= B11111000;

  // Set the Prescaler to 128 (16000KHz/128 = 125KHz)
  // Above 200KHz 10-bit results are not reliable.
  ADCSRA |= B00000111;

  // Set ADIE in ADCSRA (0x7A) to enable the ADC interrupt.
  // Without this, the internal interrupt will not trigger.
  ADCSRA |= B00001000;

  // Enable global interrupts
  // AVR macro included in <avr/interrupts.h>, which the Arduino IDE
  // supplies by default.
  sei();

  // Kick off the first ADC
  readFlag = 0;
  // Set ADSC in ADCSRA (0x7A) to start the ADC conversion
  ADCSRA |=B01000000;
}
```

Loop() and ISR:

http://www.glennsweeney.com/tutorials/interrupt-driven-analog-conversion-with-an-atmega328p

```
// Processor loop
void loop(){

  // Check to see if the value has been updated
  if (readFlag == 1){

    // Perform whatever updating needed

    readFlag = 0;
  }

  // Whatever else you would normally have running in loop().

}


// Interrupt service routine for the ADC completion
ISR(ADC_vect){

  // Done reading
  readFlag = 1;

  // Must read low first
  analogVal = ADCL | (ADCH << 8);

  // Not needed because free-running mode is enabled.
  // Set ADSC in ADCSRA (0x7A) to start another ADC conversion
  // ADCSRA |= B01000000;
}
```

http://meettechniek.info/embedded/arduino-analog.html

```
int marker = 12;    // marker output pin
int analogPin = 3;  // analog input pin
int aval = 0;       // analog value

void setup() {
  pinMode(marker, OUTPUT);   // pin = output
}

void loop() {
  bitSet(PORTB, 4);               // marker high
  aval = analogRead(analogPin);   // sample signal
  bitClear(PORTB, 4);             // marker low
  aval = analogRead(analogPin);   // sample signal
}
```

Code 1: Timing the execution time of the analogRead function.

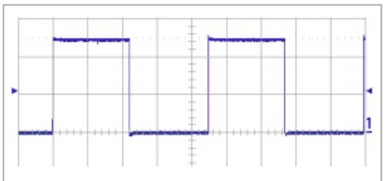


Fig. 2: Execution time of the analogRead function. (Timebase: 50 μs/div)

```
int marker = 12;    // marker output pin
int aval = 0;       // analog value

void setup() {
  pinMode(marker, OUTPUT); // pin = out
  DIDR0 = 0x3F;            // digital i
  ADMUX = 0x43;            // measuring
  ADCSRA = 0xAC;           // AD-conver
  ADCSRB = 0x40;           // AD channe
  bitWrite(ADCSRA, 6, 1);  // Start the
  sei();                   // set inter
}

void loop() {
}

/*** Interrupt routine ADC ready ***/
ISR(ADC_vect) {
  bitClear(PORTB, 4); // marker low
  aval = ADCL;        // store lower byte ADC
  aval += ADCH << 8;  // store higher bytes ADC
  bitSet(PORTB, 4);   // marker high
}
```

Code 2: Measure the execution time of reading the ADC value.

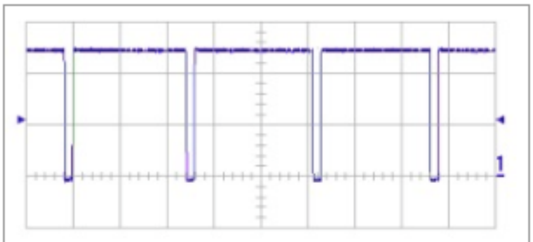


Fig. 3: Free running analog conversion with the division factor set at 16. (Timebase: 5 μs/div)