



SELECTING YOUR MICROCONTROLLER

CMPN-445

BASEM IBRAHEEM

BASEM@ENG.CU.EDU.EG

BASEMIBRAHEEM@GMAIL.COM



AGENDA

- BASES FOR SELECTING A MICROCONTROLLER
- CORTEX A-R-M
- SELECTOR TOOLS
- INTRODUCTION TO ARDUINO
 - WHY ARDUINO
 - SHIELDS
 - CONNECTION CONSIDERATIONS
 - CONSTRUCTION

BASES FOR SELECTING A MICROCONTROLLER

- DO I NEED A MICROCONTROLLER OR A MICROPROCESSOR ?
 - ONBOARD WI-FI / BLUETOOTH / FLASH / LARGE I/O / GRAPHICS UNIT
 - SPI / I2C / UART
- BITS SIZE (8/16/32) : DATA BUS , REGISTER SIZE , PORT SIZE –
ADDRESS BUS (PAGING REGISTER)
- OSCILLATOR FREQUENCY
- **PACKAGING** FOR SIZE AND ACCESSIBILITY

BASES FOR SELECTING A MICROCONTROLLER

- POWER CONSUMPTION
- RAM / ROM SIZES
- I/O PORTS
- COST PER UNIT
- MANUFACTURE AVAILABILITY
- DEVELOPMENT PLATFORM / INSTRUCTION SET / SIMULATOR / EMULATOR
- OPERATING ENVIRONMENT : MOBILE / FIXED , STABLE / SHOCKS ,
TEMPERATURE , MOIST

Results 1-11 of Total 11 Page 1

Compare Devices: 0

Share and export the results or save them, including the configurations, for viewing later.

Easily filter the list by updating the parameters to meet your needs.

Select and sort parameters based on your requirements.

463

11

 Clear Filters

sliders



	More Info	Flash (Kbytes)	Pin Count	Max. Operating Frequency	CPU	SRAM (Kbytes)	EEPROM (Bytes)	Max I/O Pins	picoPower	Operating Voltage (Vdc)	Temp. Range (deg C)
<input type="checkbox"/> AT32UC3A0E12AV2		512	144	60	32-bit AVR	64	0	109	No	3.0-3.6 or (1.85-1.95+3.0-3.6)	-40 to 85
<input type="checkbox"/> AT32UC3A1S12		512	100	60	32-bit AVR	64	0	69	No	3.0-3.6 or (1.85-1.95+3.0-3.6)	-40 to 85
<input type="checkbox"/> AT32UC3A1E12AV2		512	100	60	32-bit AVR	64	0	69	No	3.0-3.6 or (1.85-1.95+3.0-3.6)	-40 to 85
<input type="checkbox"/> AT32UC3C0E12C		512	144	60	32-bit AVR	68	0	123	No	3.0 to 3.6 or 4.5 to 5.5	-40 to 85
<input type="checkbox"/> AT32UC3C0S12CAU		512	144	60	32-bit AVR	68	0	123	No	3.0 to 3.6 or 4.5 to 5.5	-40 to 85
<input type="checkbox"/> AT32UC3C1S12C		512	100	60	32-bit AVR	68	0	81	No	3.0 to 3.6 or 4.5 to 5.5	-40 to 85
<input type="checkbox"/> AT9SAM000B12		512	217	180	ARM9	32	0	96	No	1.65 to 1.95	-40 to 85
<input type="checkbox"/> ATSAMD08C		512	100	64	Cortex M0	96	-	63	-	1.65 to 3.6	-40 to 85
<input type="checkbox"/> ATSAMD09C		1024	100	100	Cortex M0	128	0	79	No	1.65 to 3.6	-40 to 85
<input type="checkbox"/> ATSAMD09C		1024	100	64	Cortex M0	96	-	63	-	1.65 to 3.6	-40 to 85
<input type="checkbox"/> ATSAMD09E		512	100	100	Cortex M0	96	-	103	-	1.65 to 3.6	-40 to 85

Select multiple devices for easy comparison.

Quick links allow easy access to device overviews, datasheets, samples and buying.

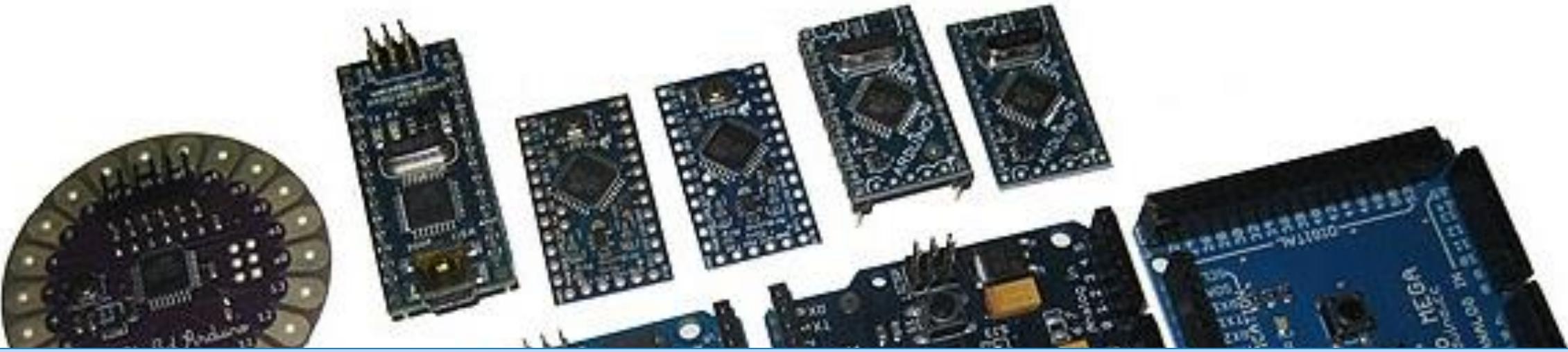
enta hena bt7dded el specs el enta 3auzhom w hwa by3mlk filter 3la asas el values el enta 3auzo.

CORTEX A – R – M BY SILICON LABS

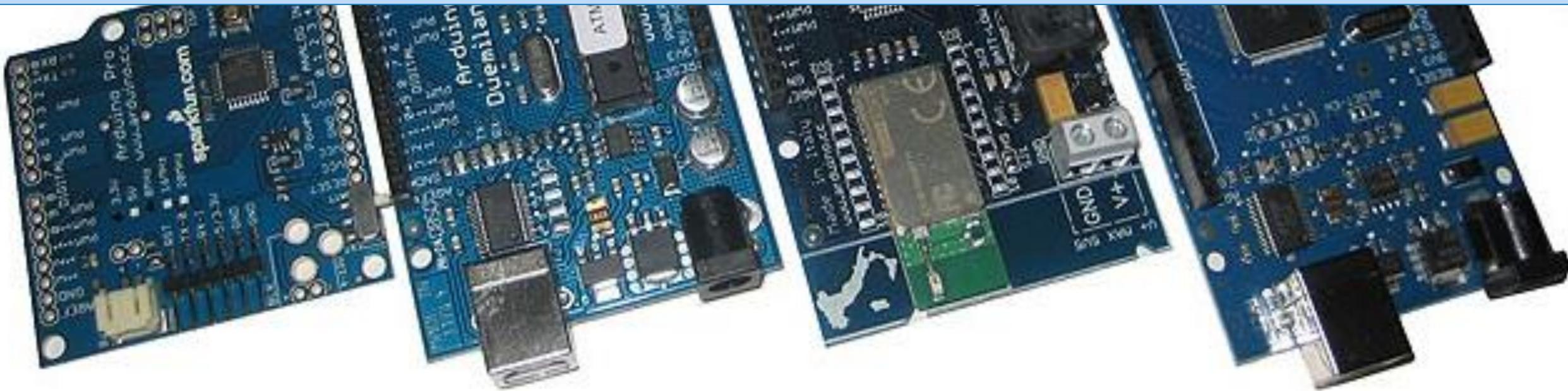
- CORTEX A
 - MICROPROCESSOR FOR PERFORMANCE INTENSIVE SYSTEMS
 - RUN AND OS SUCH AS ANDROID
 - MULTIPLE THREADED
 - COMPLEX HIGHSPEED INTERFACES : ETHERNET - MICROSD - USB - ON BOARD WI-FI
 - RASPBERRY PI 2 / SMARTPHONES

CORTEX A – R – M BY SILICON LABS

- CORTEX R
 - MICROPROCESSOR FOR HIGH PERFORMANCE REAL-TIME APPLICATIONS
 - BLUE-RAY – AIRBAGS – BRAKING SYSTEM – ENGINE MANAGEMENT
- CORTEX M
 - MICROCONTROLLER FOR EMBEDDED APPLICATIONS
 - RUN YOUR CODE OR BARE CODE AS BOOT LOADER
 - (0) SINGLE THREADED
 - ONBOARD I2C , SPI , UART , USB
 - YOUR CODE SHOULD HANDLE ALL ACTIVITIES IN A SINGLE THREAD (CODING IS HARDER)



INTRODUCTION TO ARDUINO



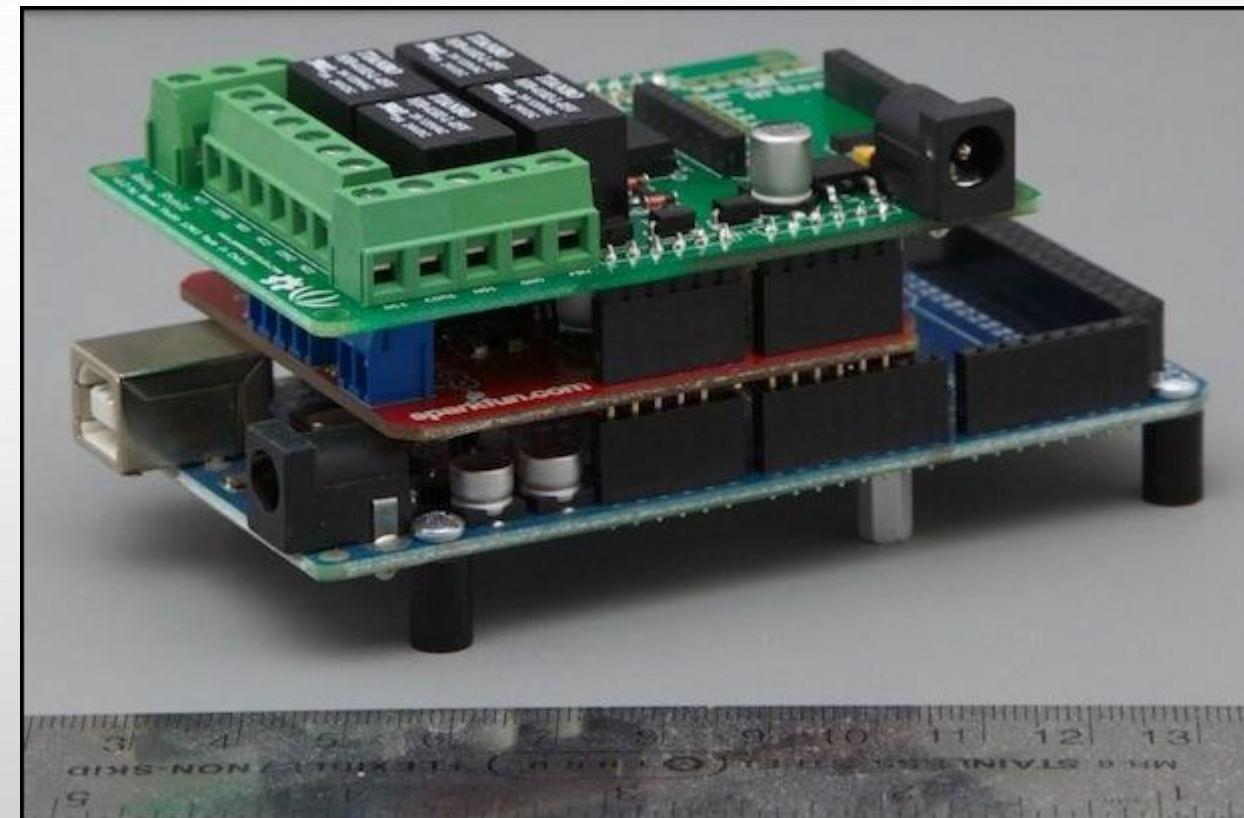
WHY ARDUINO ?

- IMPROVEMENT OVER OTHER MICRO-CONTROLLERS FOR BEGINNER USAGE
- INEXPENSIVE
- IDE ON MULTIPLE PLATFORMS (MAC, PC, LINUX)
- SIMPLE, CLEAR PROGRAMMING ENVIRONMENT: “C”
- OPEN SOURCE AND EXTENSIBLE SOFTWARE/HARDWARE
- ARDUINO SHIELDS
- COMPATIBLE WITH EMBEDDED SYSTEM’S COURSE LABS

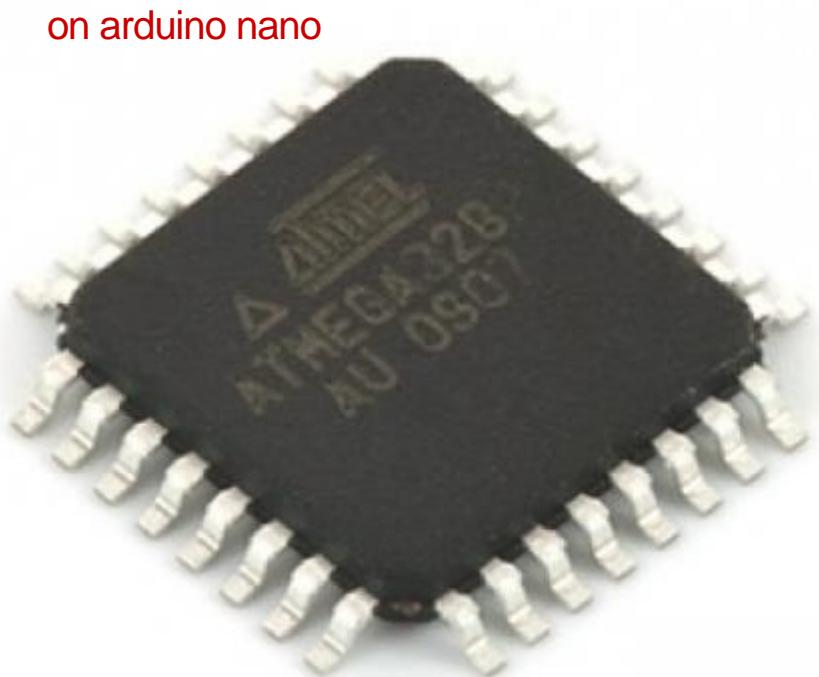
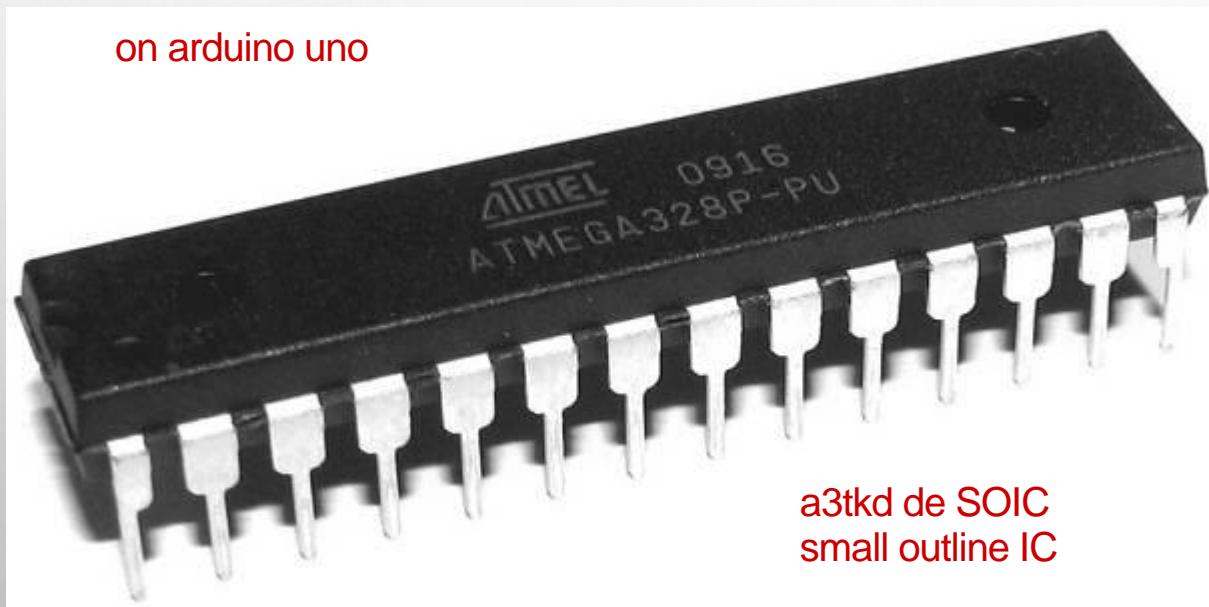
hadaf el relay eno ye3ml regulation lel current -> w da 34an lw bt3aml m3 devices btt3aml m3 power akbur mn el ana shghal 3leh fa my7r2lesh el chip bta3ty.

ARDUINO SHIELDS

- VGA CAMERA
- GPS
- LCD DISPLAYS
- ETHERNET
- WIFI
- MOTOR CONTROL (DC, STEPPER)
- Load Cell
- Accelerometer/
Gyros
- LED displays
- Memory Cards
- Weather Sensors
- Relays



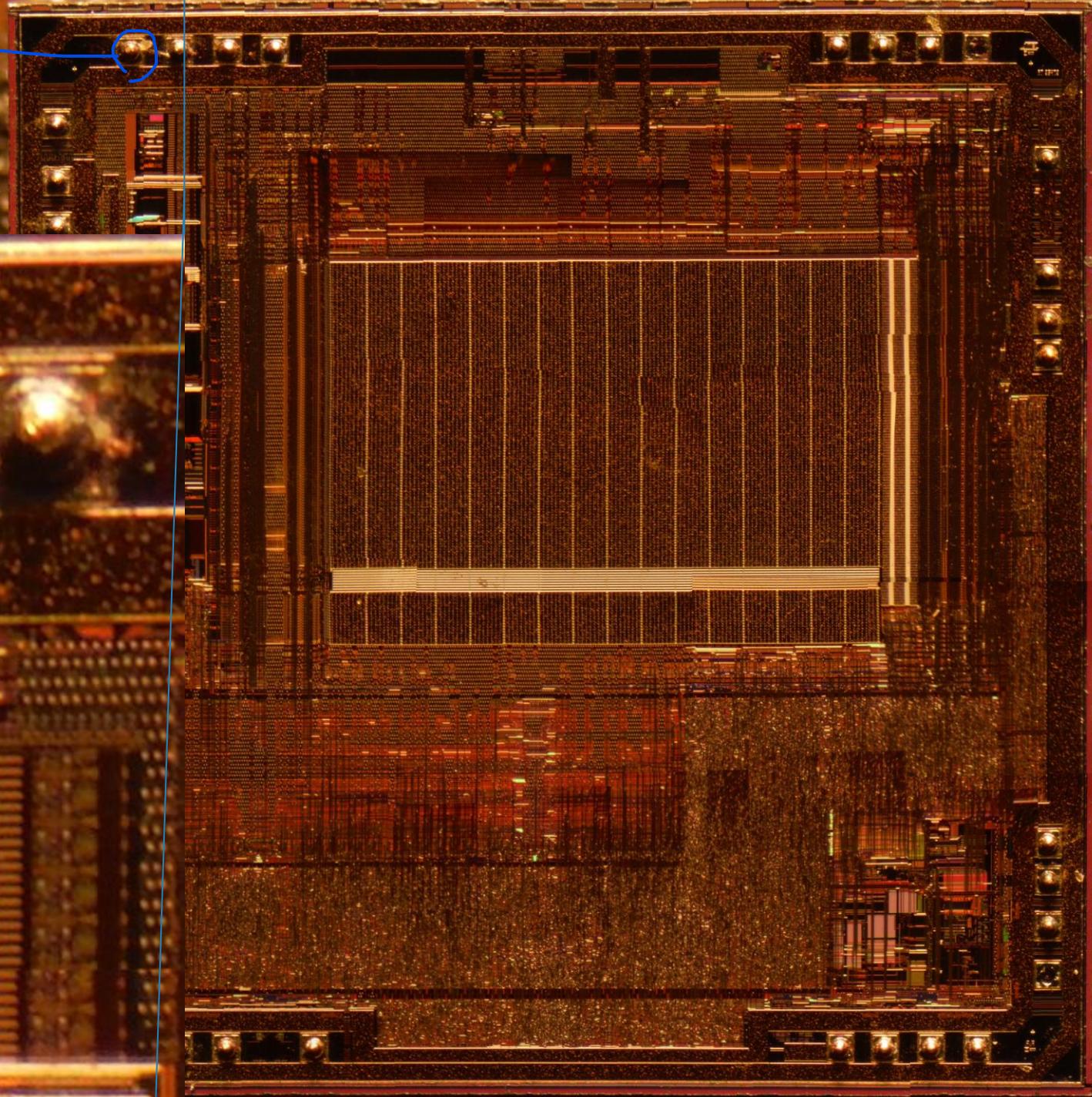
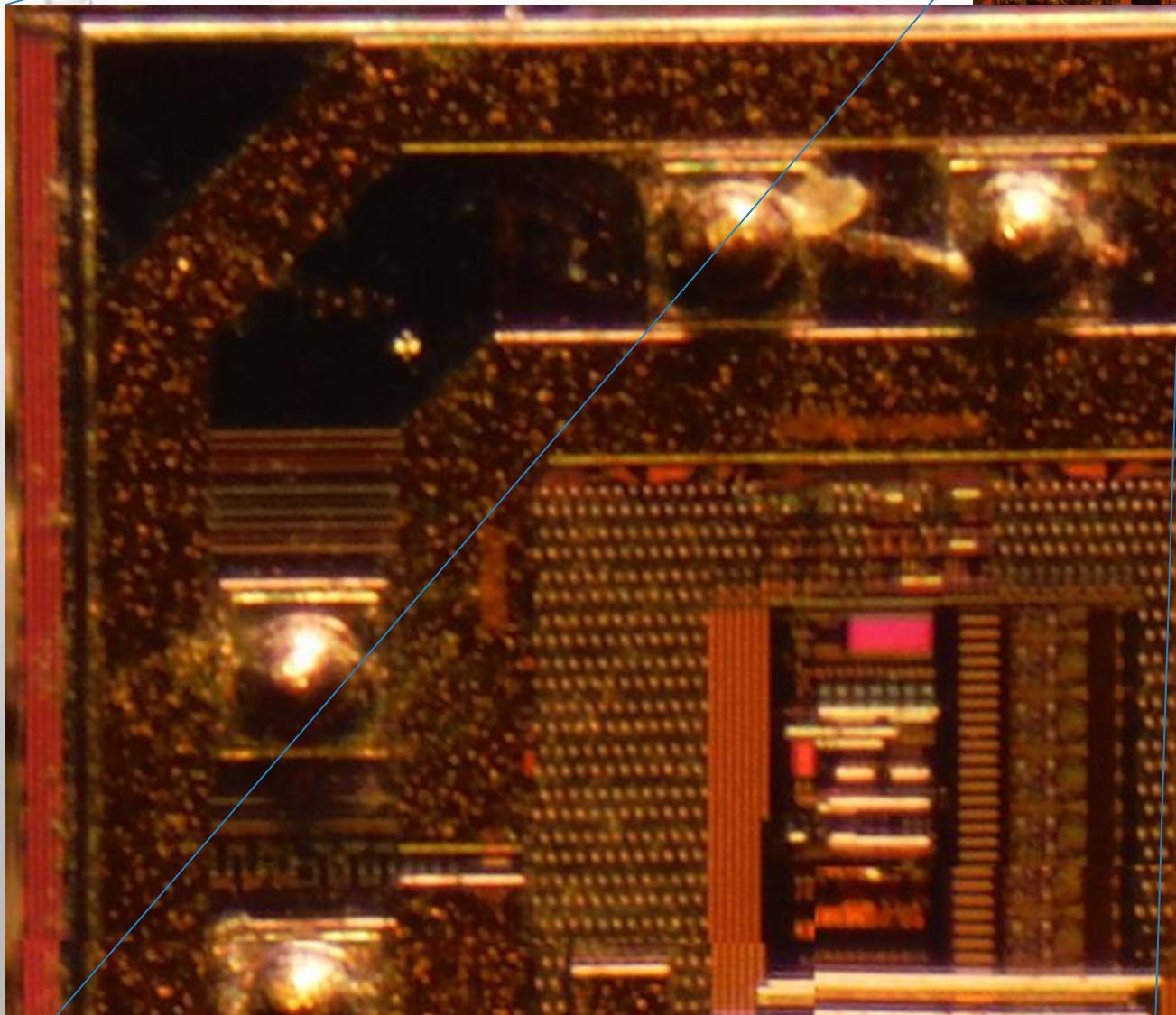
CONSTRUCTION : ATMEGA328P DIP / QFP

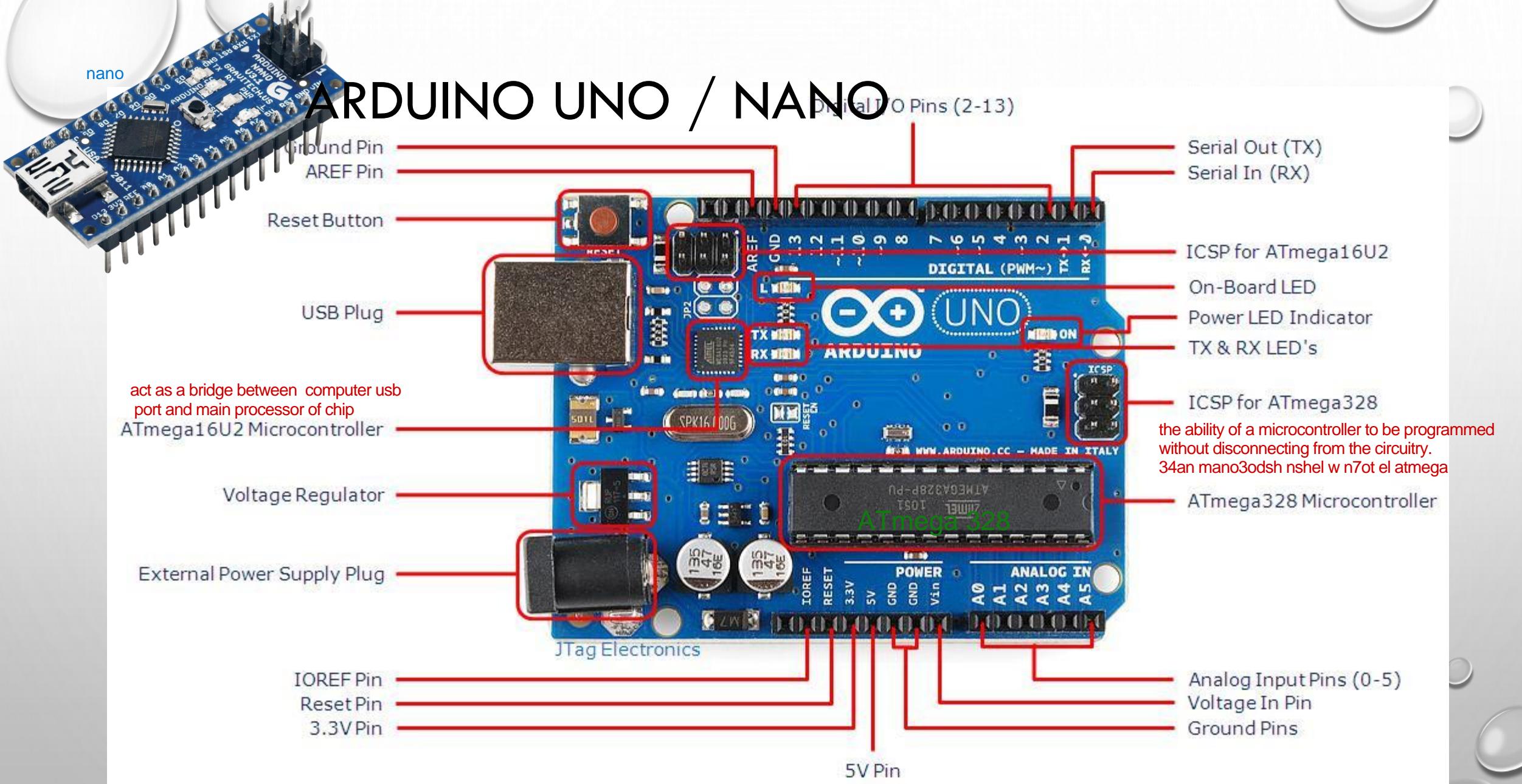


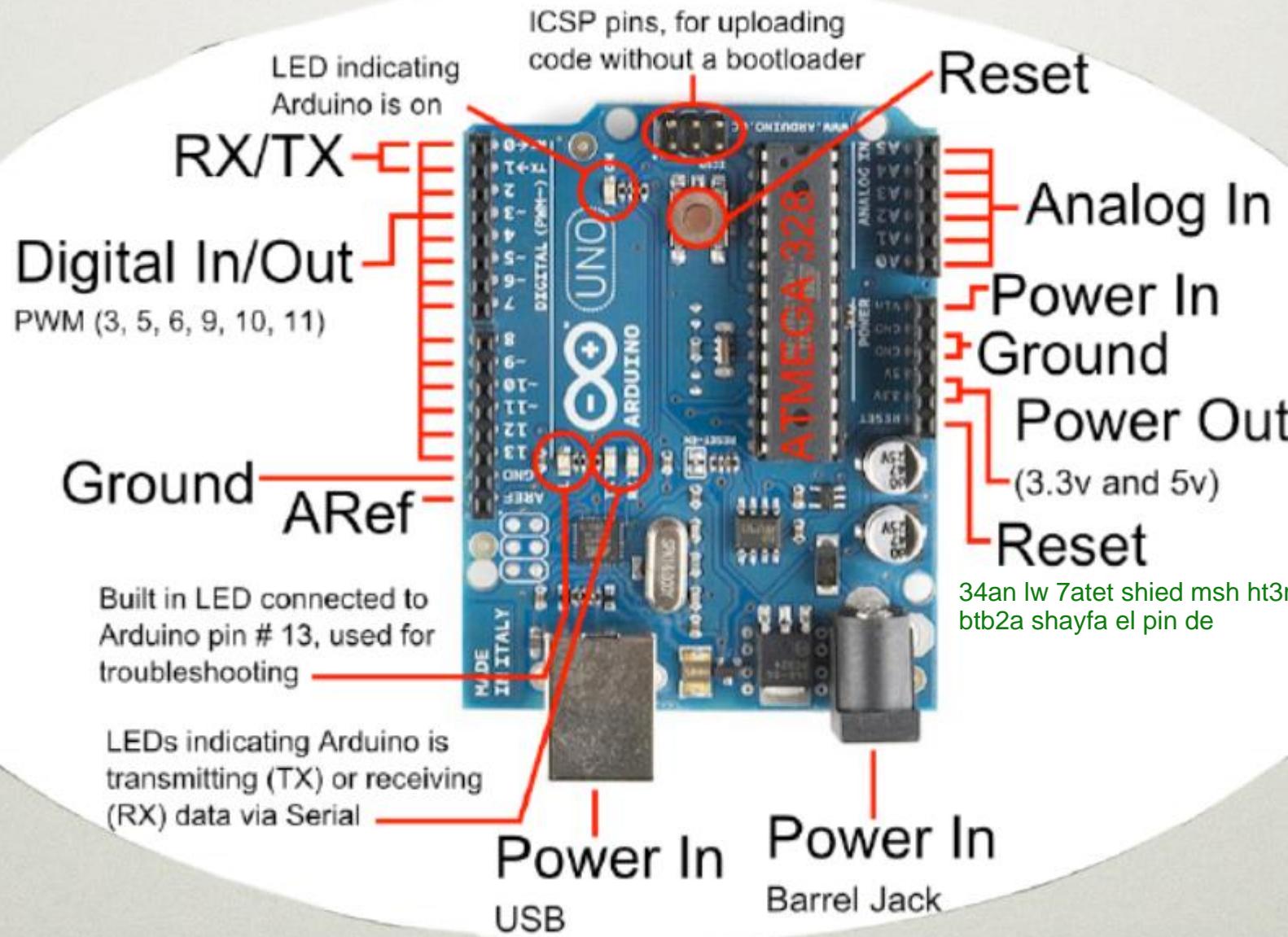
quad flat chip
lecture 1 slide 30

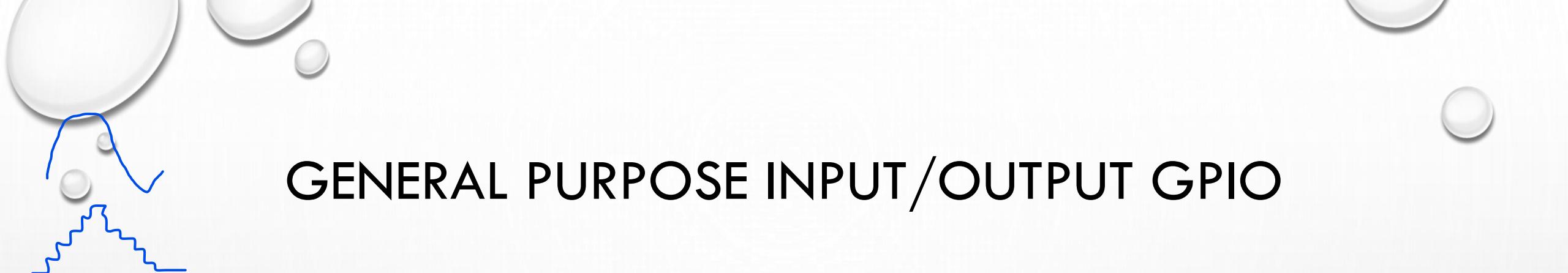
ATMEGA328P QFP

pins









GENERAL PURPOSE INPUT/OUTPUT GPIO

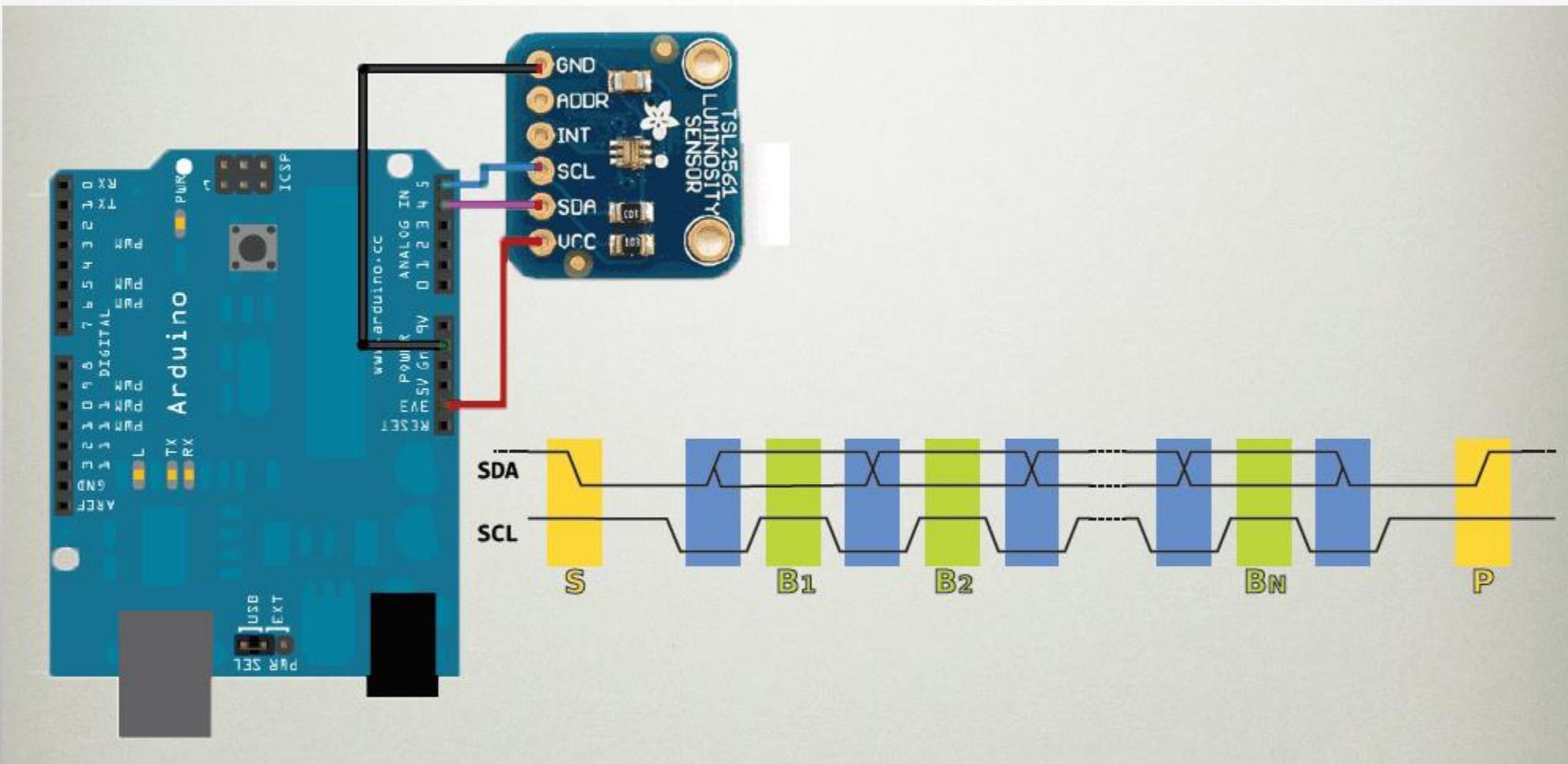
- CAN BE USED AS DIGITAL INPUT OR OUTPUT
- EACH PIN UNIQUELY ASSIGNABLE
- “ANALOG” (PWM OUTPUT ON DIGITAL PINS 3,5,6,9,10,11)
- ANALOG PINS 0-5 CAN ALSO BE USED AS GPIO general purpose input output

hwa msh real analog, bs
ehna bn3ml simulation leha.

fekrt el pwm eny a3ml timer w b7ot value (1) 3la el pin le moda zamanya mo7dda, fa kol ma el w2t da zad, enta bt7s eno real 1, w kol ma t2l bt7s eno tafa w lw kema fl nos, fa enta bygelk e7sass enk kema analog in between kda ben el 0,5
fa lw enta mwslo 3la led msln htla2y el nor msh akwa 7aga, w nfs el klam lw mator, msh htl2eh full speed.

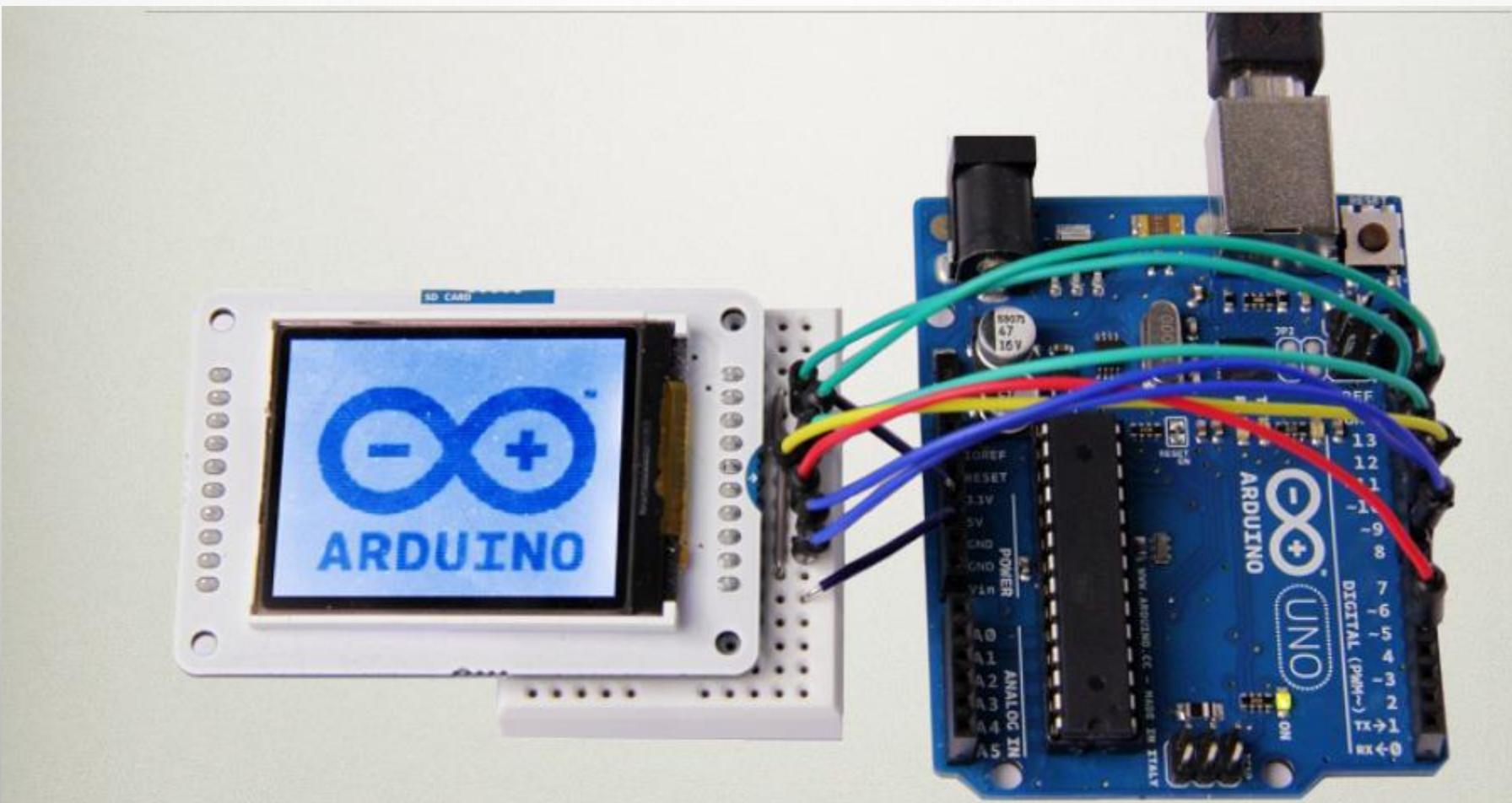
doctor skip

I₂C / 2WI

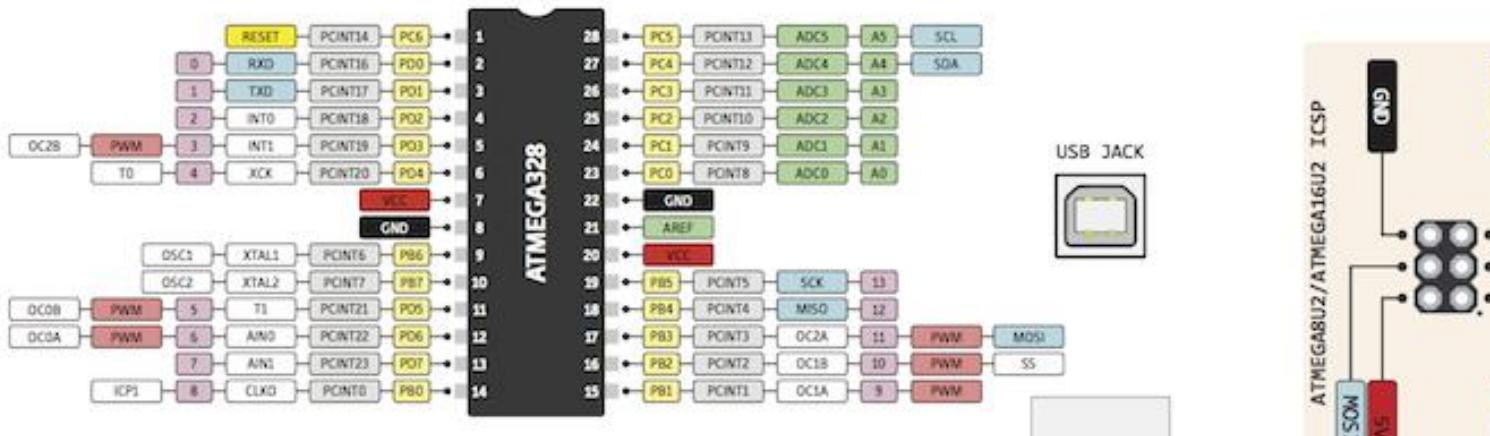


SPI

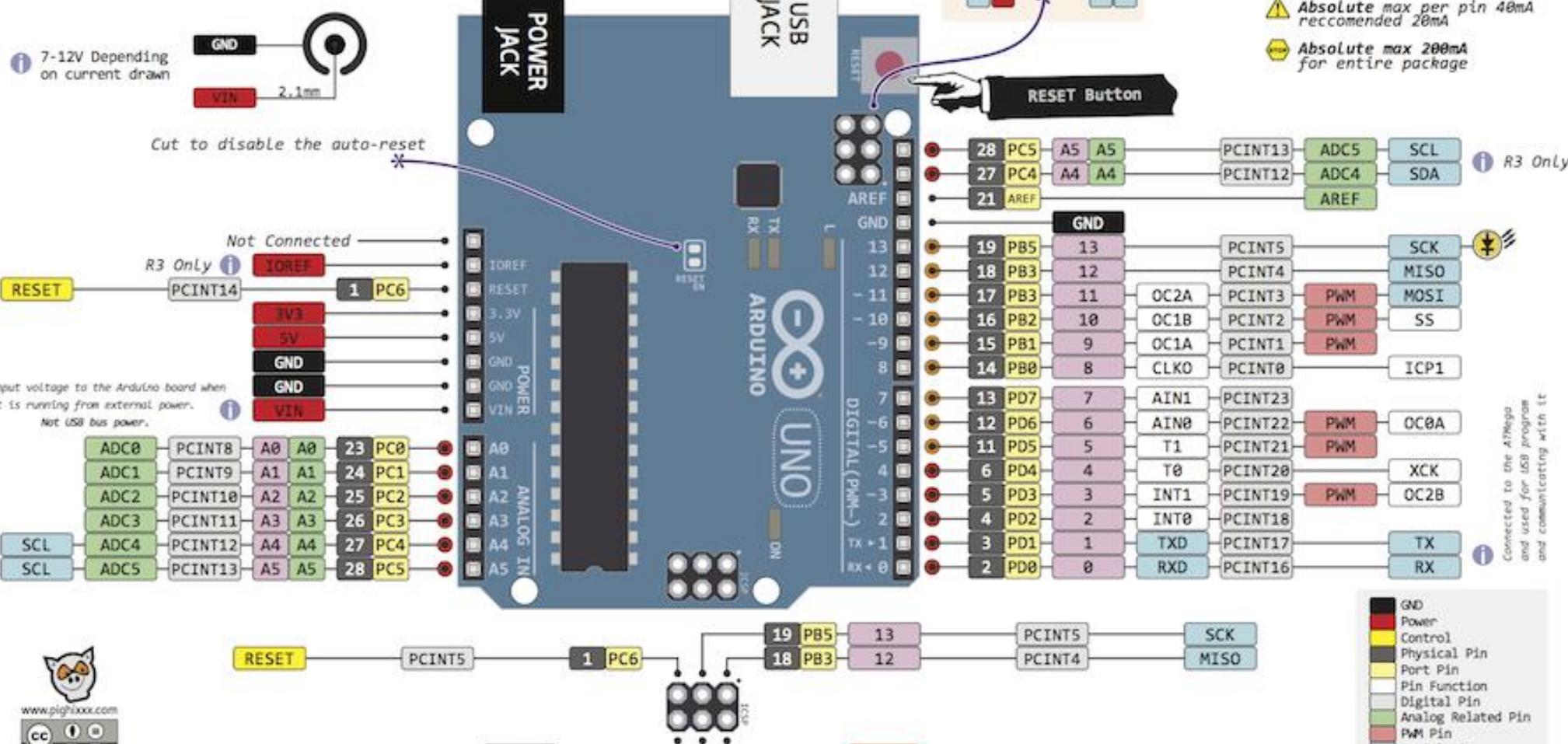
Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances , master and slaves



THE
DEFINITIVE
ARDUINO
UNO
PINOUT DIAGRAM

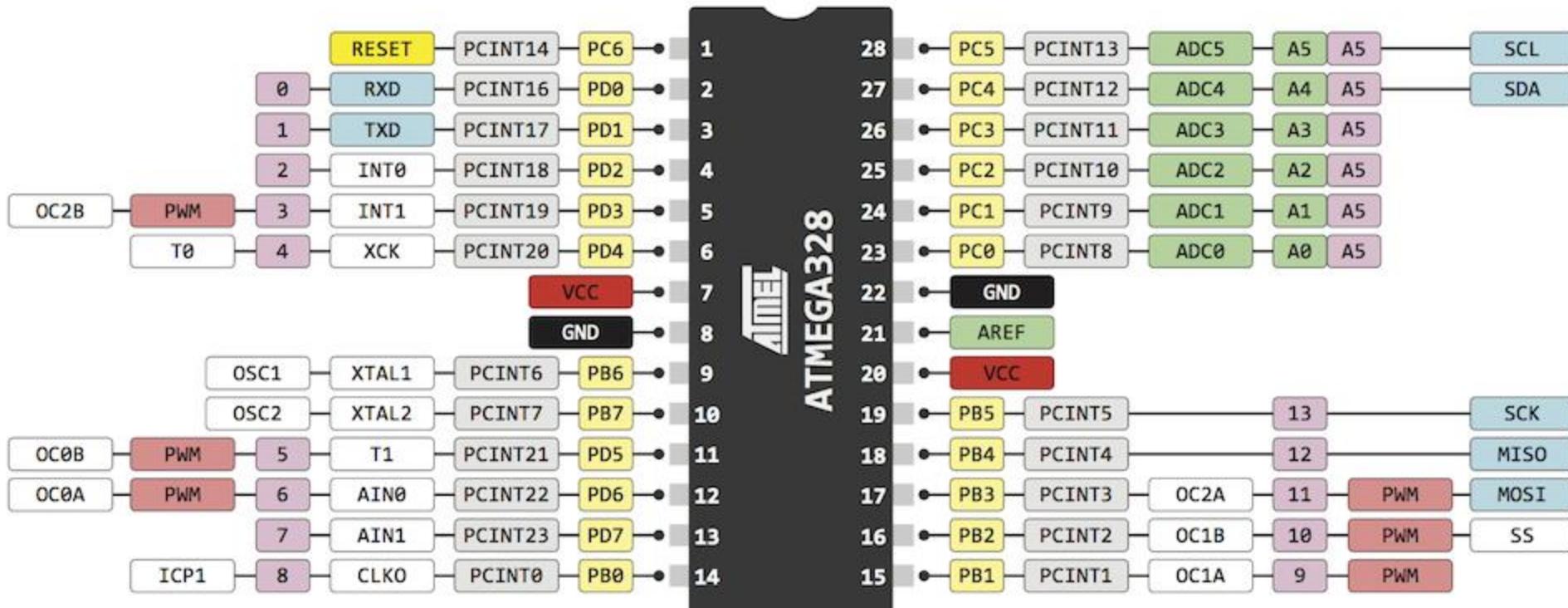


data sheet



THE
DEFINITIVE
ATMEGA328
&Arduino
PINOUT DIAGRAM

	GND
	Power
	Control
	Physical Pin
	Port Pin
	Pin Function
	Digital Pin
	Analog Related Pin
	PWM Pin
	Serial Pin
	IDE



www.pighoox.com



18 FEB 2013

ver 2 rev 0 - 19.02.2013

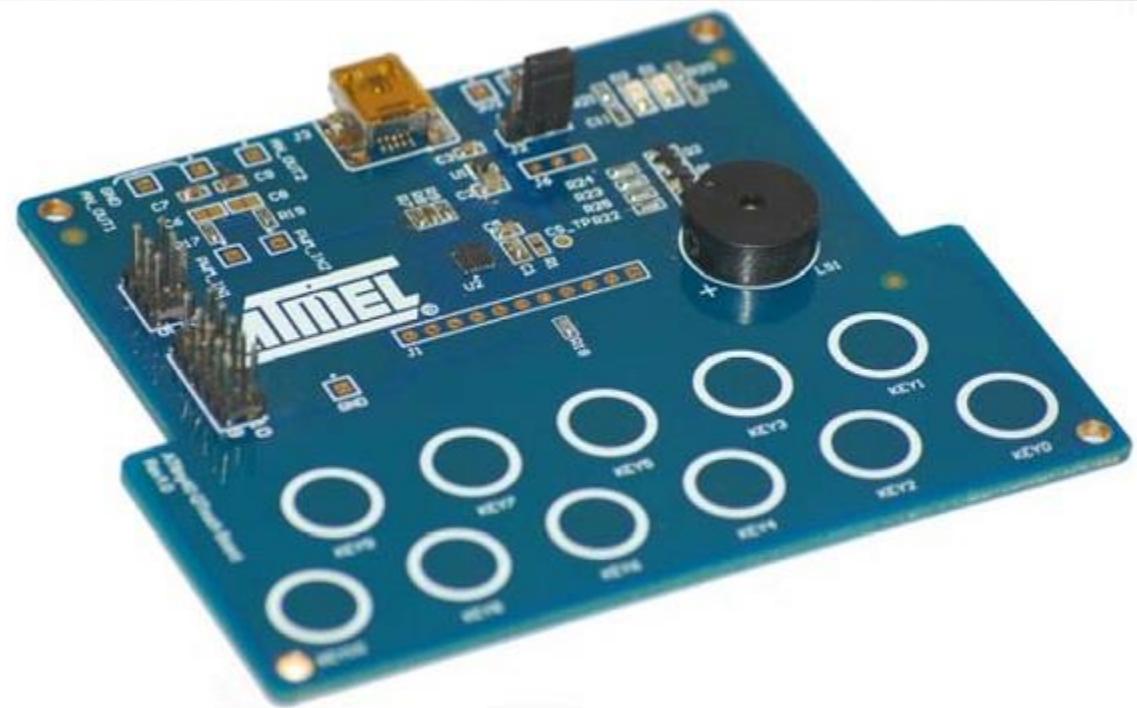
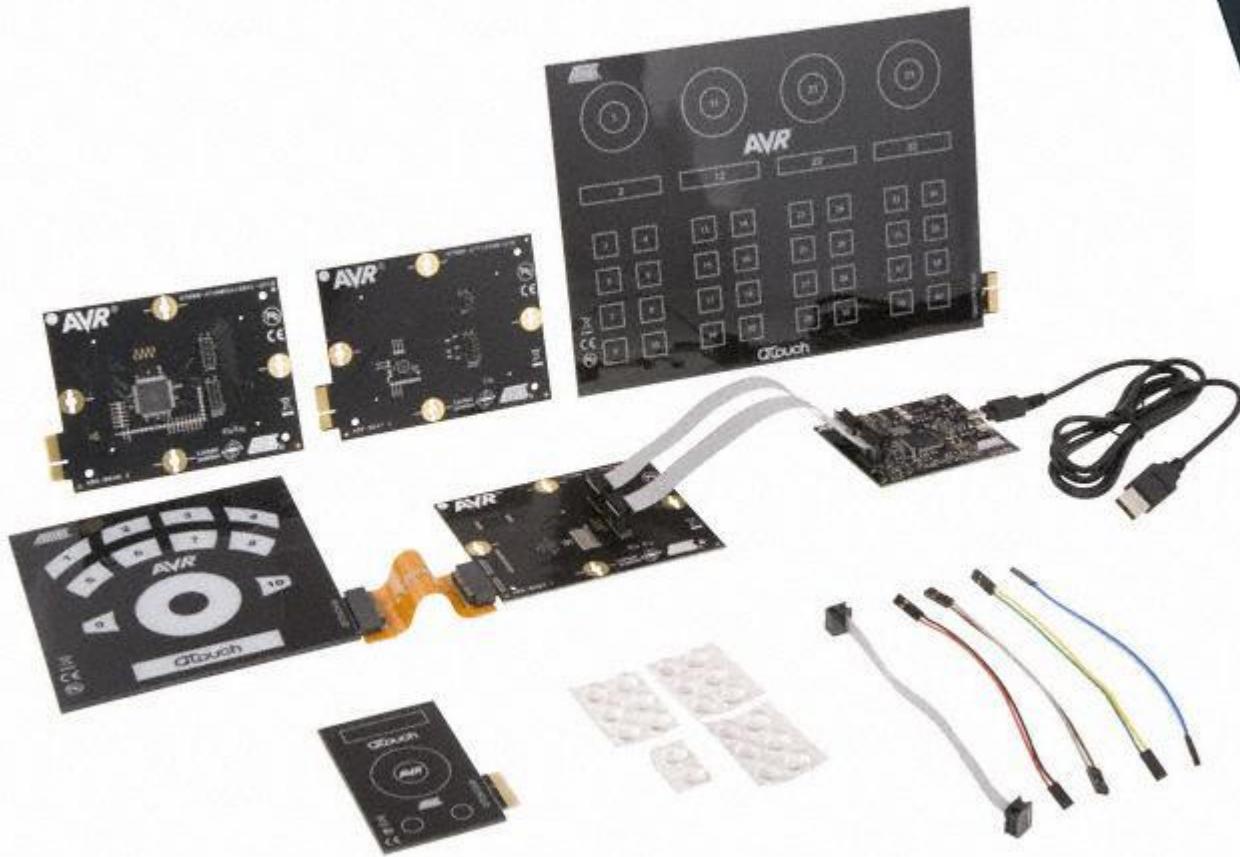
CONNECTING MICROCONTROLLERS TO OUTSIDE WORLD

INTERFACING

TYPES OF CONNECTIONS

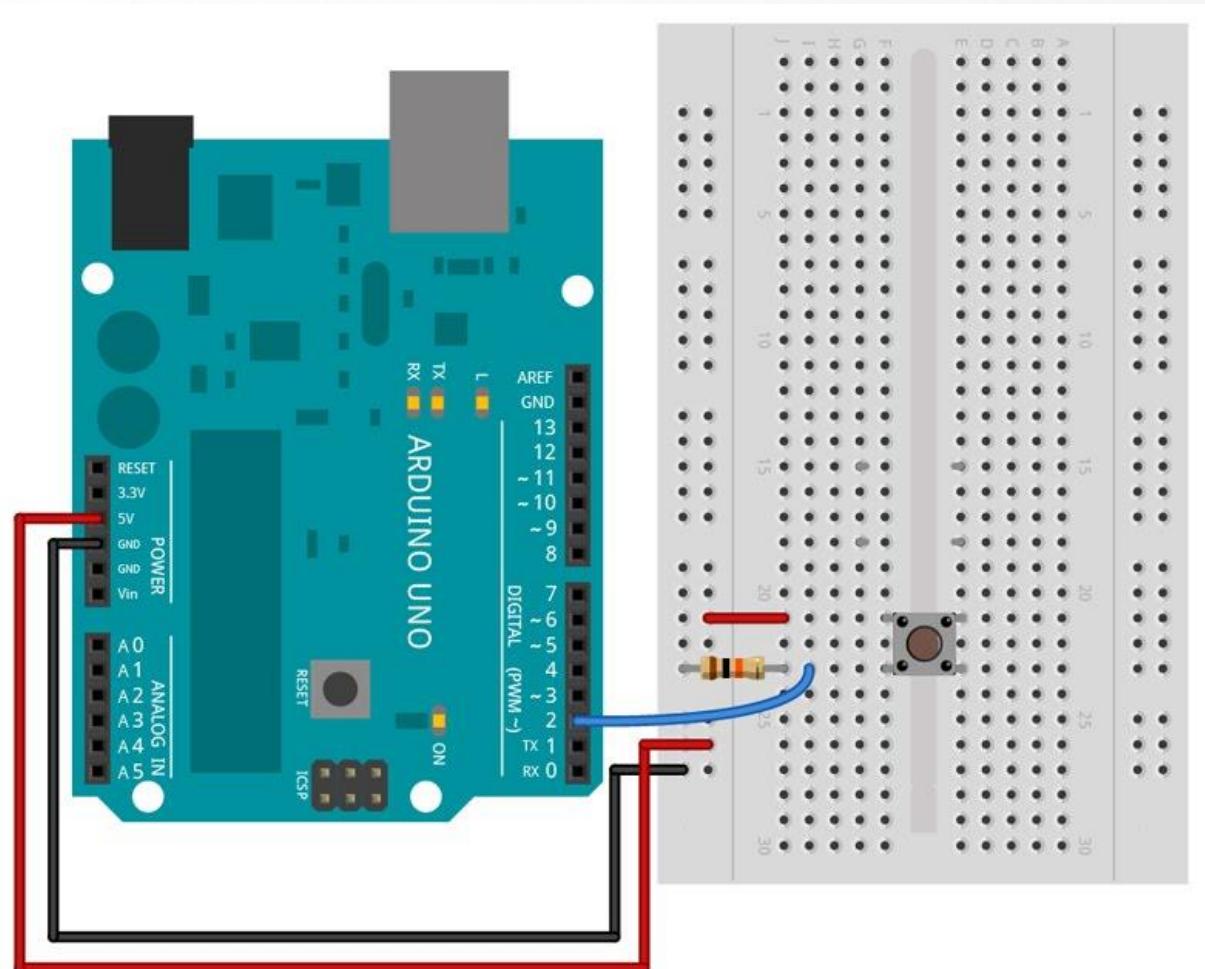
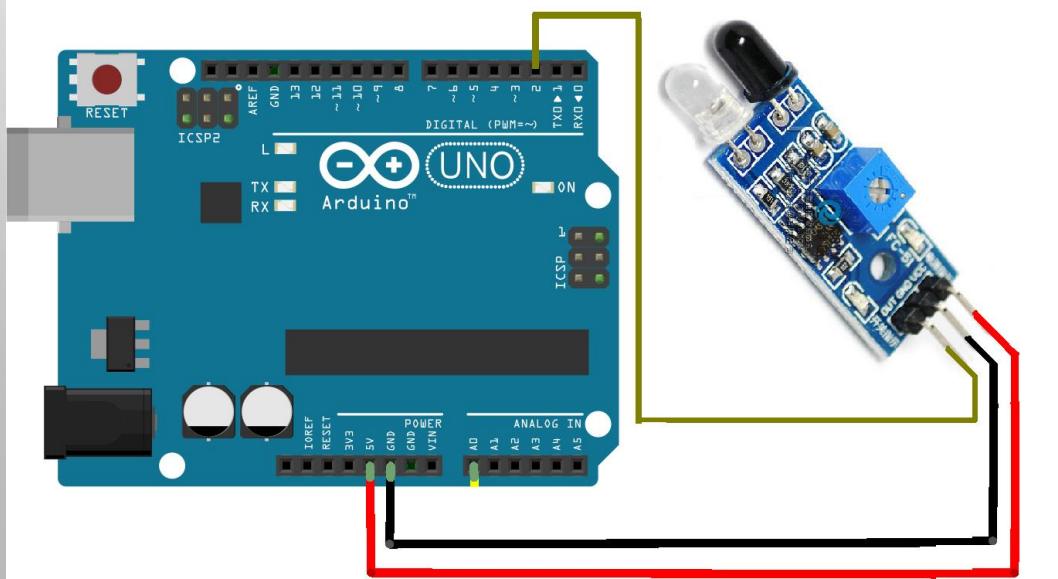
- MICROCONTROLLER DRIVER
 - CLOCK (INTERNAL/EXTERNAL)
 - POWER SOURCE
- INPUTS (GPIOS/PROTOCOLS)
- OUTPUTS
- INTERNAL WIRING (TIMERS/COUNTERS)
- INTERNAL MEMORY





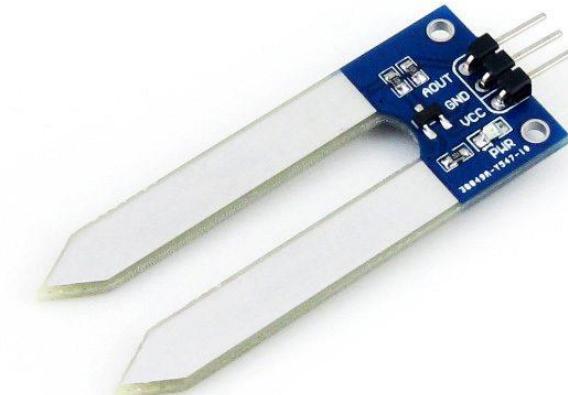
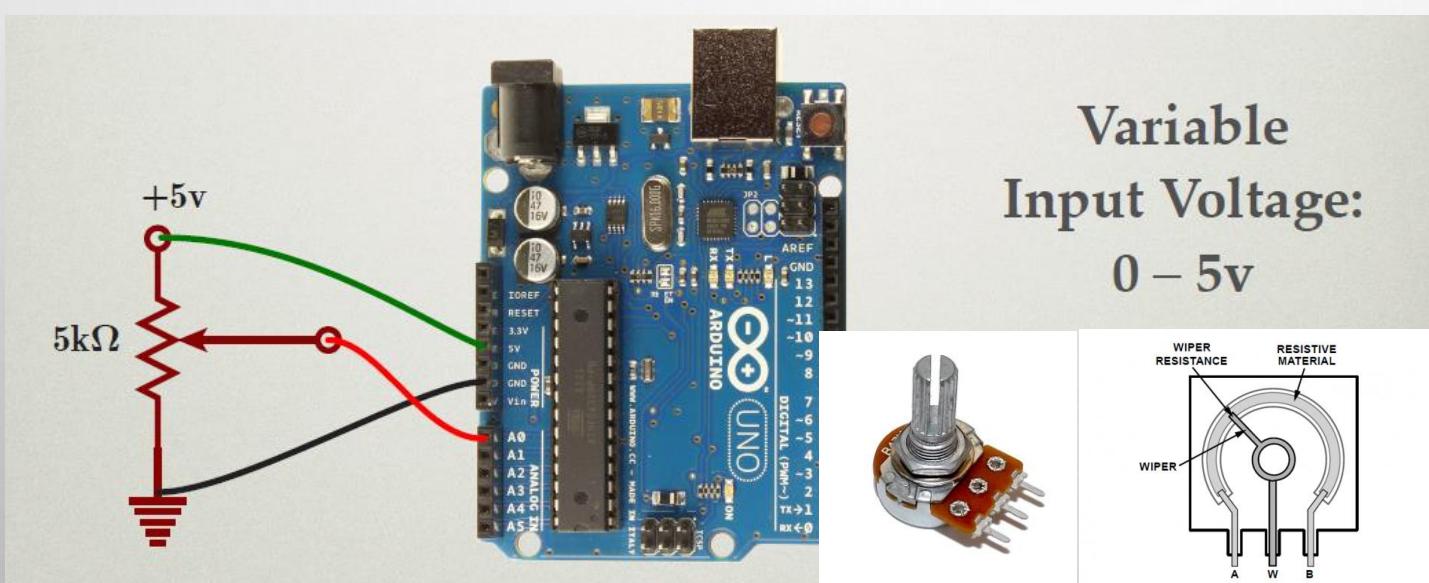
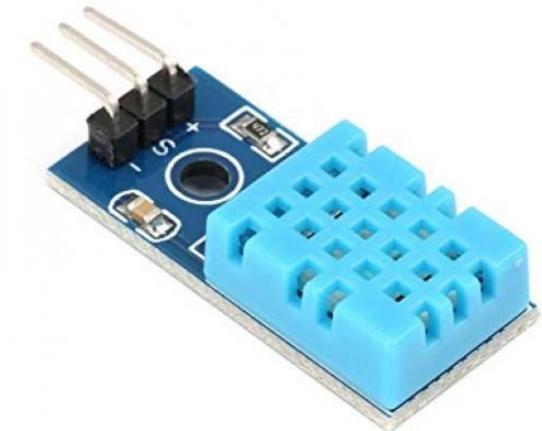
DIGITAL INPUT

- BUTTON
- SWITCH
- IR SENSOR



ANALOGUE INPUT

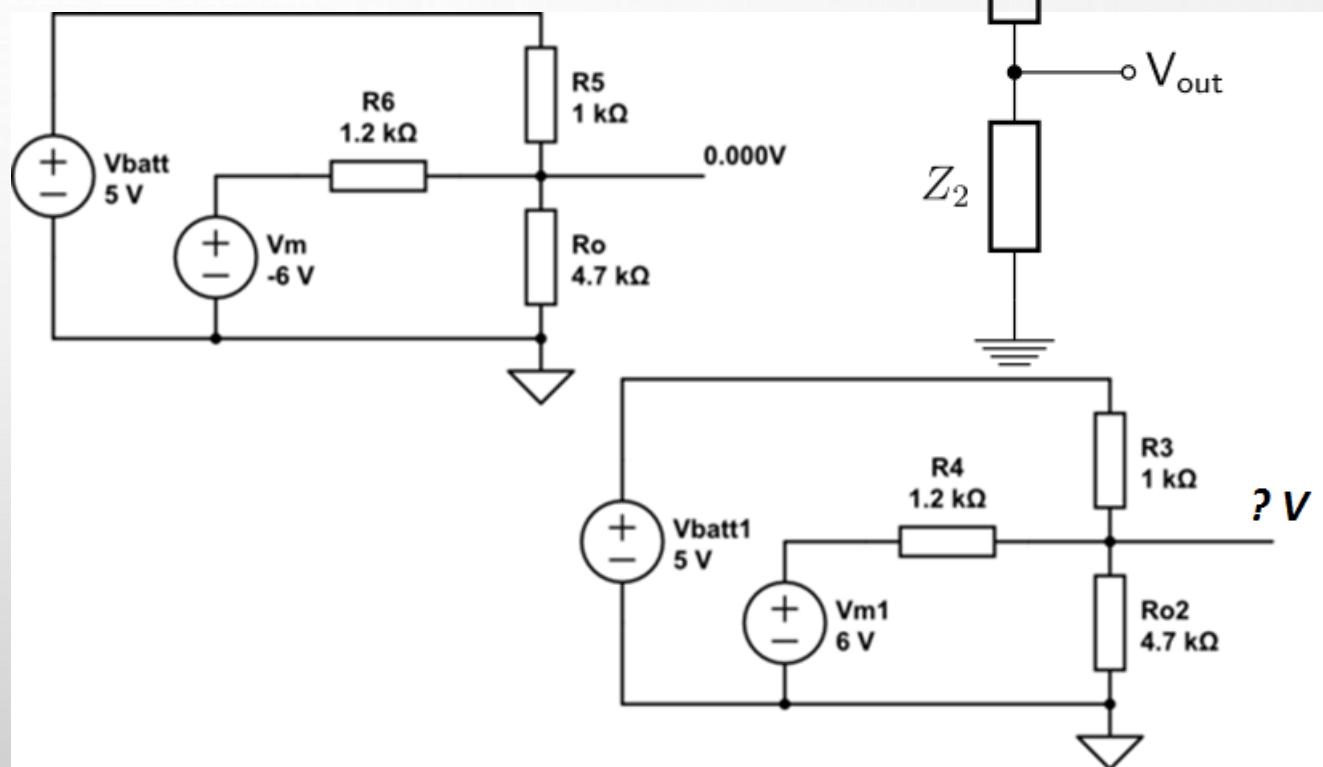
- HUMIDITY SENSOR *Tateto fe maya hayedek 5v, Tateto fe 7aga 3azla, haydek 0 w hakaza baa*
- TEMPERATURE SENSOR (LM35)



skiped

OUT OF RANGE VOLTAGE

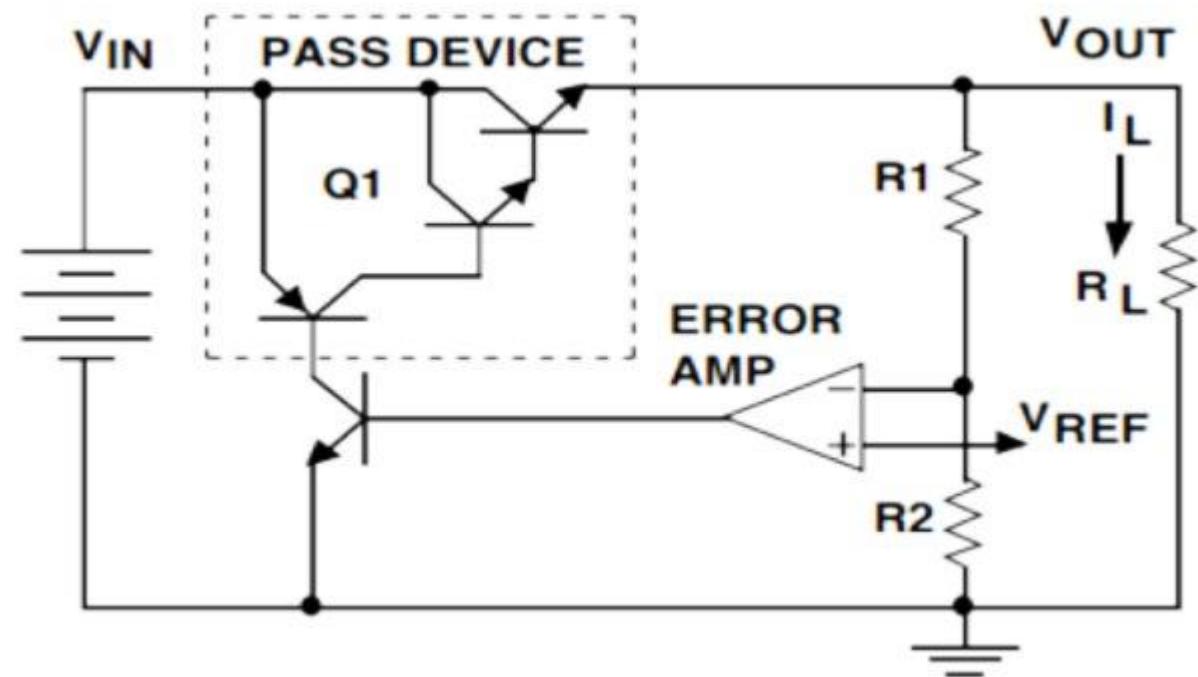
- **+6 TO -6 INPUT -> 0 TO 5 OUTPUT**
- RESISTORS DROP VOLTAGE. WOULD IT NOT BE A SIMPLE FIX TO JUST USE **RESISTORS** TO DROP THE VOLTAGE ACCORDING TO OHMS LAW? BUT THEN, RESISTORS DROP VOLTAGE DEPENDING ON THE CURRENT FLOWING THROUGH THEM. THE MOMENT YOUR COMPONENT STARTS DRAWING LESS CURRENT, THE VOLTAGE SHOOTS UP AND KILLS IT.
- IS VOLTAGE DIVIDERS SAFE ?



skipped

VOLTAGE REGULATOR

- 78XX SERIES



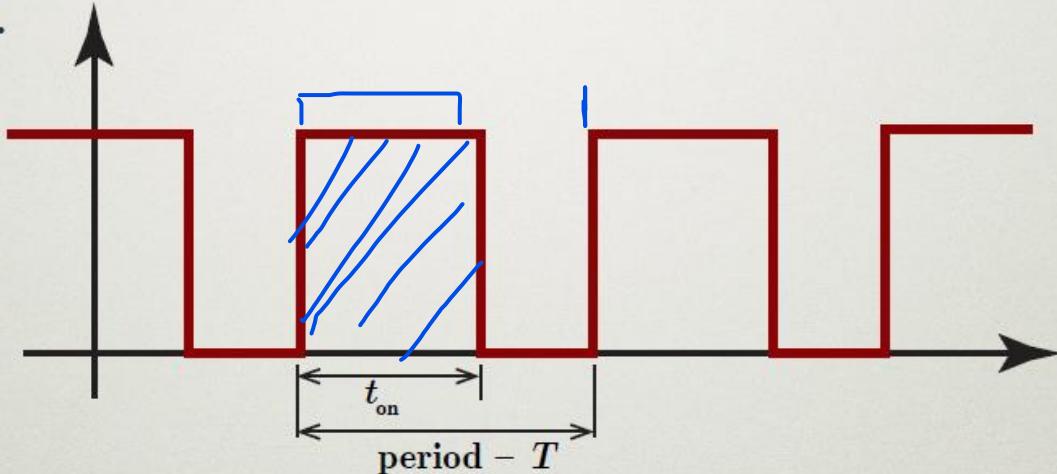
ANALOGUE OUTPUT EMULATION

- PWM 500hz



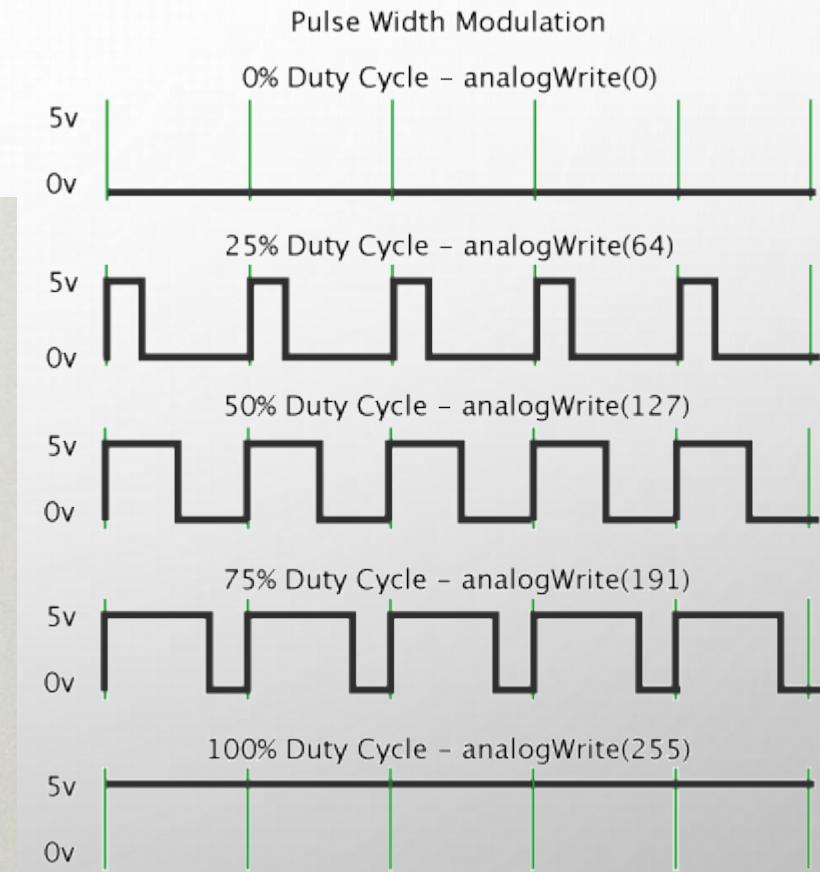
- `analogWrite(pin, dutyCycle)` command code

- PWM is defined in terms of its period and its duty cycle.



Duty Cycle: $\Delta = 100 \times \frac{t_{on}}{T} (\%)$

kam % hya shaghala mn el w2t operation

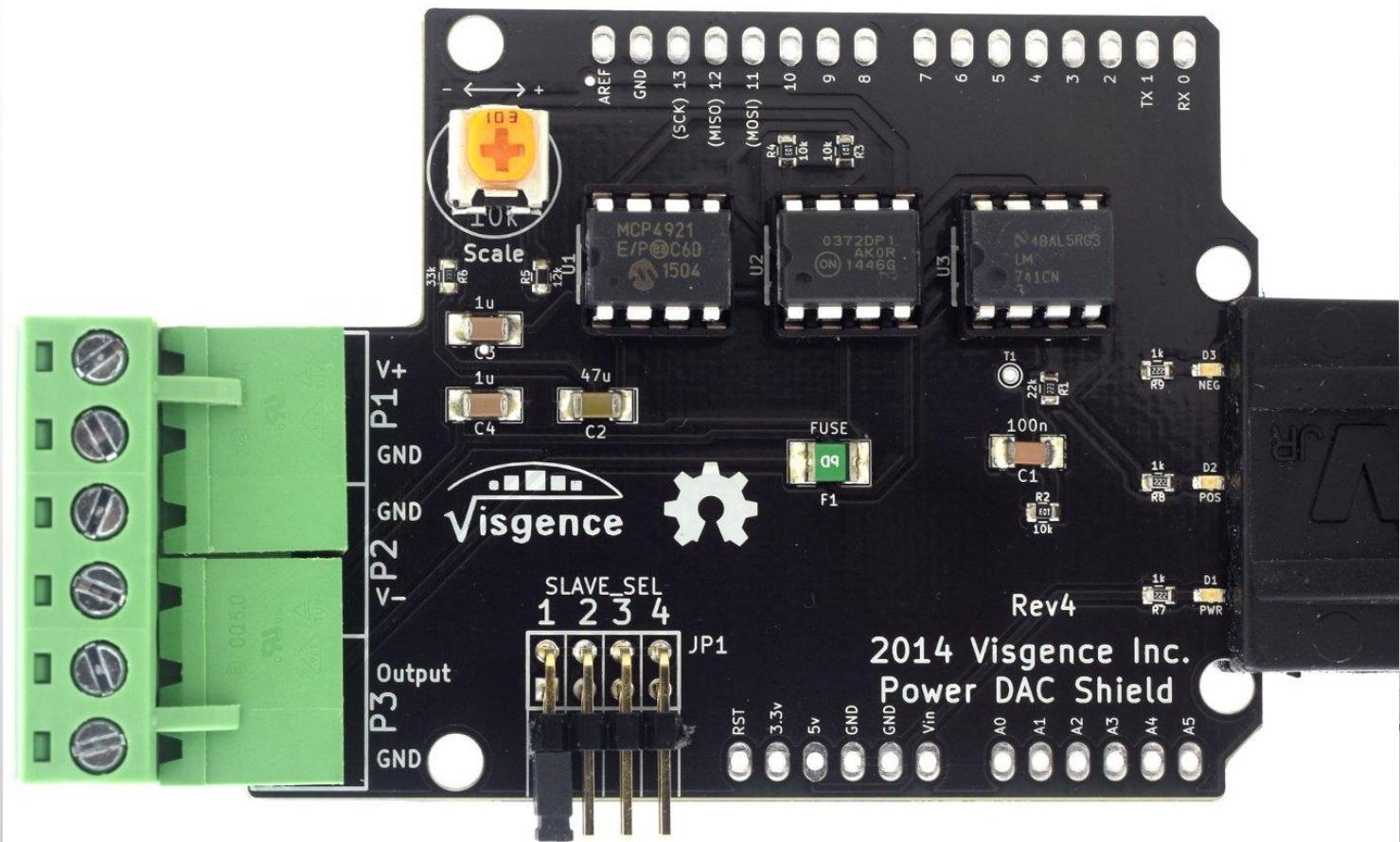


laka el lambda btr34. el lambda btfy bl kaml awl btnwr bl kamel
lakn enta 3enak msh btshof da, bsbb eno by7sl bsor3a -> fa da m3na integerated as.

this is used to have DAC

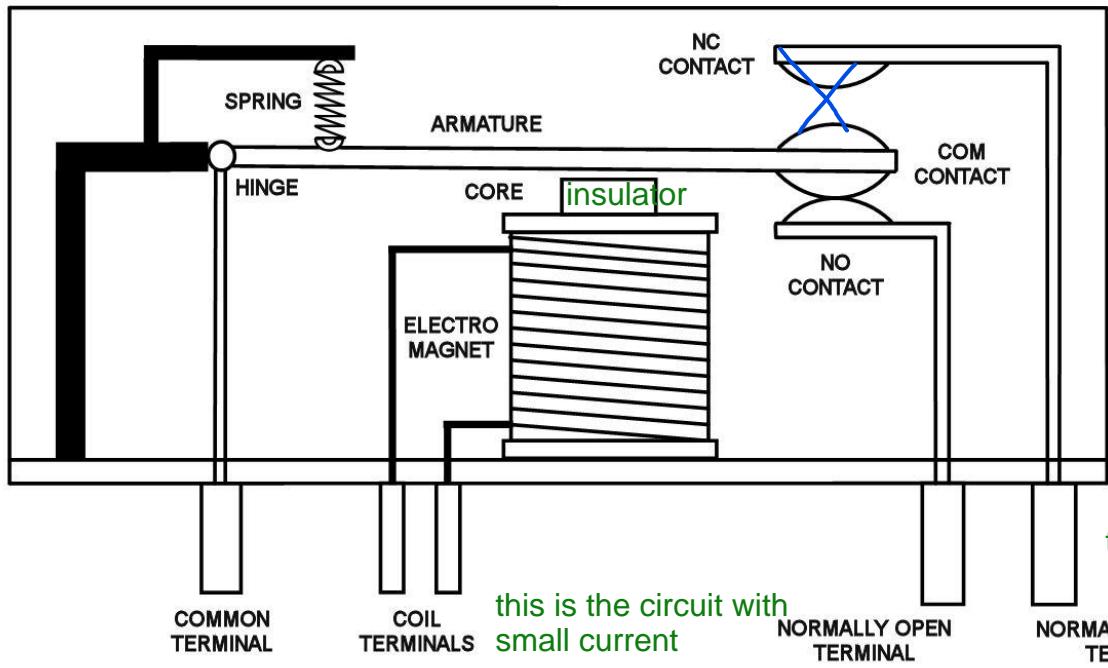
DIGITAL TO ANALOGUE SHIELDS

- TRUE AC WAVE FORM
- BIPOLE POWER SUPPLY
- SPI OR I2C
- DC MOTORS



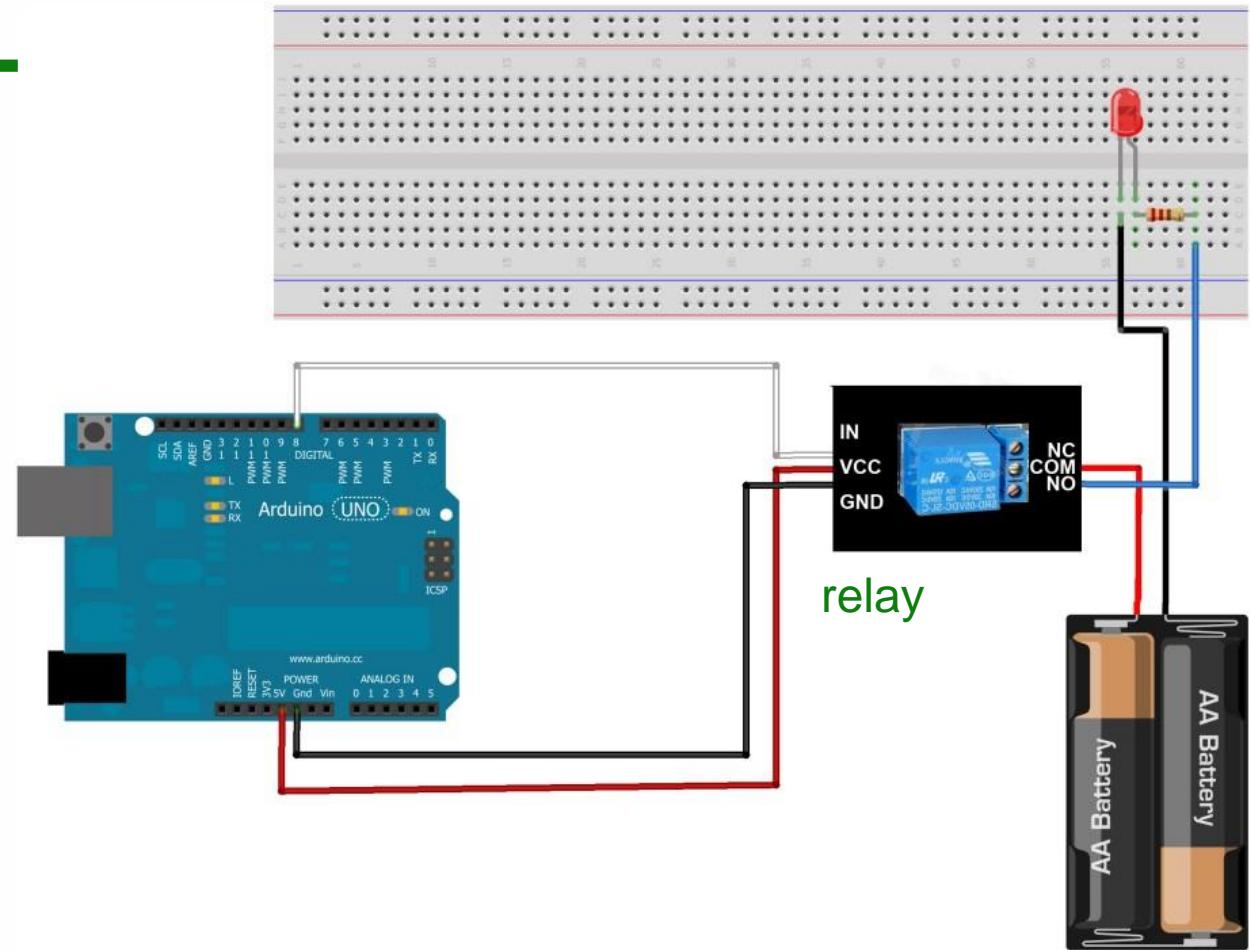
CIRCUIT

by3ml magnetic field, by7awl



this is the circuit with large current

el fekra kolaha en lama el low current by3dy, el wire by3ml act akno
magnet, fa yshed el NC Contact tlz2 fl Com contact, fa keda b3ml connect
lel circut, fa sa3tha lw 3ndk out battery tebd2 tmshy el current baa.
Made with Fritzing.org

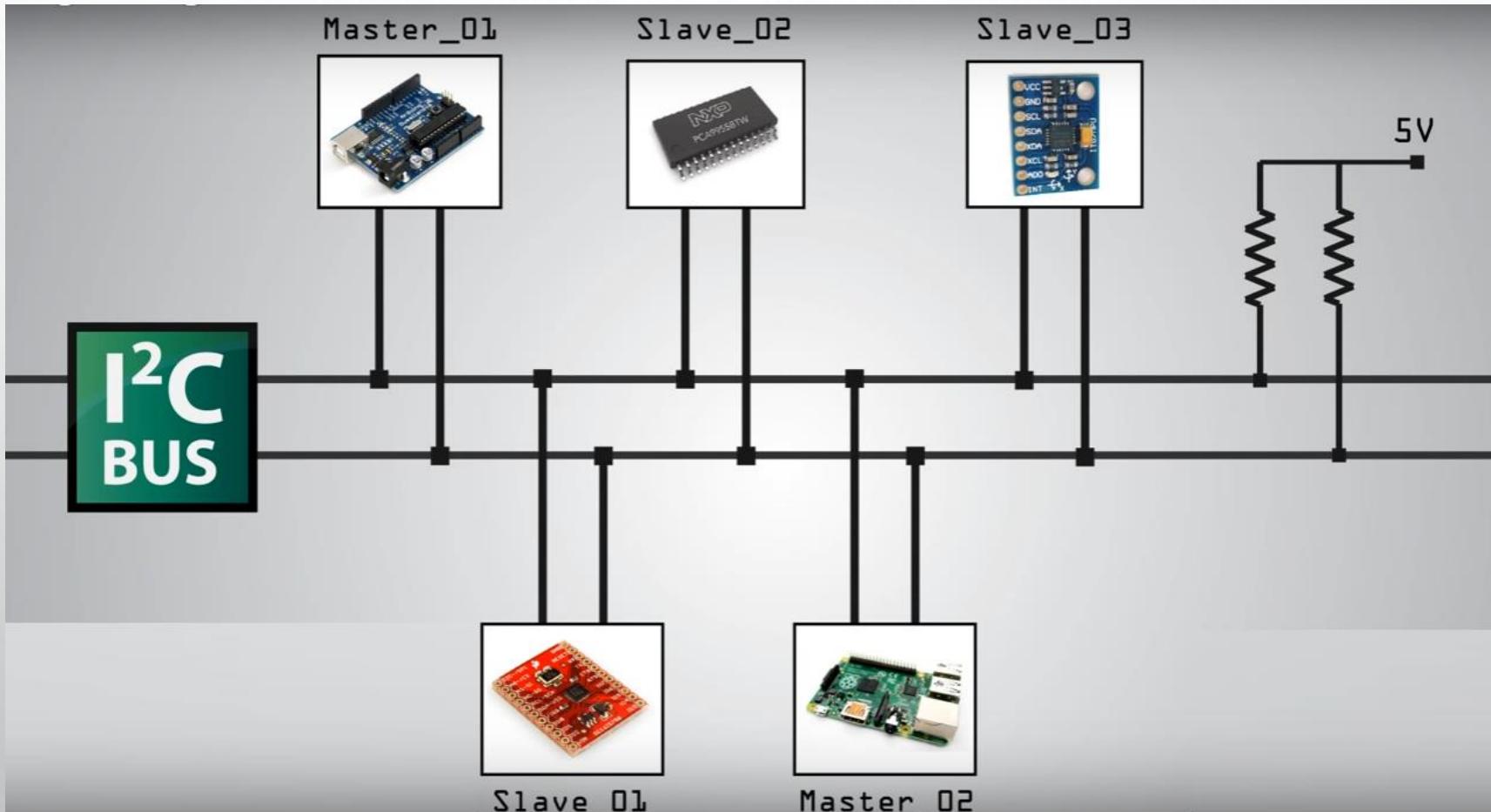


CONNECTING MICROCONTROLLERS TO OUTSIDE WORLD

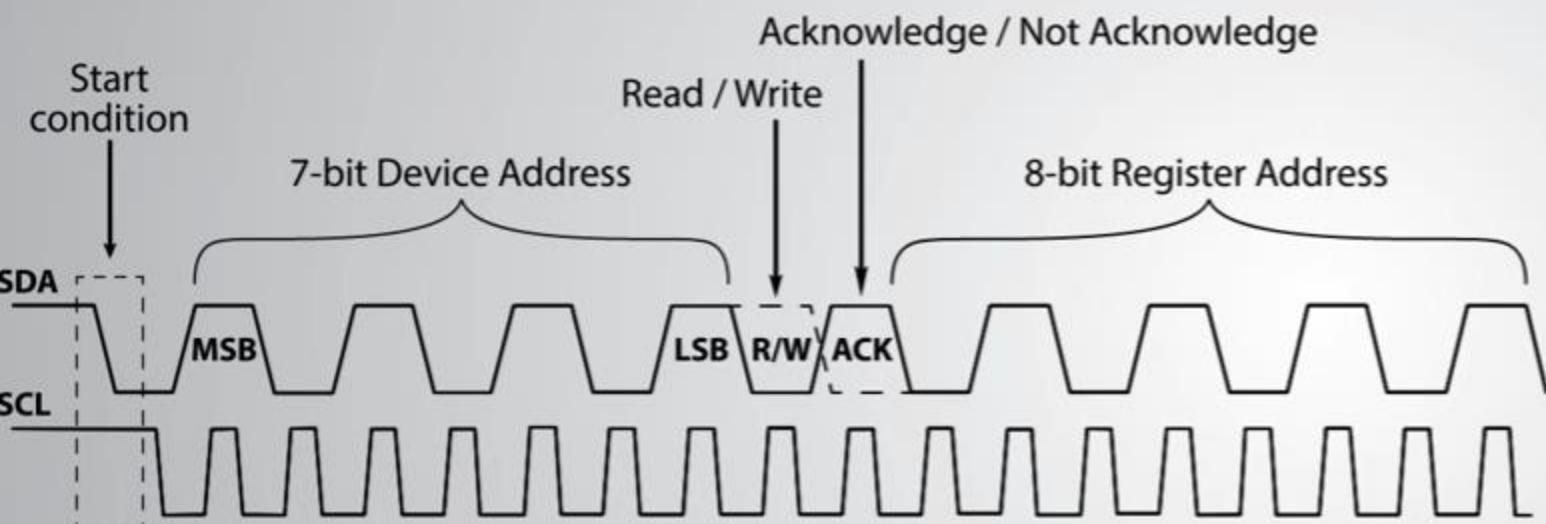
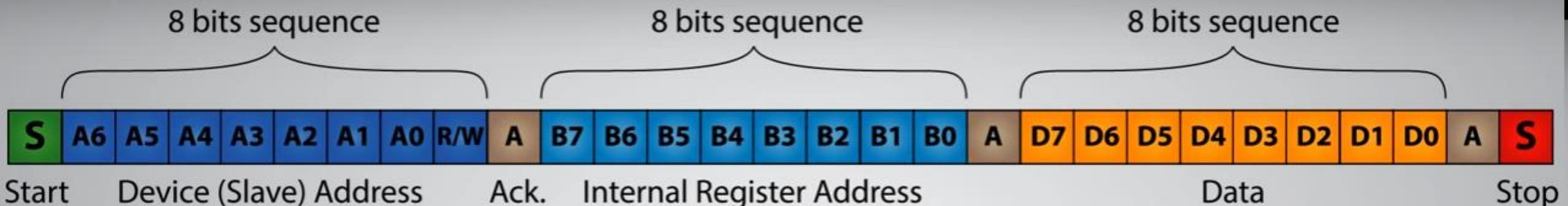
COMPLEX INTERFACES

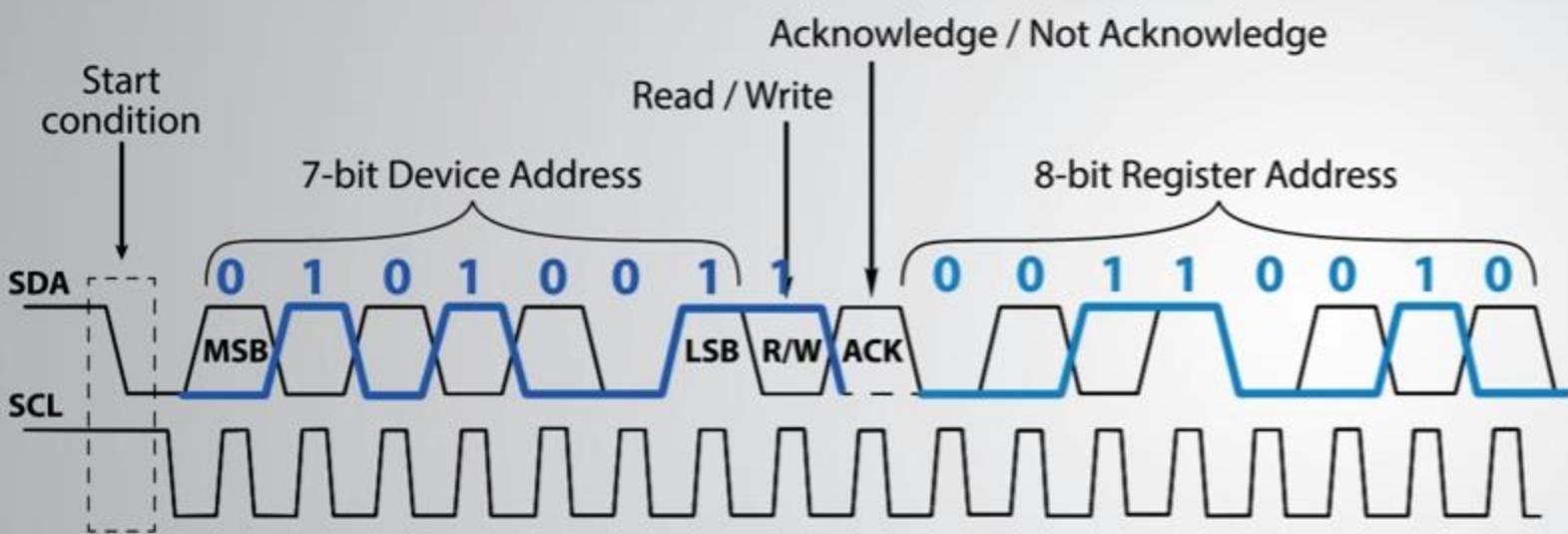
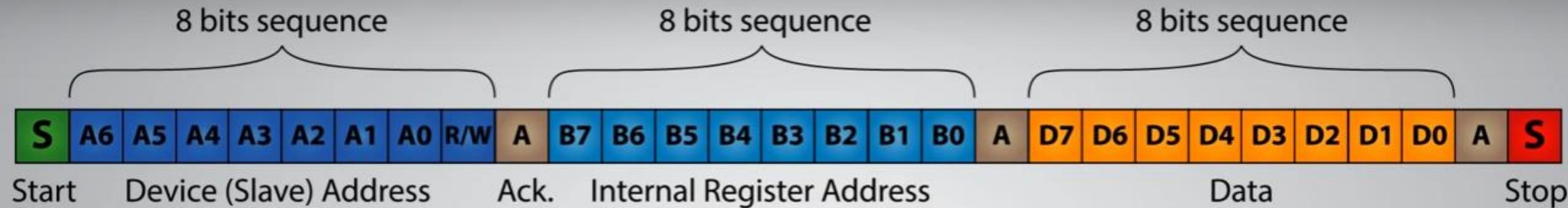
will be explained later

I²C



I2C PROTOCOL

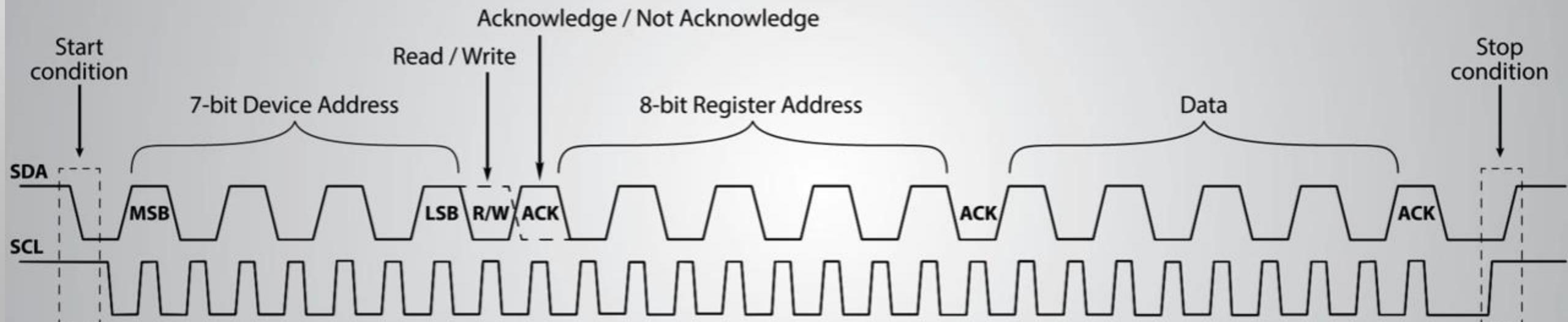
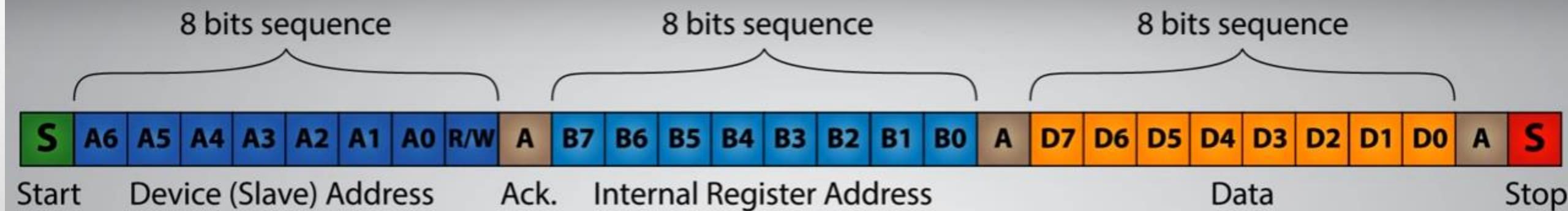




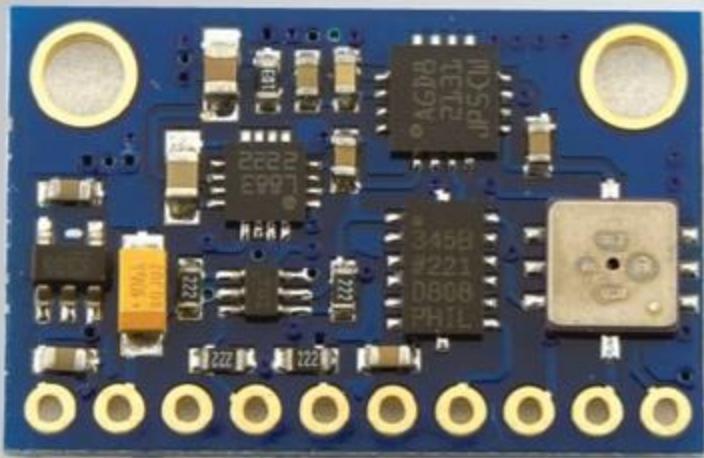
For example: ADXL345 Accelerometer

Device Address: _____ 0x53 or 0101 0011 (read mode, 8th bit high)
 Internal Register Address
 for the X Axis: _____ 0x32 or 0011 0010

I2C



GY - 80

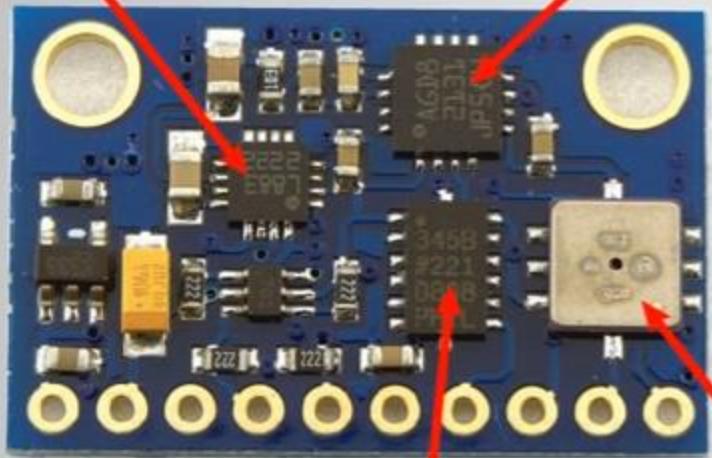


GY - 521



MC5883L
Magnetometer
Address: 0x1E

GY - 80



L3G4200D
Gyroscope
Address: 0x69

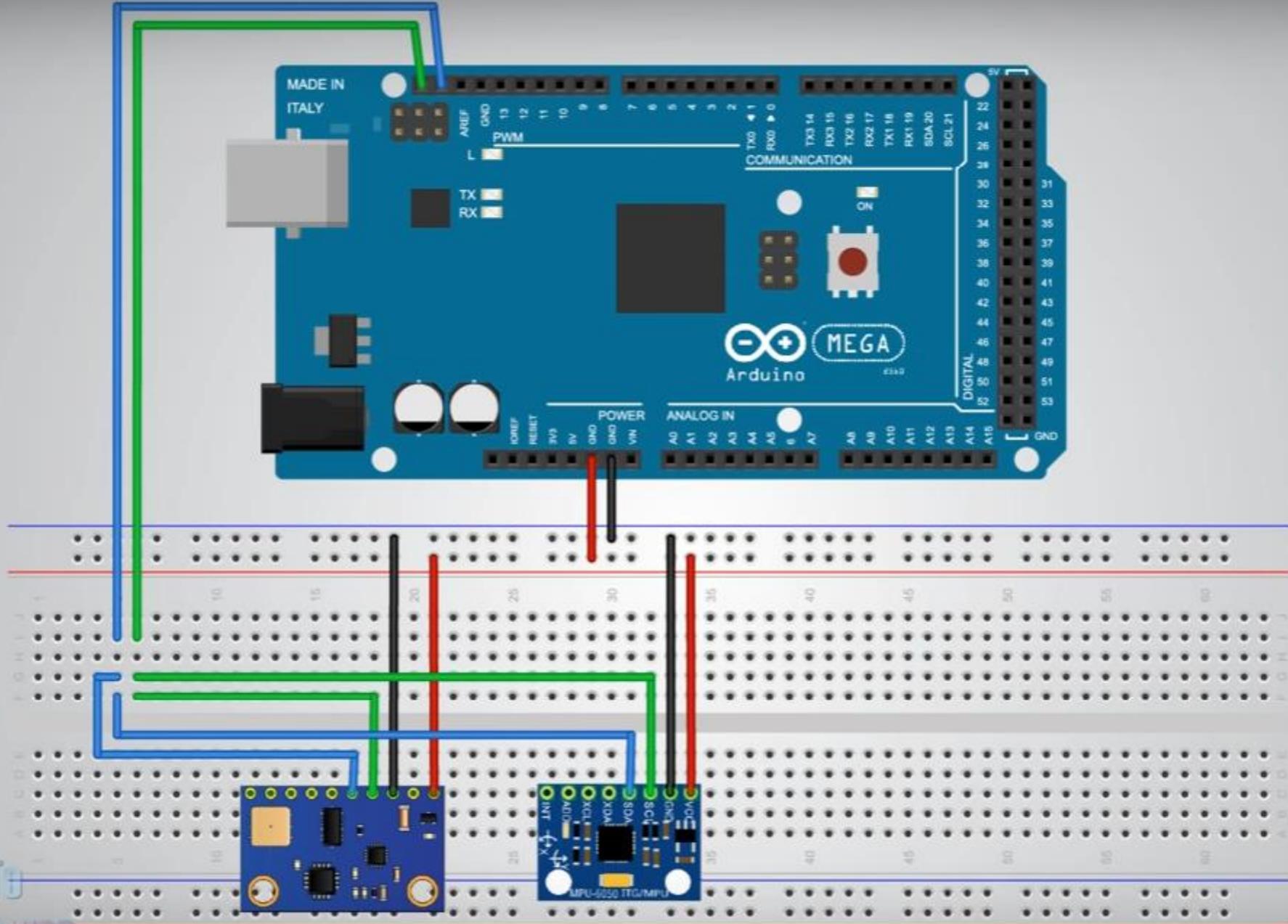
BMP085
Barometer + Thermometer
Address: 0x77

ADXL345
Accelerometer
Address: 0x53

GY - 521

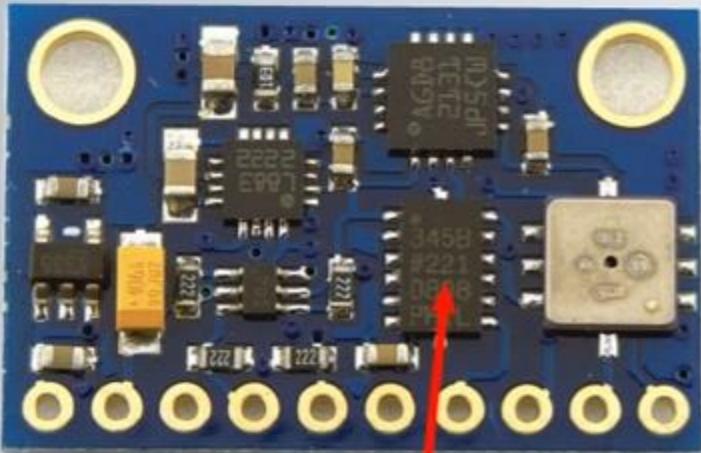


Note: These addresses are for Read Mode, included logic high at R/W bit.



For Example:

Read data from the X Axis
of the ADXL345 Accelerometer



ADXL345
Accelerometer
Address: 0x53

Data Sheet

ADXL345

REGISTER MAP

Table 19.

Address		Name	Type	Reset Value	Description
Hex	Dec				
0x00	0	DEVID	R	11100101	Device ID
0x01 to 0x1C	1 to 28	Reserved			Reserved; do not access
0x1D	29	THRESH_TAP	R/W	00000000	Tap threshold
0x1E	30	OFSX	R/W	00000000	X-axis offset
0x1F	31	OFSY	R/W	00000000	Y-axis offset
0x20	32	OFSZ	R/W	00000000	Z-axis offset
0x21	33	DUR	R/W	00000000	Tap duration
0x22	34	Latent	R/W	00000000	Tap latency
0x23	35	Window	R/W	00000000	Tap window
0x24	36	THRESH_ACT	R/W	00000000	Activity threshold
0x25	37	THRESH_INACT	R/W	00000000	Inactivity threshold
0x26	38	TIME_INACT	R/W	00000000	Inactivity time
0x27	39	ACT_INACT_CTL	R/W	00000000	Axis enable control for activity and inactivity detection
0x28	40	THRESH_FF	R/W	00000000	Free-fall threshold
0x29	41	TIME_FF	R/W	00000000	Free-fall time
0x2A	42	TAP_AXES	R/W	00000000	Axis control for single tap/double tap
0x2B	43	ACT_TAP_STATUS	R	00000000	Source of single tap/double tap
0x2C	44	BW_RATE	R/W	00001010	Data rate and power mode control
0x2D	45	POWER_CTL	R/W	00000000	Power-saving features control
0x2E	46	INT_ENABLE	R/W	00000000	Interrupt enable control
0x2F	47	INT_MAP	R/W	00000000	Interrupt mapping control
0x30	48	INT_SOURCE	R	00000010	Source of interrupts
0x31	49	DATA_FORMAT	R/W	00000000	Data format control
0x32	50	DATAX0	R	00000000	X-Axis Data 0
0x33	51	DATAX1	R	00000000	X-Axis Data 1
0x34	52	DATAY0	R	00000000	Y-Axis Data 0
0x35	53	DATAY1	R	00000000	Y-Axis Data 1
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1
0x38	56	FIFO_CTL	R/W	00000000	FIFO control
0x39	57	FIFO_STATUS	R	00000000	FIFO status

*Figure from the datasheet of the ADXL345 Accelerometer

SUBSCRIBE
NOW!

CONNECTION CONSIDERATIONS

- SENSORS

- VOLTAGE INPUT CONDITIONING +VE 0 TO 5 IS OK
- WHAT IF SENSING IS REQUIRED IN –VE RANGE OR 10 V

apply some function to transform this range into 0-5 range

- POWER

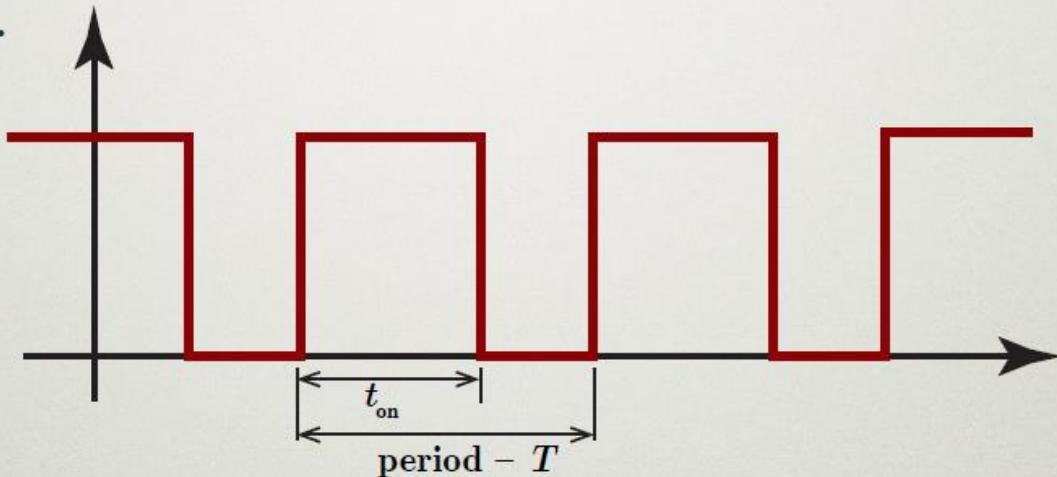
- DIGITAL OUTPUT IS OK .
- WHAT IF ANALOG OUTPUT REQUIRED ? we don't have DAC -> so either DAC shield or PWM
- HIGHER DRIVE CURRENT OR AC ?

repeated

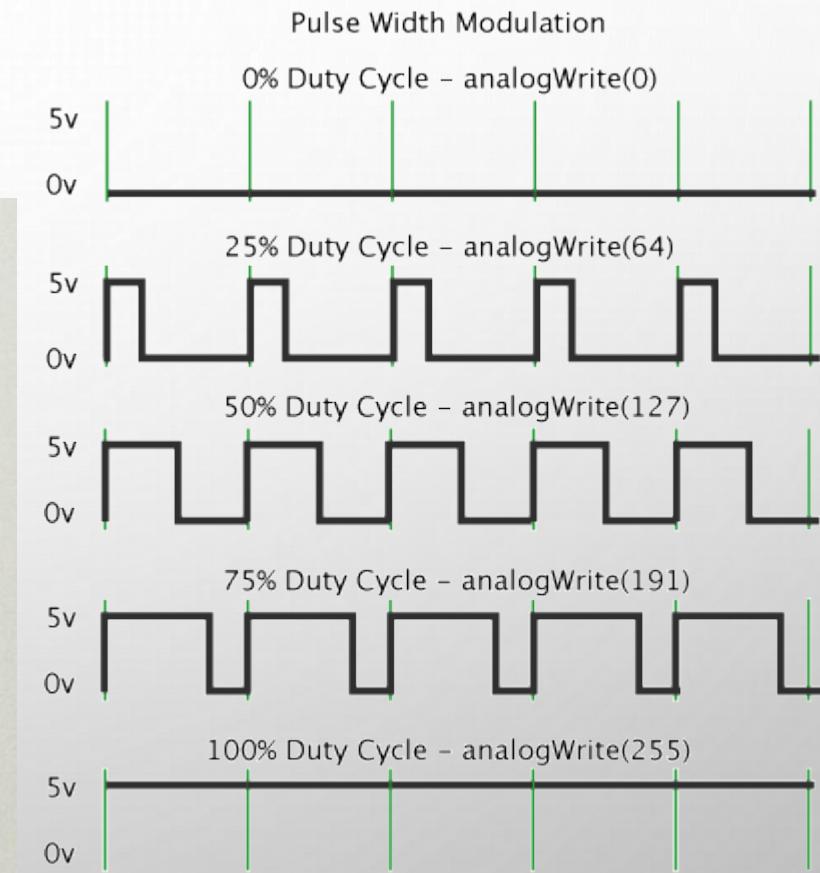
CONSIDERATIONS ON ARDUINO

- PWM 500hz
 - `analogWrite(pin, dutyCycle)`

- PWM is defined in terms of its period and its duty cycle.



$$\text{Duty Cycle: } \Delta = 100 \times \frac{t_{on}}{T} (\%)$$





PWM BIT TAGGING

how you can manually implement PWM

```
• VOID SETUP()
{
    PINMODE(13, OUTPUT); declare the pin on which you want to be used as output
}

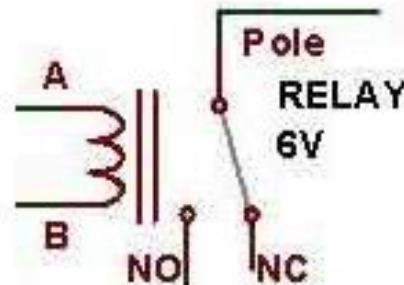
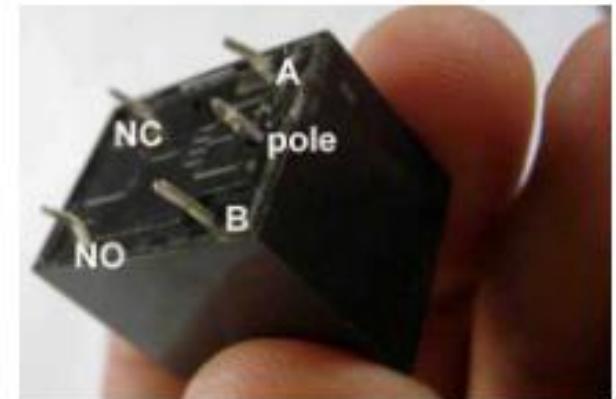
VOID LOOP()
{
    DIGITALWRITE(13, HIGH);
    DELAYMICROSECONDS(100); // APPROXIMATELY 10% DUTY CYCLE @ 1KHZ
    DIGITALWRITE(13, LOW);
    DELAYMICROSECONDS(1000 - 100);
}
```

this is how we manually implement PWM with 10%

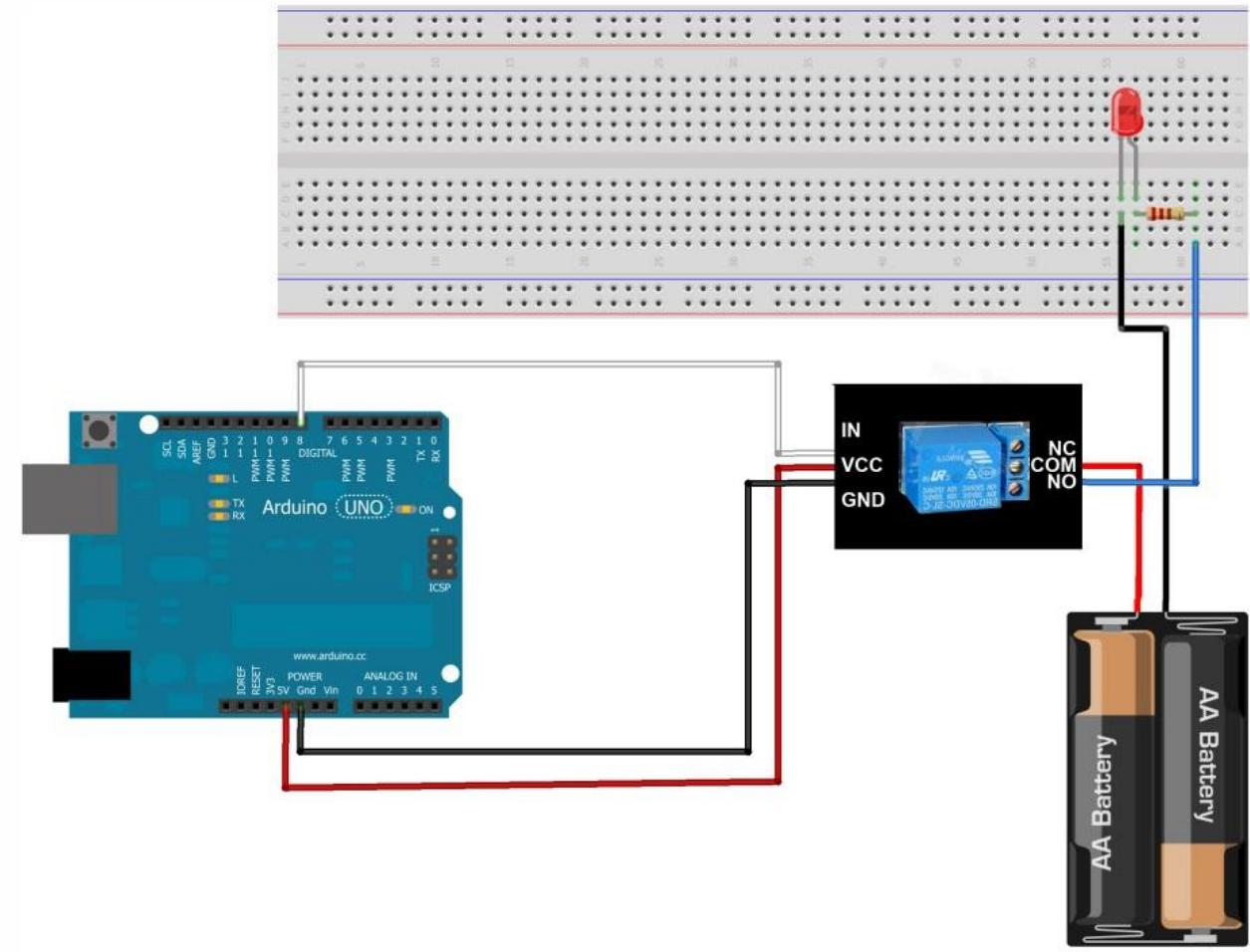
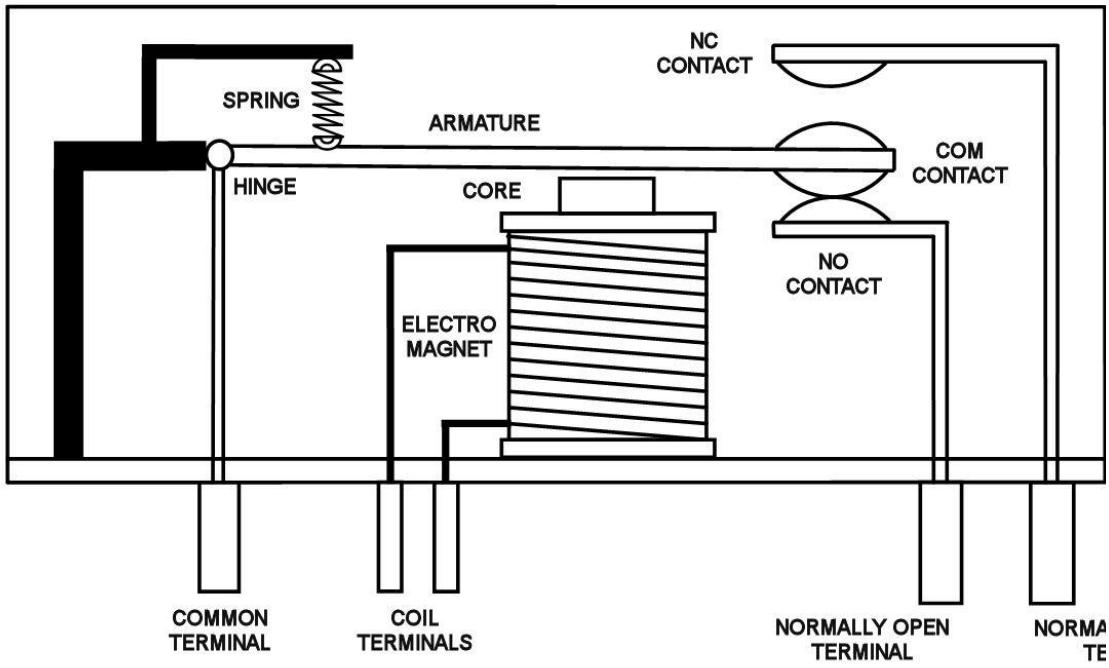
USING RELAYS

- [HTTP://WWW.BUILDCIRCUIT.COM/HOW-TO-USE-A-RELAY/](http://WWW.BUILDCIRCUIT.COM/HOW-TO-USE-A-RELAY/)

A relay shown in the picture is an electromagnetic or mechanical relay.

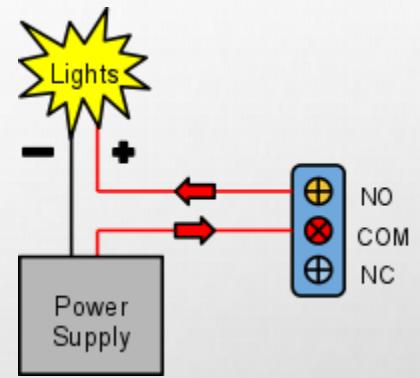
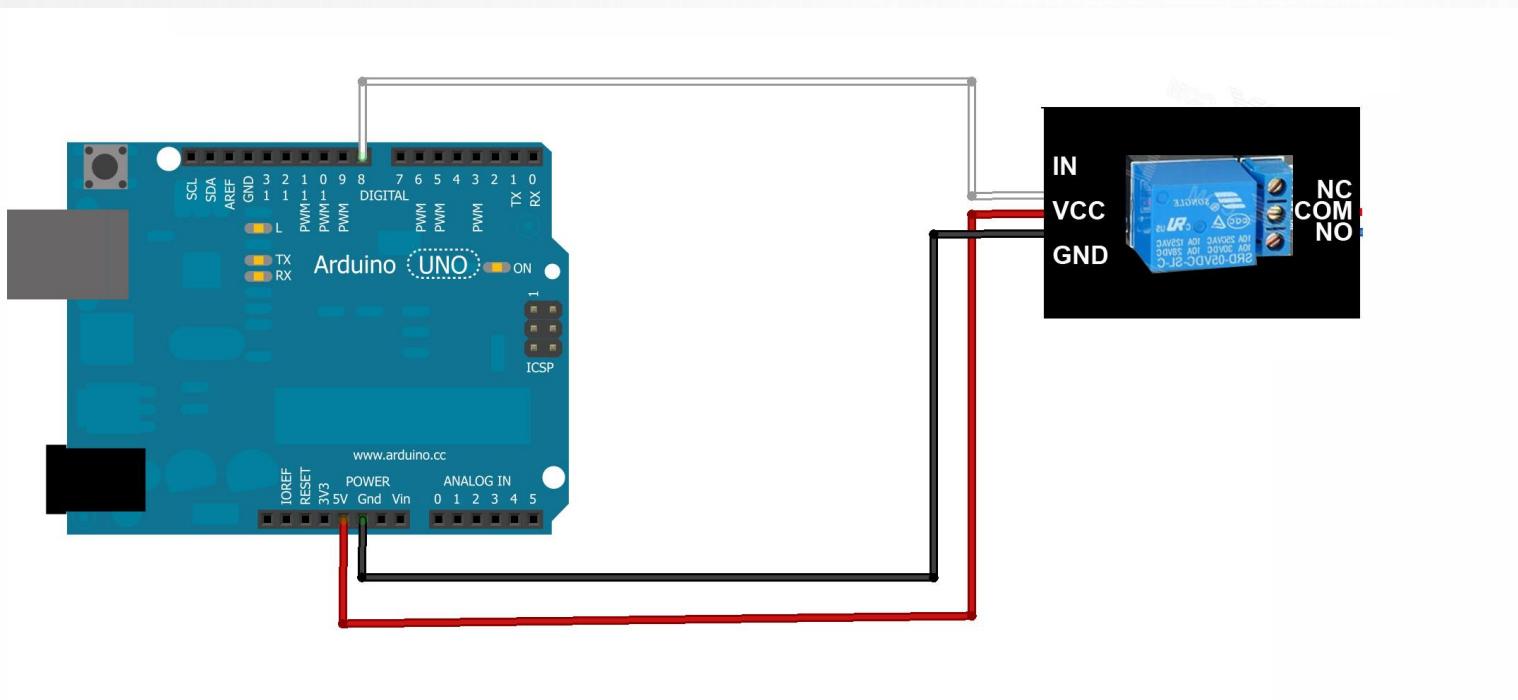


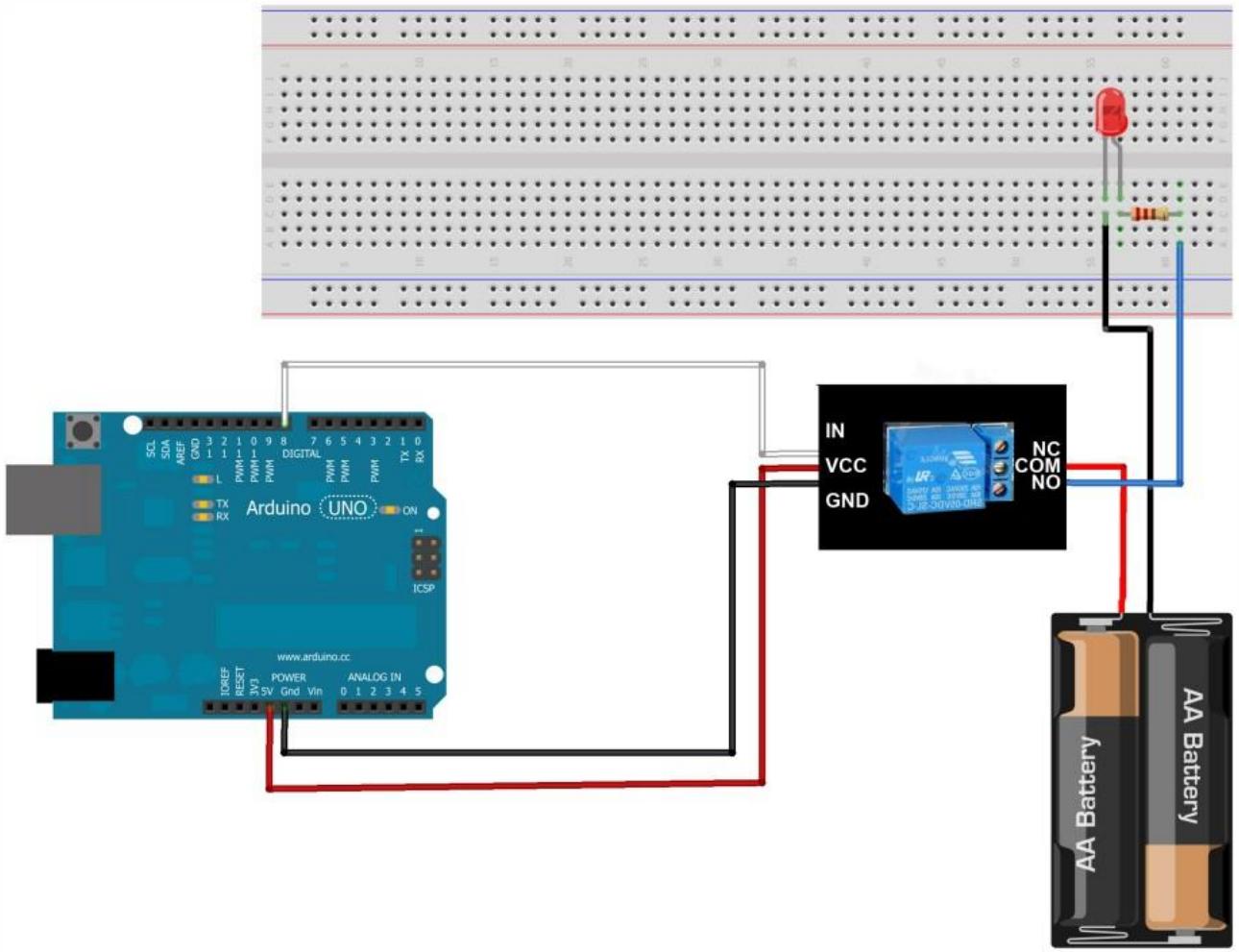
CIRCUIT CONTROL WITH RELAY



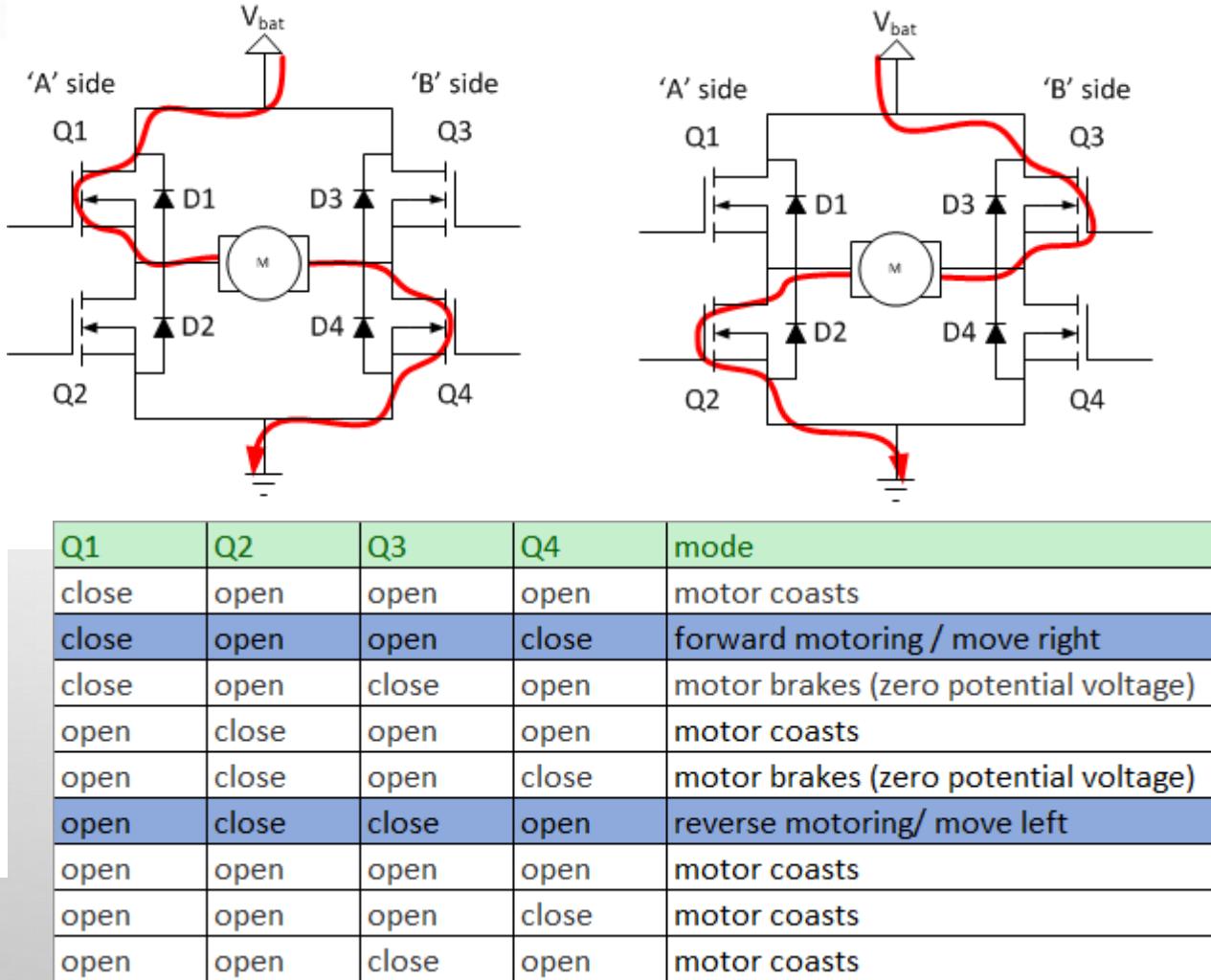
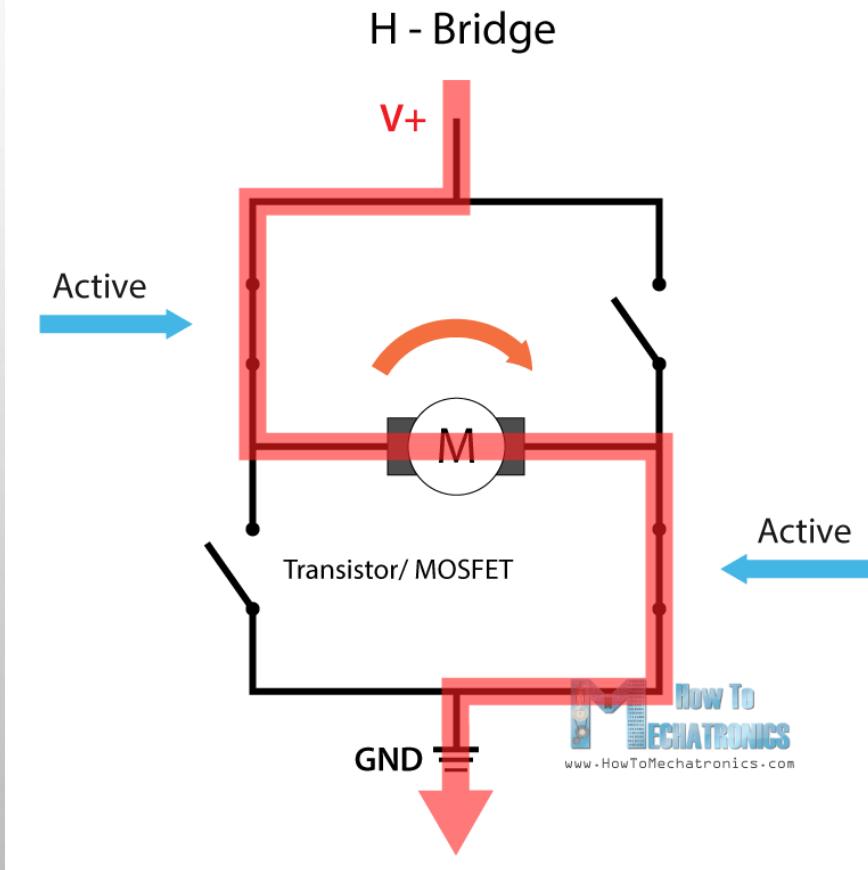
- [HTTPS://WWW.YOUTUBE.COM/WATCH?V=NYJHYKEOOGI](https://www.youtube.com/watch?v=NYJHYKEOOGI)

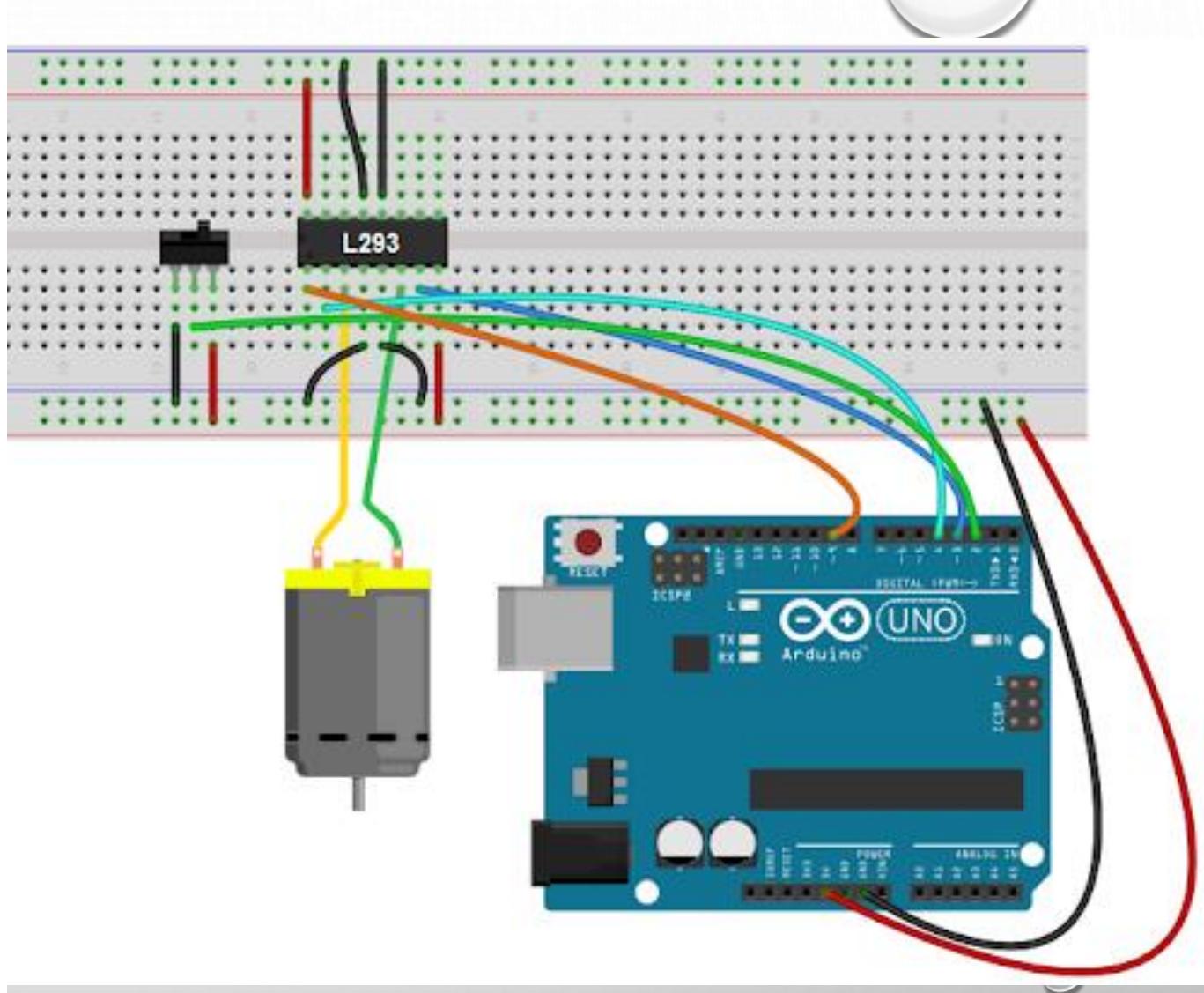
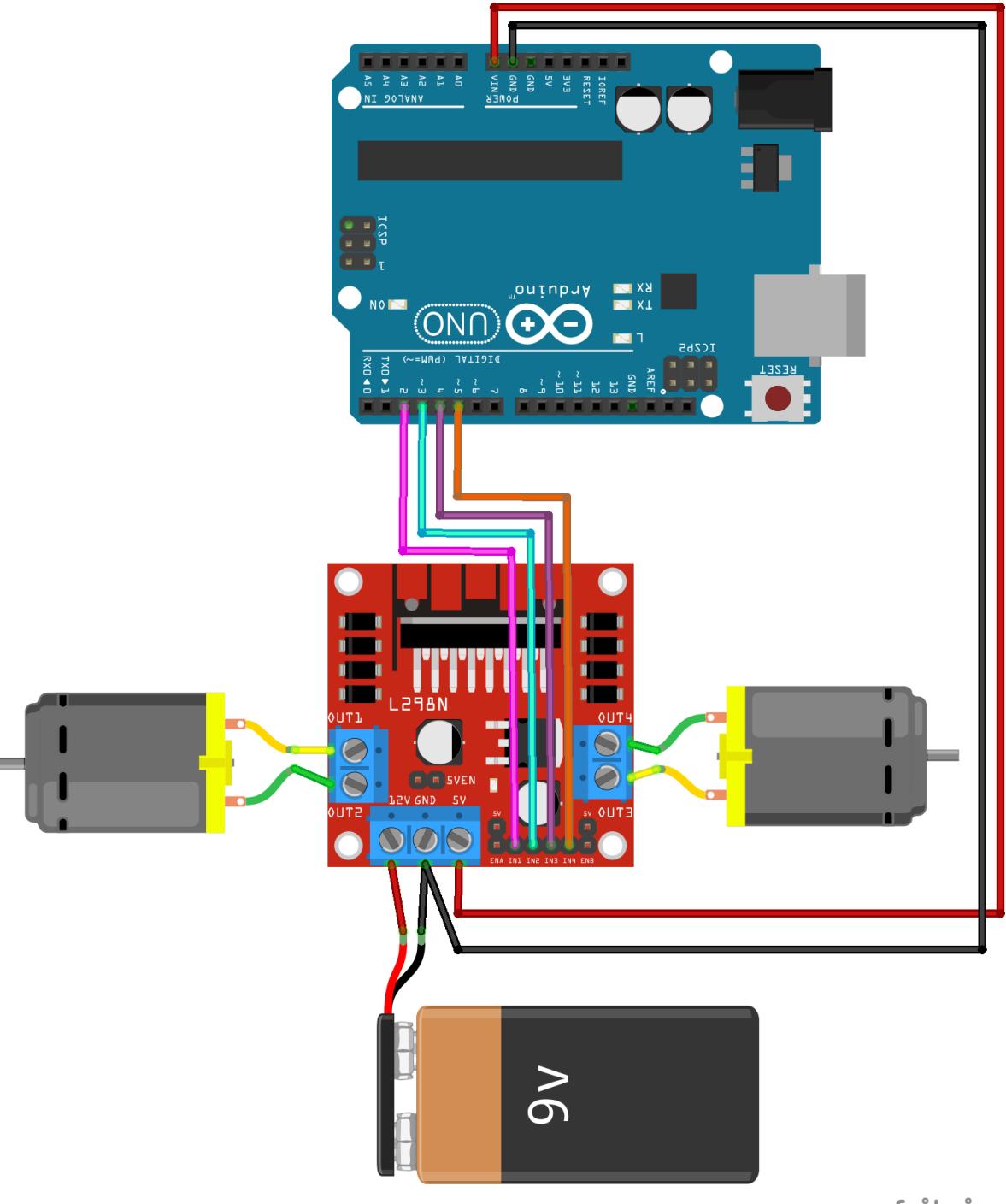
USING RELAYS





LOAD POLARITY SWITCH





[HTTPS://WWW.YOUTUBE.COM/WATCH?V=I7IFSQ4TQU8](https://www.youtube.com/watch?v=I7IFSQ4TQU8)

EXPANDING I/O CAPABILITIES

this is a shift register

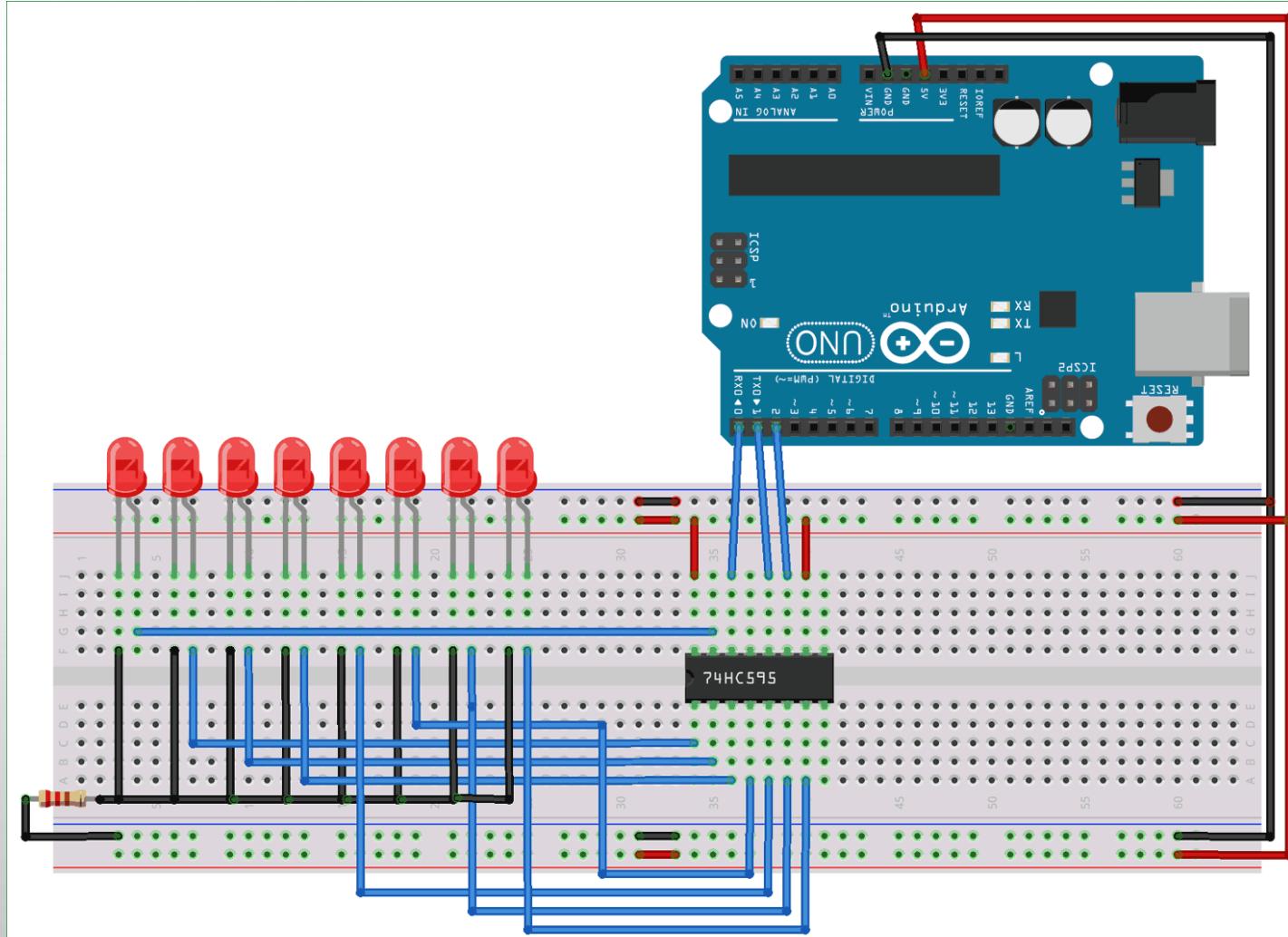
74HC595 SIFO
serial in parallel output
btst5dm 3 pins w ttl3 8 output

8 BIT SERIAL IN

PARALLEL OUT

SHIFT REGISTER

btd5lo el values, w hwa bytl3hom kolohom mara wahda, b7es enk t2dr
el 8 bits m3 b3d





MULTIPLEXER DEMULTIPLEXER

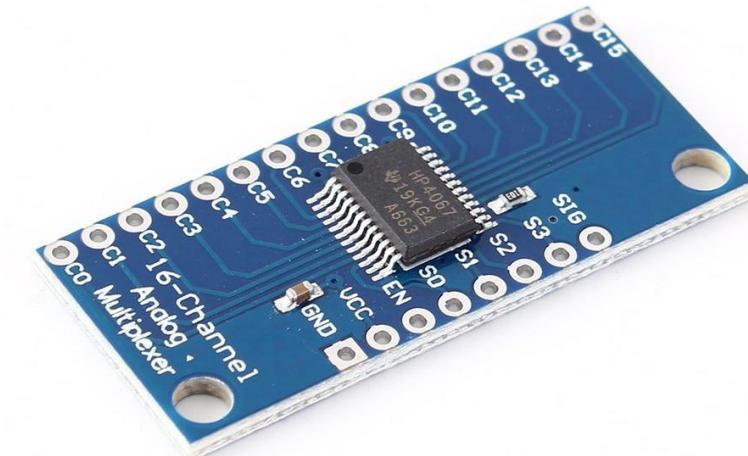
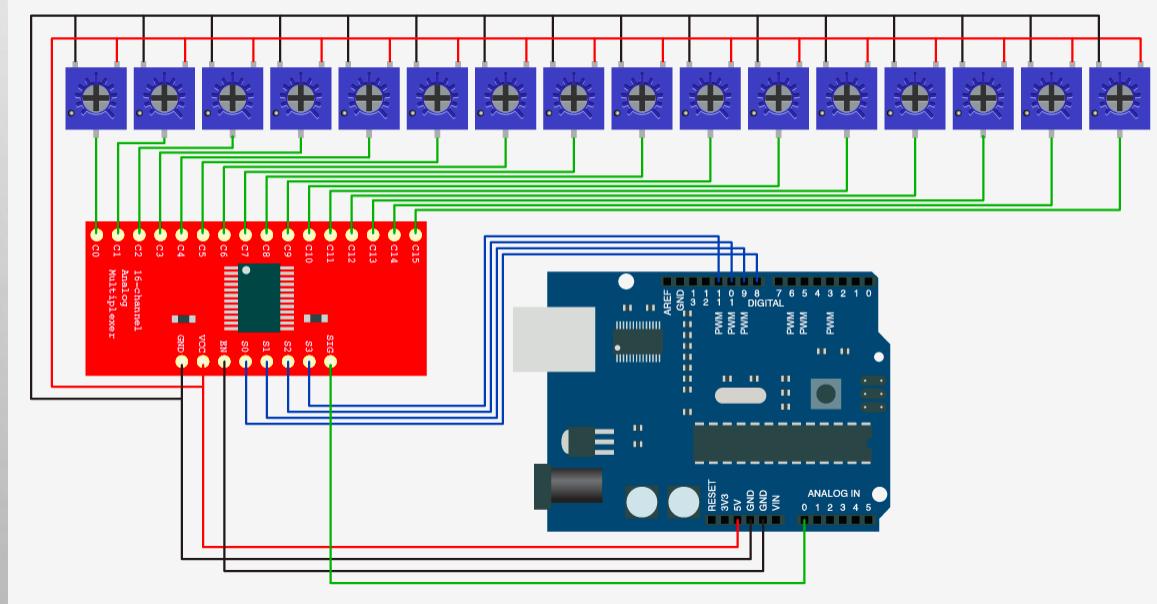
;w 3ndk multiple inputs w 3auz tkhlhom 3la 3dd a2al, fa bnst5dm decodeing

- MAIN DIFFERENCE FORM SHIFT REGISTERS ?

msh lazm yakhud el data bl trreb
A multiplexer switches between inputs and is normally used to sample multiple inputs in a sequence which can generally be arbitrary.

A shift register takes a parallel word and shifts it out as a sequence of bits.

The multiplexer is more versatile than the shift register and therefore usually more expensive.



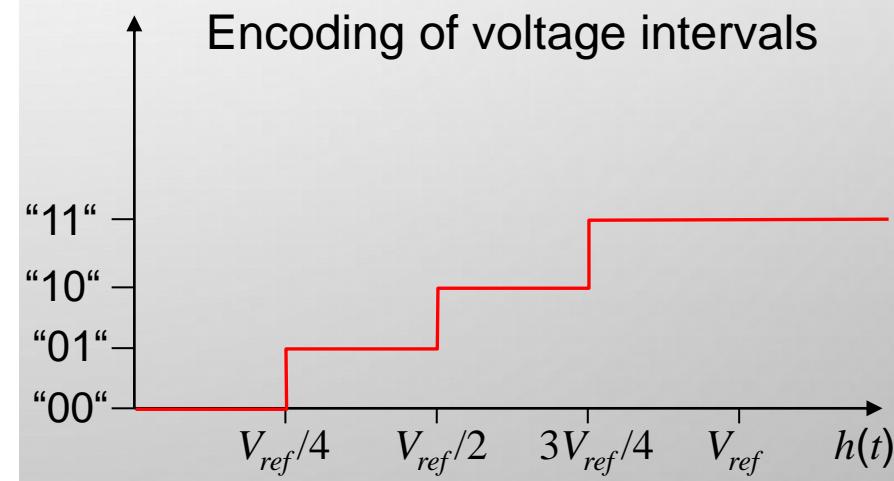
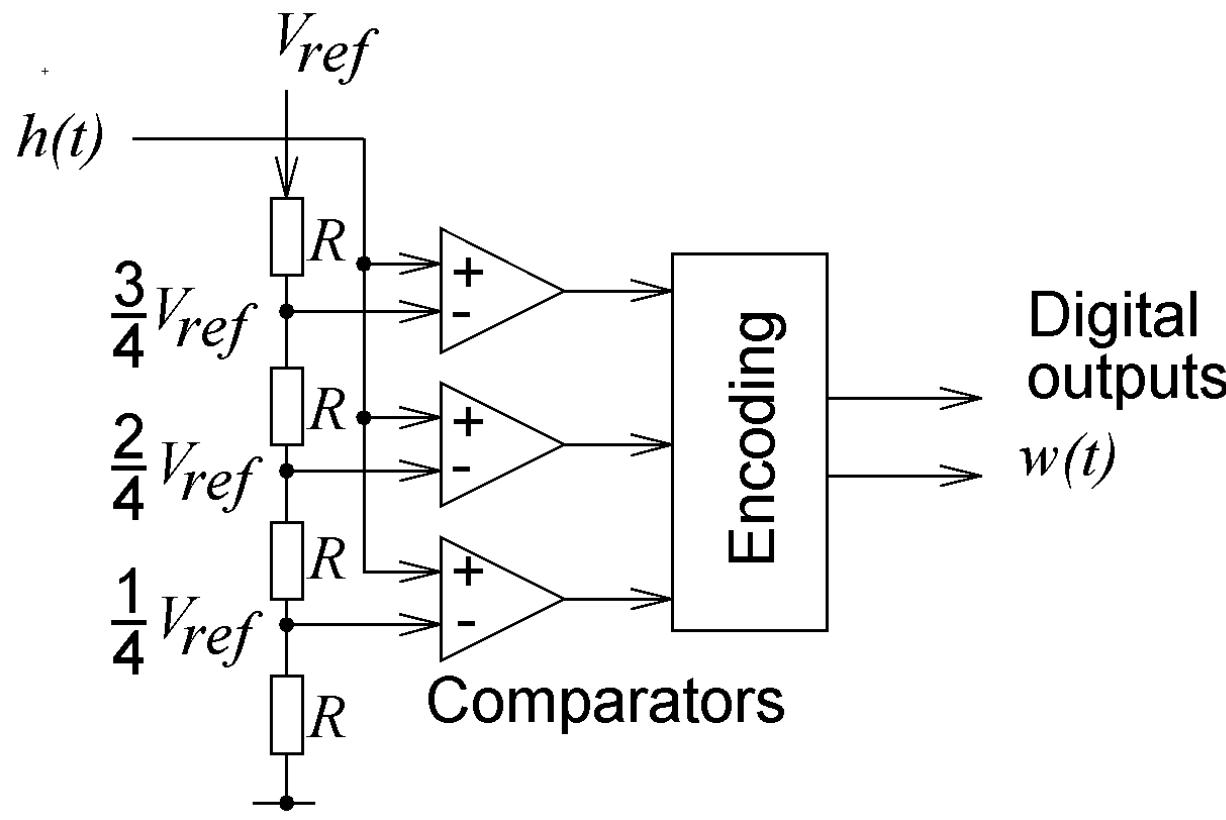
ADC / DAC

skipped



FLASH A/D CONVERTER

No decoding of $h(t) > V_{ref}$



RESOLUTION

- RESOLUTION (IN BITS): NUMBER OF BITS PRODUCED
- RESOLUTION Q (IN VOLTS): DIFFERENCE BETWEEN TWO INPUT VOLTAGES CAUSING THE OUTPUT TO BE INCREMENTED BY 1

$$Q = \frac{V_{FSR}}{n} \quad \text{with}$$

Q : resolution in volts per step

V_{FSR} : difference between largest and smallest voltage

n : number of voltage intervals

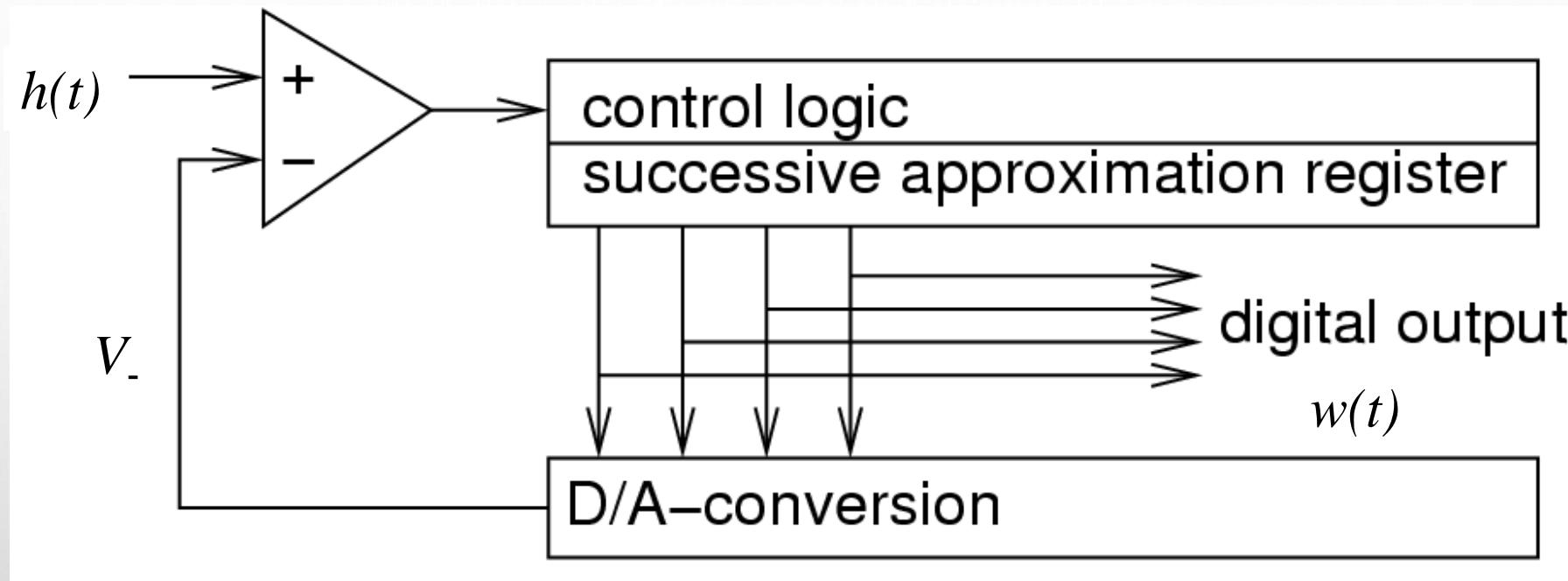
Example:

$Q = V_{ref}/4$ for the previous slide

RESOLUTION AND SPEED OF FLASH A/D- CONVERTER

- **PARALLEL COMPARISON WITH REFERENCE VOLTAGE**
- **SPEED:** $O(1)$
- **HARDWARE COMPLEXITY:** $O(N)$
- **APPLICATIONS:** E.G. IN VIDEO PROCESSING

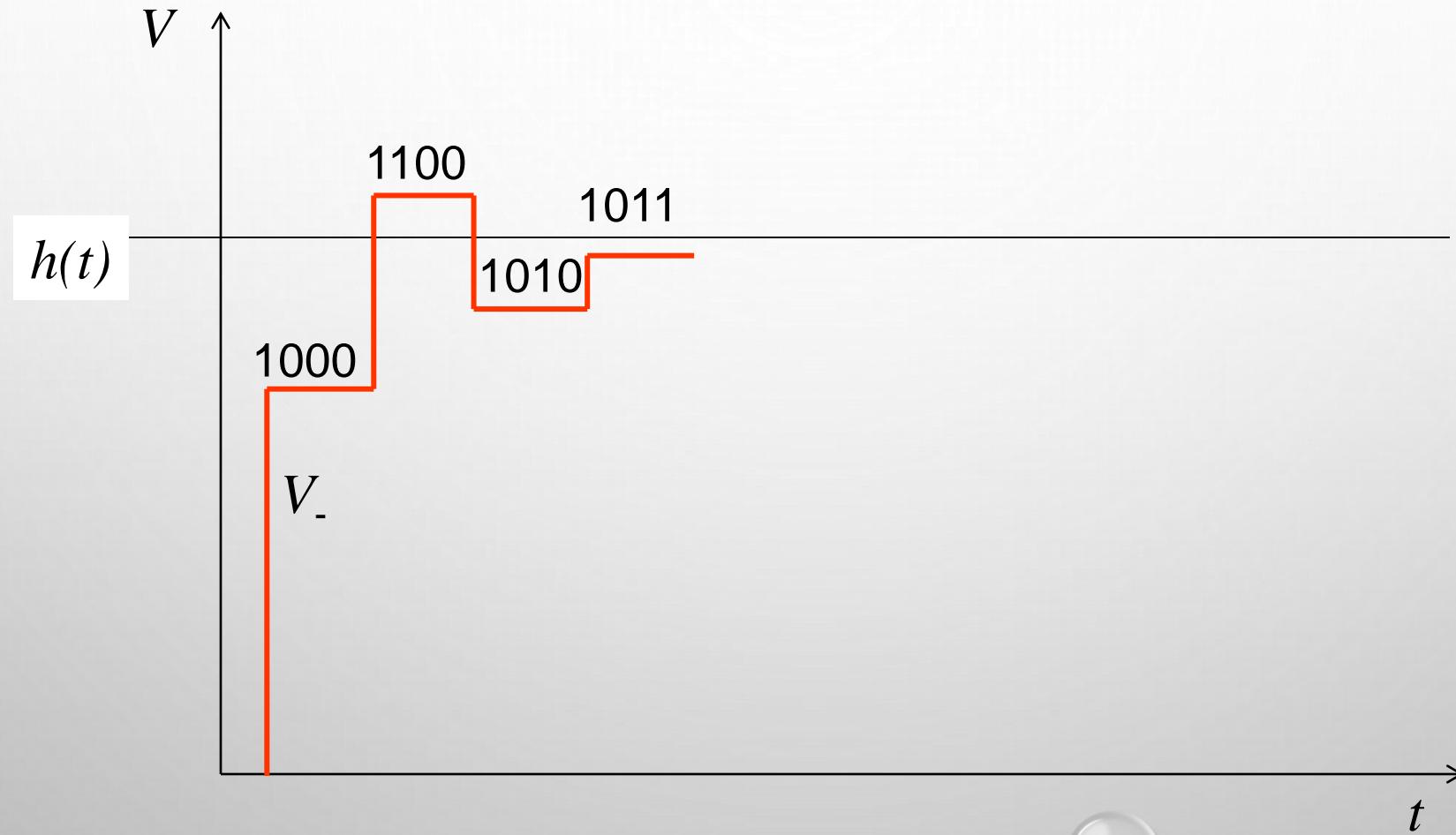
HIGHER RESOLUTION: SUCCESSIVE APPROXIMATION



- KEY IDEA: BINARY SEARCH:
- SET MSB='1'
- IF TOO LARGE: RESET MSB
- SET MSB-1='1'
- IF TOO LARGE: RESET MSB-1

Speed: $O(\log_2(n))$
Hardware complexity: $O(\log_2(n))$
with $n = \#$ of distinguished
voltage levels;
slow, but high precision possible.

SUCCESSIVE APPROXIMATION (2)



PROGRAMMING MICROCONTROLLERS

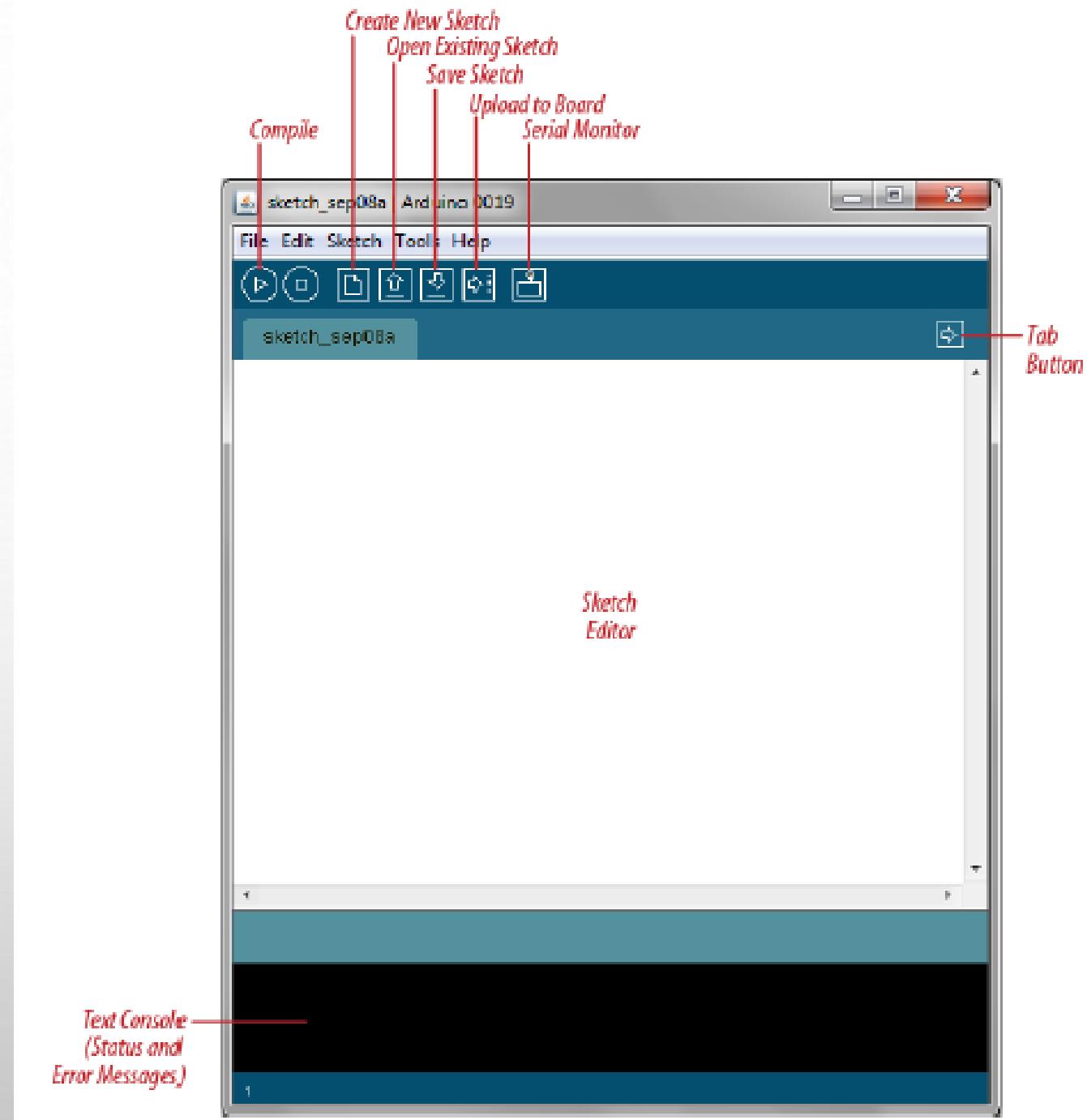
ARDUINO PLATFORM PROGRAMMING

AGENDA

- ARDUINO IDE
- MAJOR CONCEPTS
- RANGE MAPPING
- INTERRUPTS
- DELAYS
- REMEMBERING CHANGES

ARDUINO IDE

- FILE HANDLER
- CODE EDITOR
- COMPILER
- CONSOLE



BARE MIN.

The screenshot shows the Arduino IDE interface with the title bar "BareMinimum | Arduino 1.0.3". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for file operations. The main code area contains the following code:

```
void setup() {
    // put your setup code here, to run once:
}

void loop() {
    // put your main code here, to run repeatedly:
}
```

The status bar at the bottom indicates "LilyPad Arduino w/ ATmega328 on COM28".

TWO REQUIRED FUNCTIONS
METHODS / ROUTINES:

VOID **SETUP ()**

{

// RUNS ONCE

}

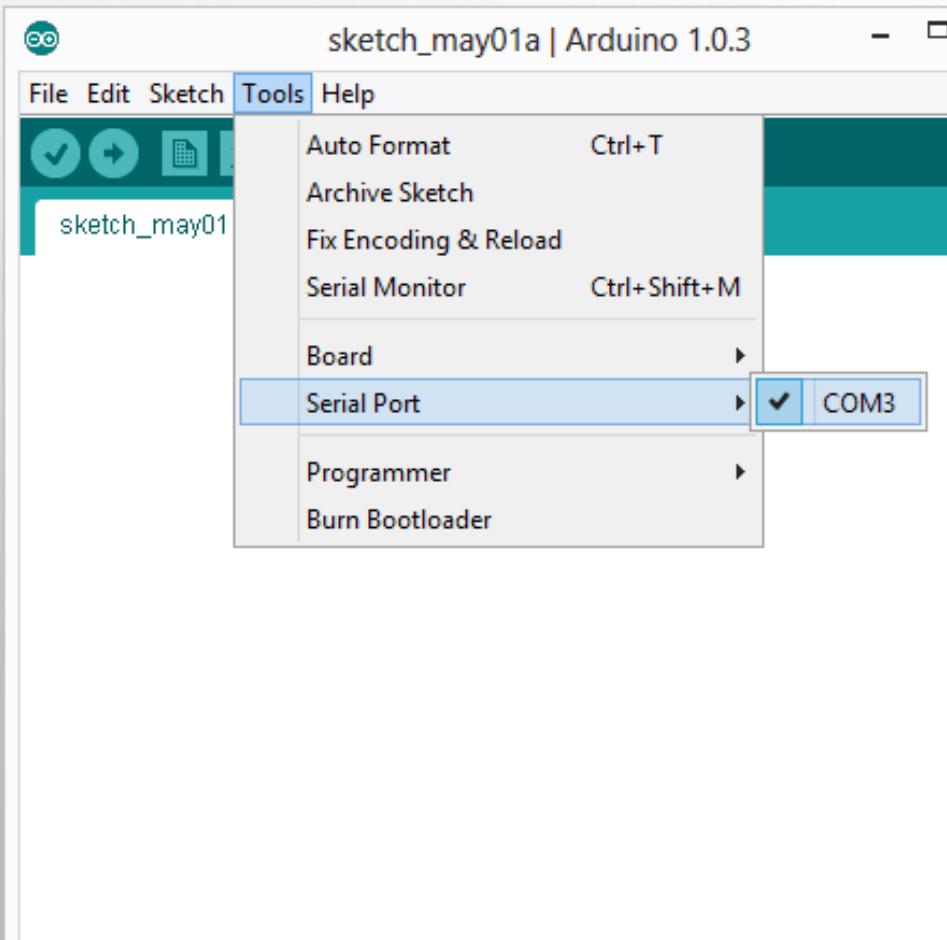
VOID **LOOP ()**

{

// REPEATS

}

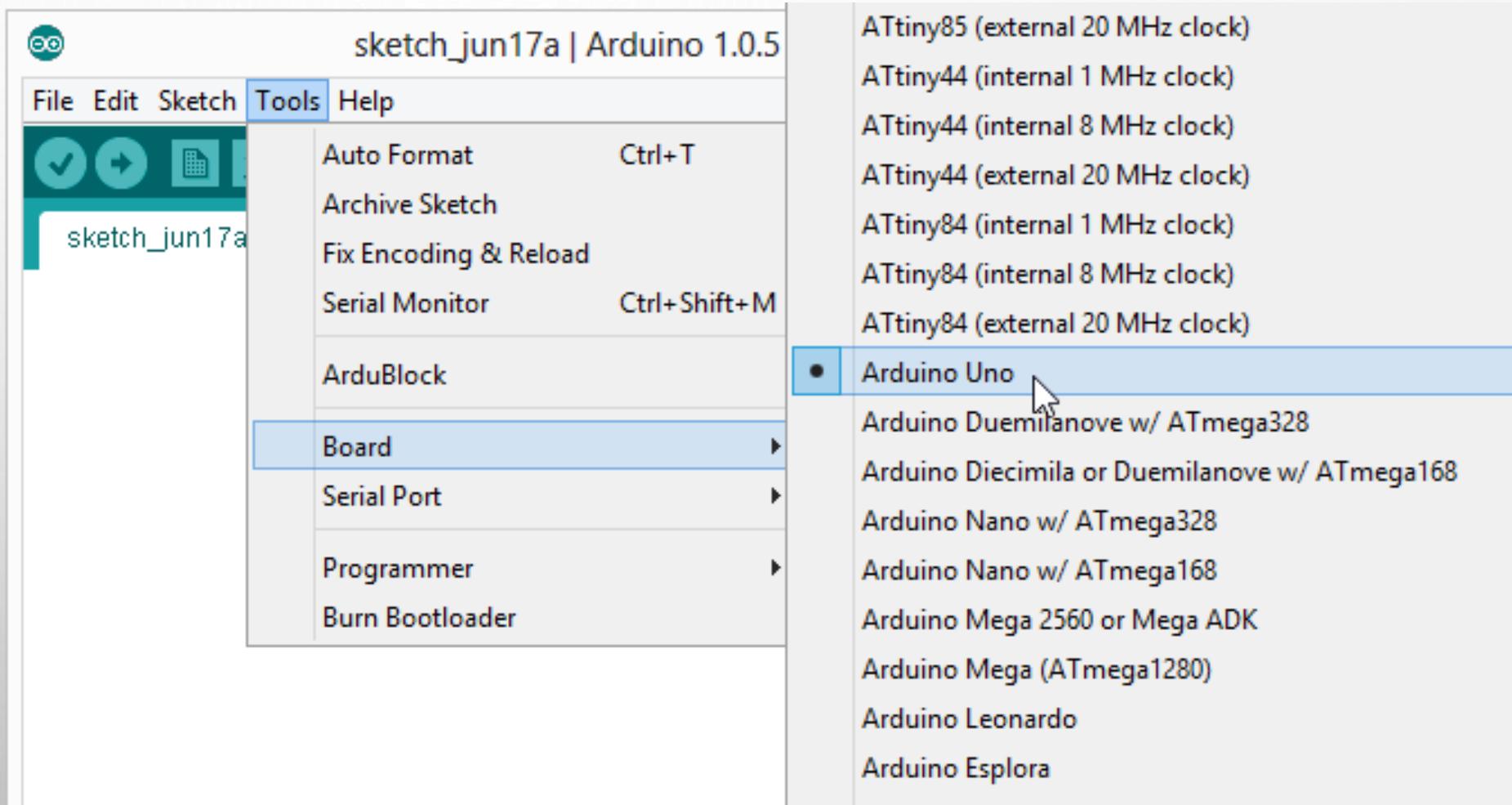
SETTINGS: TOOLS → SERIAL PORT



- YOUR COMPUTER COMMUNICATES TO THE ARDUINO MICROCONTROLLER VIA A SERIAL PORT → THROUGH A USB-SERIAL ADAPTER.
- CHECK TO MAKE SURE THAT THE DRIVERS ARE PROPERLY INSTALLED.

SETTINGS: TOOLS → BOARD

- NEXT, DOUBLE-CHECK THAT THE PROPER BOARD IS SELECTED UNDER THE TOOLS → BOARD MENU.



MAJOR CONCEPTS

- **SERIAL.PRINTLN(VALUE);**
 - PRINTS THE VALUE TO THE SERIAL MONITOR ON YOUR COMPUTER
- **PINMODE(PIN, MODE);**
 - CONFIGURES A DIGITAL PIN TO READ (INPUT) OR WRITE (OUTPUT) A DIGITAL VALUE



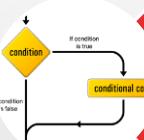
`digitalWrite()`



`analogWrite()`



`digitalRead()`



`if() statements / Boolean`



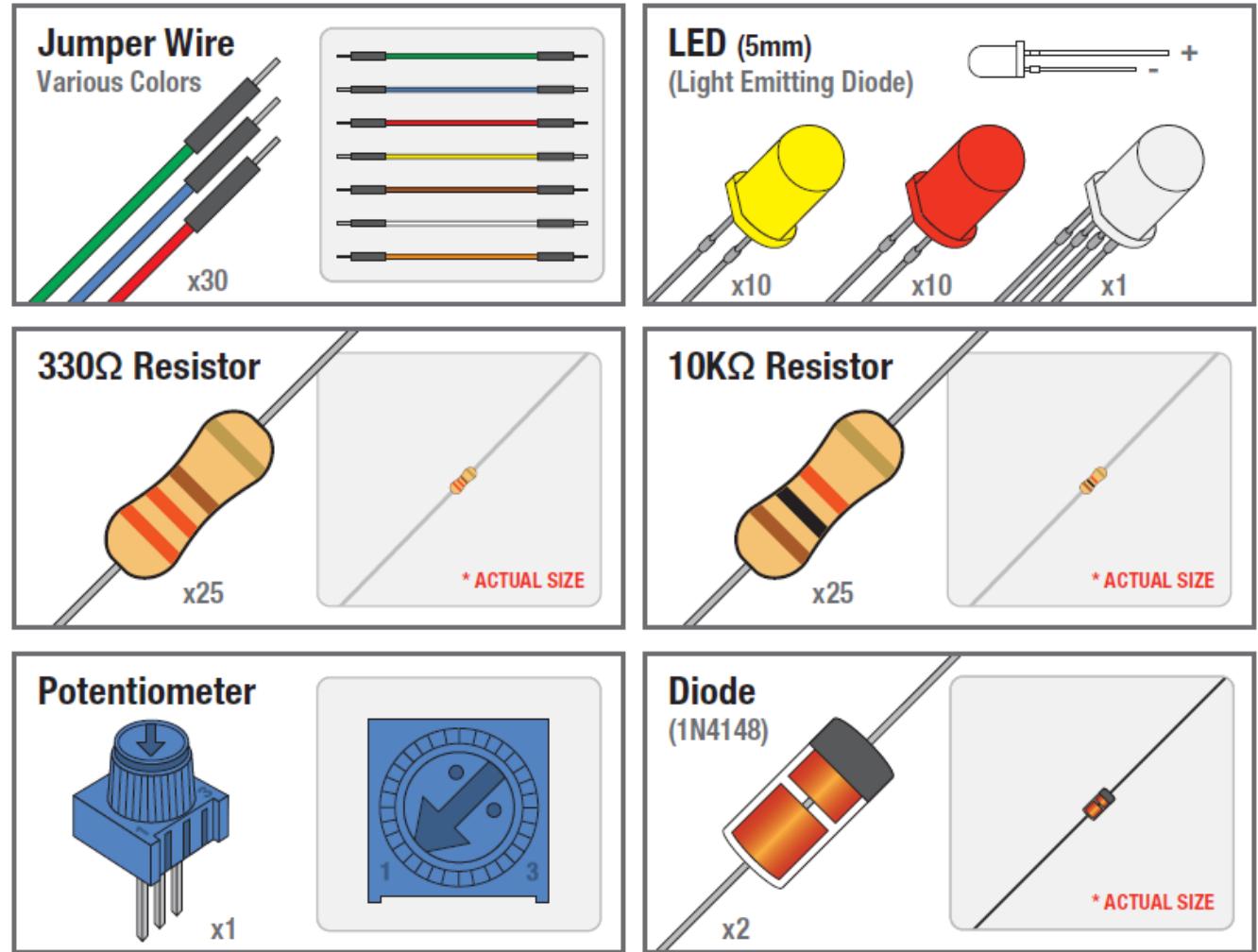
`analogRead()`



`Serial communication`

LEDS

- SIK COMPONENTS
- COMMON CATHODE RGB LED
- THESE 5MM UNITS HAVE FOUR PINS - CATHODE IS THE LONGEST PIN. ONE FOR EACH COLOR AND A COMMON CATHODE. USE THIS ONE LED FOR THREE STATUS INDICATORS OR PULSE WIDTH MODULATE ALL THREE AND GET MIXED COLORS!



LEDS

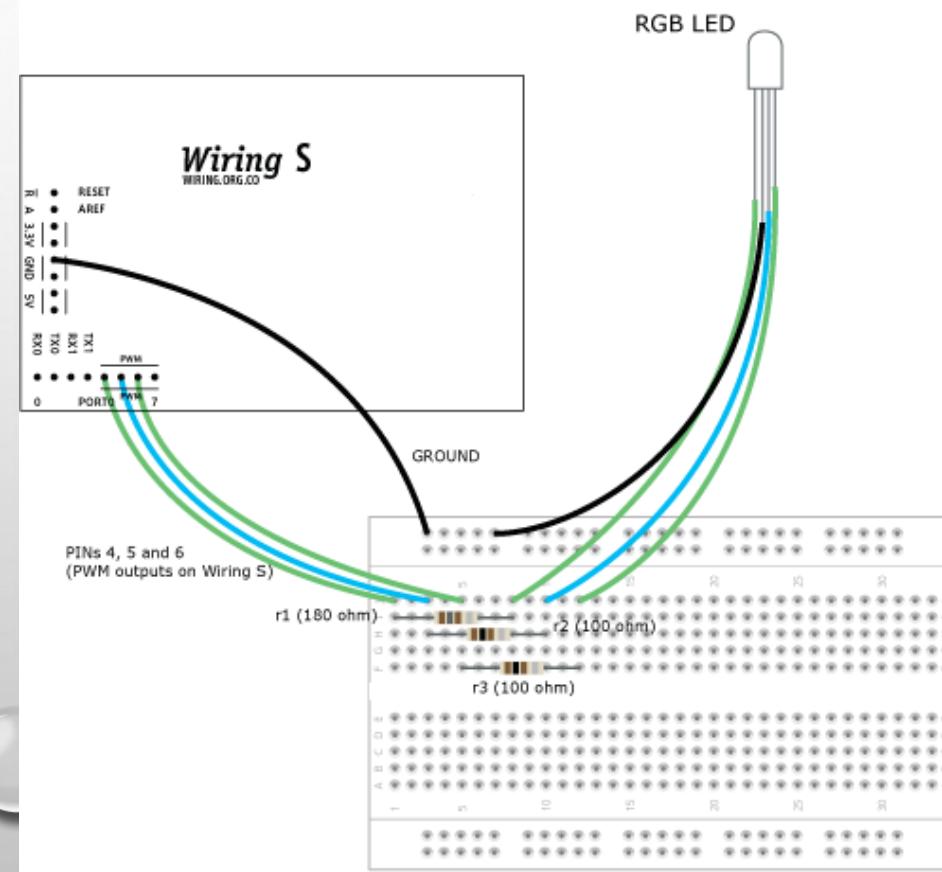
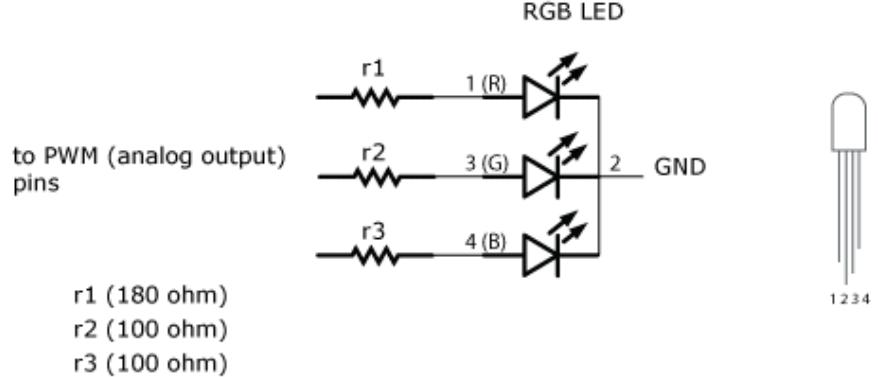
```
void setup()
{
    pinMode(13, OUTPUT); //configure pin 77
    as output
}
// blink an LED once
void blink1()
{
    digitalWrite(13,HIGH); // turn the LED on
    delay(500); // wait 500 milliseconds
    digitalWrite(13,LOW); // turn the LED off
}

void loop() //BLINK A LED REPEATEDLY
{
    delay(500); // wait 500 milliseconds
}

digitalWrite(13,HIGH); // TURN THE LED ON
delay(500); // WAIT 500 MILLISECONDS
digitalWrite(13,LOW); // TURN THE LED OFF
delay(500); // WAIT 500 MILLISECONDS
}
```

RGB LED

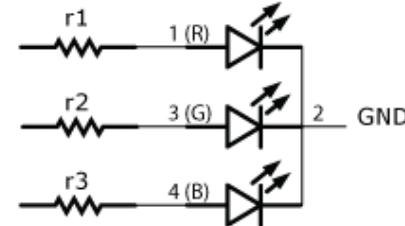
```
int REDPin = 4;      // RED pin of the LED to  
// PWM pin 4  
  
int GREENPin = 5;    // GREEN pin of the LED  
// to PWM pin 5  
  
int BLUEPin = 6;     // BLUE pin of the LED  
// to PWM pin 6  
  
int brightness = 0;   // LED brightness  
  
int increment = 5;    // brightness increment
```



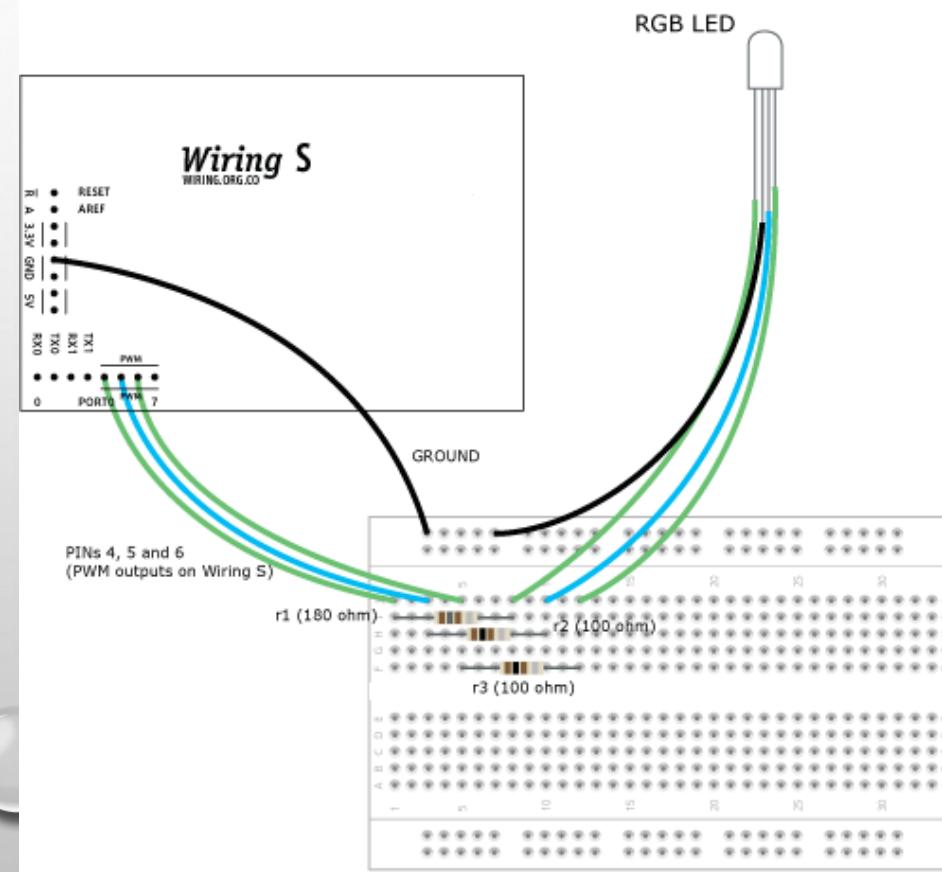
RGB LED

```
void setup()
{
    pinMode(REDPin, OUTPUT);
    pinMode(GREENPin, OUTPUT);
    pinMode(BLUEPin, OUTPUT);
    Serial.begin(9600);
}
```

to PWM (analog output) pins



r1 (180 ohm)
r2 (100 ohm)
r3 (100 ohm)



RGB LED

```
void loop()
{
    brightness = brightness + increment; // increment brightness
    for next loop iteration

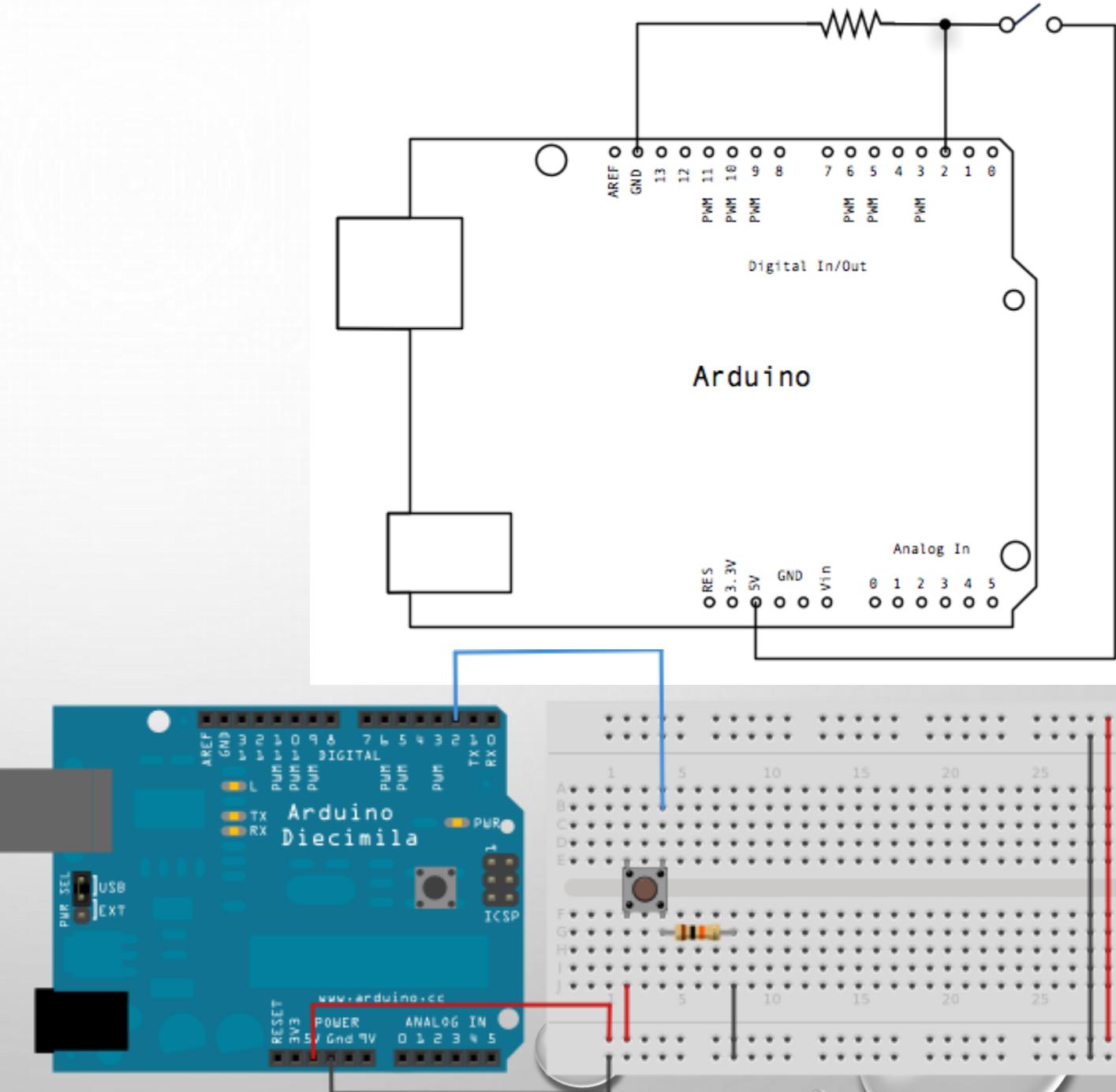
    if (brightness <= 0 || brightness >= 255) // reverse the
        direction of the fading
    {
        increment = -increment;
    }
    → brightness = constrain(brightness, 0, 255);
    analogWrite(REDPin, brightness);
    analogWrite(GREENPin, brightness);
    analogWrite(BLUEPin, brightness);

    delay(20); // wait for 20 milliseconds to see the dimming
    effect
}
```

SWITCHES

```
CONST INT INPUTPIN = 2;  
  
VOID SETUP() {  
    PINMODE(INPUTPIN, INPUT);  
}  
  
VOID LOOP() {  
    INT VAL = DIGITALREAD(INPUTPIN);
```

- ## • WHAT DOES A RESISTOR DO ?



INTERRUPT PIN

INTERRUPT PINS OF UNO BOARD ARE 2,3

THEY ALLOW EXECUTION TO JUMP ONCE AN INPUT IS HIGH OR LOW (ACCORDING TO PULLUP/PULLDOWN MODES)

THE PROGRAM WILL CONTINUE EXECUTION FROM WHERE IT STOPPED

THE FOLLOWING CODE SWITCHES THE STATE BETWEEN LOW AND HIGH ONCE THE INTERRUPT PIN 2 HAS HIGH INPUT

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

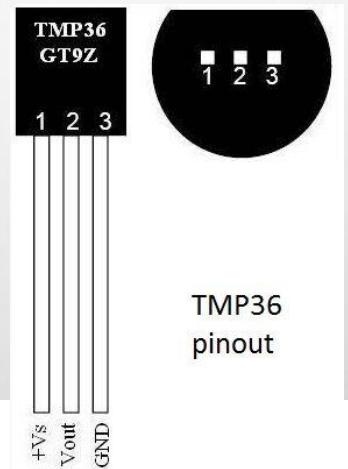
void loop() {
  digitalWrite(ledPin, state);
}

void blink() {
  state = !state;
}
```

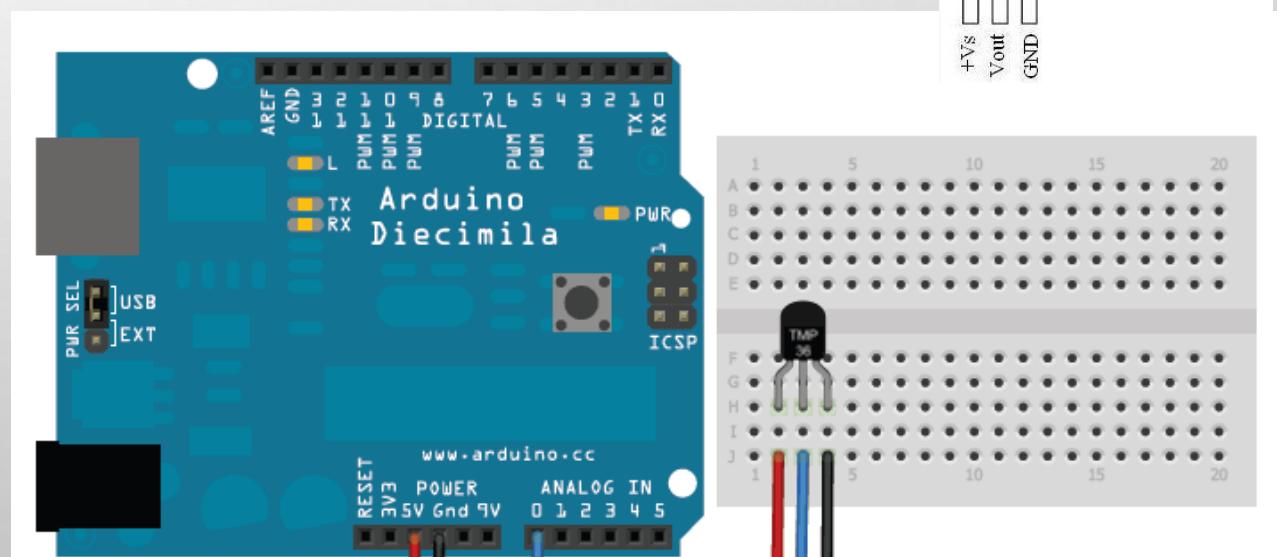
ANALOG INPUT – TEMPERATURE SENSOR

- TMP36
- **AD22100 TEMPERATURE SENSOR**
- OUTPUT V_O RANGING FROM 0.25 V AT -50°C TO +4.75 V AT +150°C
- ASSUME LINEAR RESPONSE AND TEMPERATURE COEFFICIENT OF 22.5 MV/°C , VOLTAGE AT 0°C IS 1.375
-

```
INT SENSORVALUE = ANALOGREAD(A0);
FLOAT VOLTAGE= SENSORVALUE * (5.0 / 1023);
TEMPERATURE = (FLOAT VOLTAGE-1.375) / 0.0225
```

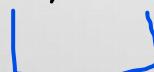


TMP36
pinout



ANALOG INPUT – RANGE MAPPING

```
CONST INT POTPIN = 0; // SELECT THE INPUT PIN FOR THE POTENTIOMETER  
VOID LOOP() {  
INT VAL; // THE VALUE COMING FROM THE SENSOR  
INT PERCENT; // THE MAPPED VALUE  
VAL = ANALOGREAD(POTPIN); // READ THE VOLTAGE ON THE POT (VAL RANGES  
FROM 0 TO 1023)  
PERCENT = MAP(VAL,0,1023,0,100); // PERCENT WILL RANGE FROM 0 TO 100.
```



CONTROLLING DC MOTOR DIRECTION & SPEED

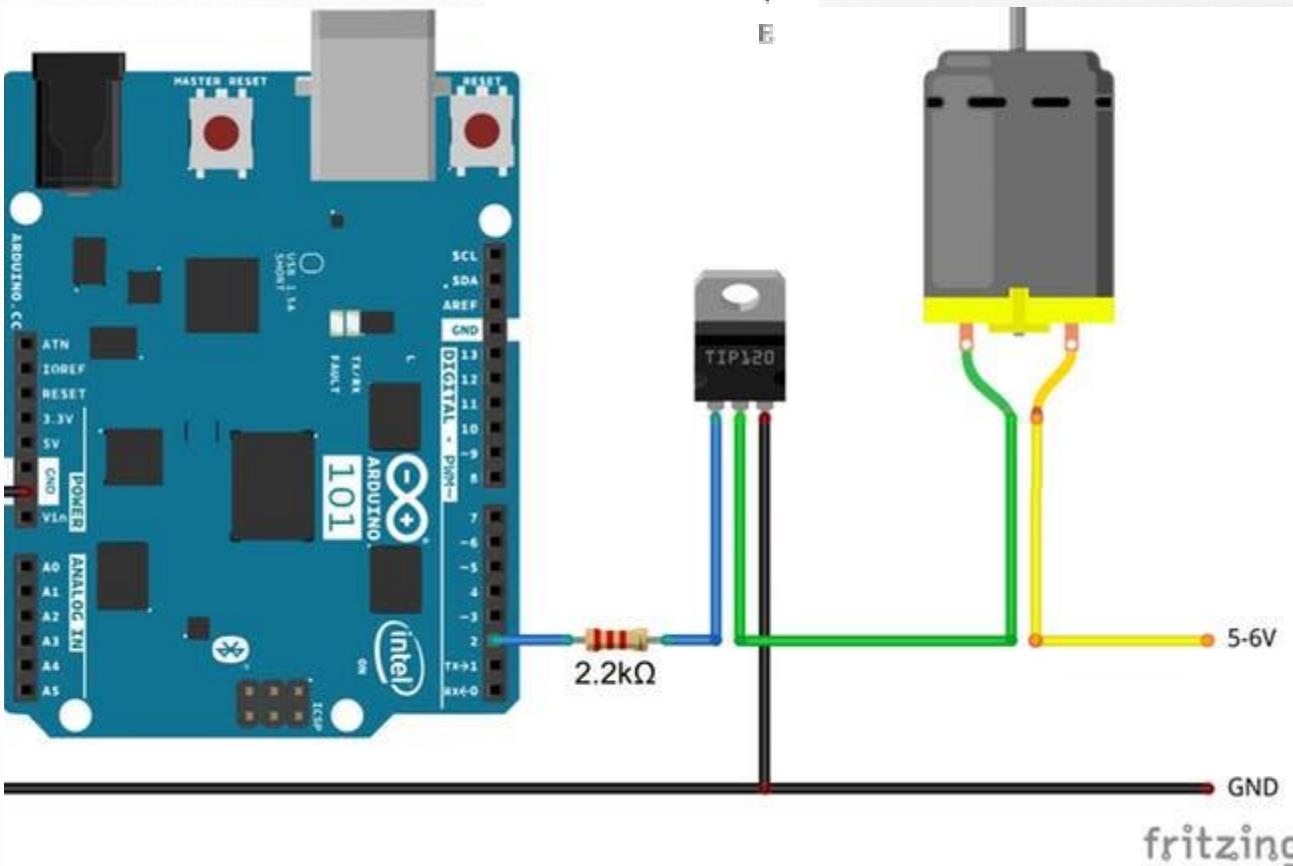
- OPEN DISCUSSION

CONTROLLING DC MOTOR

```
#define DC_MOTOR_PIN 2

void setup() {
  /* Initialize DC motor control pin as digital output */
  pinMode( DC_MOTOR_PIN, OUTPUT );
}

void loop() {
  /* Run motor */
  digitalWrite( DC_MOTOR_PIN, HIGH );
  /* Let it run for a while */
  delay( 2000 );
  /* Stop motor */
  digitalWrite( DC_MOTOR_PIN, LOW );
  /* Do nothing */
  while( 1 );
}
```

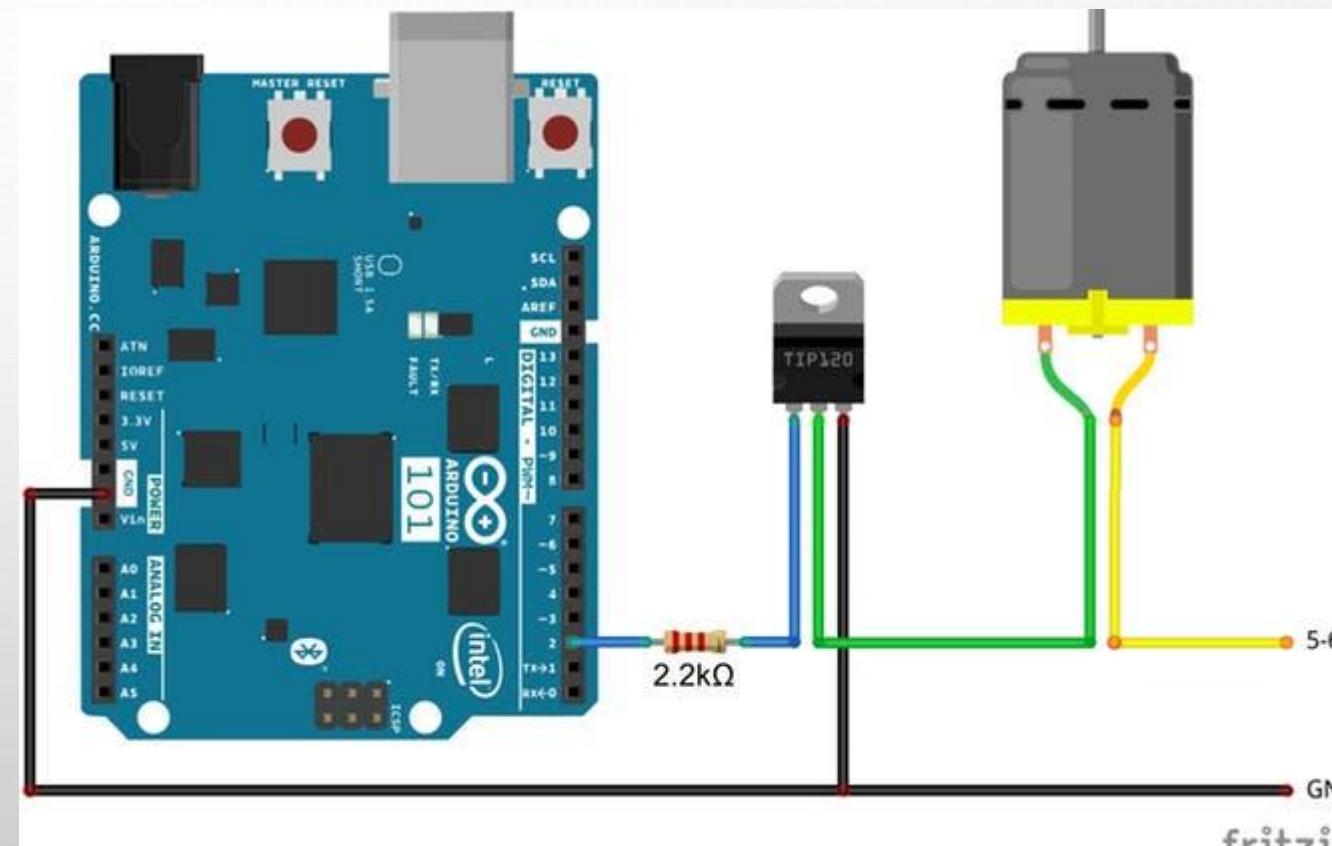


CONTROLLING DC MOTOR SPEED

```
#define DC_MOTOR_PIN 3

void setup() {
  /* Initialize DC motor control pin as digital output */
  pinMode( DC_MOTOR_PIN, OUTPUT );
}

void loop() {
  /* Gradually increase motor speed to full speed */
  for( int i = 0; i < 255; i = i+5 )
    analogWrite( DC_MOTOR_PIN, i );
  /* Gradually decrease motor speed to half speed */
  for( int i = 255; i > 128;i = i-5 )
    analogWrite( DC_MOTOR_PIN, i );
  /* Stop the motor */
  analogWrite( DC_MOTOR_PIN, 0 );
  /* Do nothing */
  while( 1 );
}
```



THE BUILD PROCESS

- CODE VERIFICATION C / C++ SYNTAX CHECK
- TRANSFORMATIONS (INCLUDING ARDUINO.H)
- CODE COMPILATION AVR-GCC -> MACHINE BASED INSTRUCTIONS .O FILE
- LINKAGE WITH LIBRARIES (ORIGINAL IMPLEMENTATIONS) .HEX FILE
- UPLOAD INTO BOARD FLASH MEMORY - AVRDUDE

DELAY

- DELAY

BASIC DELAY

```
void setup() {  
    pinMode(13, OUTPUT);  
}  
  
void loop() {  
    digitalWrite(13, HIGH); // set the LED on  
    delay(1000);          // wait for a second  
    digitalWrite(13, LOW); // set the LED off  
    delay(1000);          // wait for a second  
}
```

```
void setup() {  
    pinMode(13, OUTPUT);  
}  
  
void loop() {  
    digitalWrite(13, HIGH); // set the LED on  
  
    for (int x=0; x < 1000; x++) { // Wait for 1 second  
        delay(1);  
    }  
    digitalWrite(13, LOW); // set the LED on  
  
    for (int x=0; x < 1000; x++) { // Wait for 1 second  
        delay(1);  
    }  
}
```

```
void setup() {
    pinMode(13, OUTPUT);
    pinMode(12, OUTPUT);
}

void loop() {
    digitalWrite(13, HIGH); // set the LED on

    for (int x=0; x < 1000; x++) { // wait for a secoond
        delay(1);

        if (x==500) {
            digitalWrite(12, HIGH);
        }
    }

    digitalWrite(13, LOW); // set the LED off

    for (int x=0; x < 1000; x++) { // wait for a secoond
        delay(1);

        if (x==500) {
            digitalWrite(12, LOW);
        }
    }
}
```

```
1 unsigned long interval=1000; // the time we need to wait
2 unsigned long previousMillis=0; // millis() returns an unsigned long.
3
4 bool ledState = false; // state variable for the LED
5
6 void setup() {
7   pinMode(13, OUTPUT);
8   digitalWrite(13, ledState);
9 }
10
11 void loop() {
12   unsigned long currentMillis = millis(); // grab current time
13
14   // check if "interval" time has passed (1000 milliseconds)
15   if ((unsigned long)(currentMillis - previousMillis) >= interval) {
16
17     ledState = !ledState; // "toggles" the state
18     digitalWrite(13, ledState); // sets the LED based on ledState
19     // save the "current" time
20     previousMillis = millis();
21   }
22 }
```

REMEMBERING CHANGES

- THE SUPPORTED MICRO-CONTROLLERS ON THE UNO ARDUINO BOARDS HAVE DIFFERENT AMOUNTS OF EEPROM: 1024 BYTES ON THE ATMEGA328
- EPROM IS MEMORY WHOSE VALUES ARE KEPT WHEN THE BOARD IS TURNED OFF (LIKE A TINY HARD DRIVE).
- YOU CAN READ , WRITE , CLEAR , UPDATE
- [EEPROM UPDATE](#): STORES VALUES READ FROM A0 INTO EEPROM, WRITING THE VALUE ONLY IF DIFFERENT, TO INCREASE EEPROM LIFE.

REMEMBERING CHANGES

EEPROM

write()

Description

Write a byte to the EEPROM.

Syntax

EEPROM.write(address, value)

Parameters

address: the location to write to, starting from 0 (*int*)

value: the value to write, from 0 to 255 (*byte*)

```
#include <EEPROM.h>
int addr = 0;
void setup() {}
void loop() {
    int val = analogRead(0) / 4;
    →EEPROM.write(addr, val);
    addr = addr + 1;
    if (addr == EEPROM.length()) {
        addr = 0;
    }
    delay(100);
}
```

REFERENCES

- [HTTPS://WWW.BALDENGINEER.COM/MILLIS-TUTORIAL.HTML](https://www.baldengineer.com/millis-tutorial.html)
- [HTTPS://WWW.ARDUINO.CC/EN/REFERENCE/MILLIS](https://www.arduino.cc/en/Reference/Millis)
- [HTTPS://THEALARM CLOCK SIXAM.COM/2012/01/22/OBJECT-ORIENTED-BLINKING-WITH-ARDUINO/](https://thealarmclocksixam.com/2012/01/22/object-oriented-blinking-with-arduino/)