**GUÍA PERSONAL ARDUINO**

Autor: Adrián Ricárdez Ortigosa. Facultad de Ingeniería UNAM.

Nota 1: No todas los procedimientos están apuntados, pero se encuentran los más relevantes o los que más propensos están a olvidarse. Los ejemplos que no presenten Void Setup y Loop es que ya lo tienen incluido y hay que colocarlos en el lugar correspondiente.

Nota 2: Los rebotes tardan de 10 a 20 milisegundos, usar una función delay para eliminarlos (como en los push buttons).

**Estructura del programa:**

void setup() {

Aquí va a ir todo lo que ejecutará solo una vez, por ejemplo: pines de entrada y/o salida, variables, serial.begin, etc.

}

void loop() {

Aquí va todo lo que queremos que se esté ejecutando todo el tiempo (desde que se prende hasta que se apaga), por ejemplo condicionantes, activadores, serial.print, etc.

}

**Comandos**

pinMode(# del pin/ nombre del pin, OUTPUT/INPUT); **Declara un pin de salida o entrada. Va en setup.**

digitalWrite(# del pin/ nombre del pin, HIGH/LOW); **Activa o desactiva un pin.**

digitalRead(pin de entrada); **Lee el estado de alguna entrada digital. Ésta generalmente se guarda y luego se usa, ejemplo:**

*int estado=0;*

*pinMode(8, INPUT);*

*estado= digitalRead(8);*

**En este caso el estado que devolvería sería HIGH o LOW.**

analogWrite(# del pin/ nombre del pin, valor querido); **Activa o desactiva un pin PWM, recordar que 255 es el valor máximo, ejemplo:**

*const int LED= 5;*

*pinMode(LED, OUTPUT);*

*analogWrite(LED, 122);* **ó** *analogWrite(LED, 0-255);* **ó** *analogWrite(LED, 0); analogWrite(LED, otravariable)  este último es como para que cambie conforme cambia otra variable * **, etc…**

analogRead(# del pin/ nombre del pin); **Lee el estado de alguna entrada analógica. Ésta generalmente se guarda y luego se usa, ejemplo:**

*int brillo;*

*const int POT= 0;*

*brillo= analogRead(POT)/4;*

**En este caso los valores que maneja son entre 0 y 1023, escribí entre 4 porque no es compatible directamente con los valores de analogWrite, así lo hago a proporción.**

delay(tiempo en mili segundos/alguna variable); **Tiempo de espera para realizar la siguiente instrucción.**

int nombre de la variable; **Declara variables enteras (puede cambiar).**

const int nombre de la variable= valor asignado; **Declara una constante entera, no puede cambiar.**

boolean nombre de la variable; **Todo valor que solo es posible tener dos valores: true y false.**

if(variable == condición){

Aquí va el código a cumplir después de la condicionante.

}

else{

Aquí va el código a cumplir en caso de que no se cumpla el condicionante.

}

**Para poder agregar otra condicionante AND en la misma condicionante se usa &&. Para agregar otracondicionante OR se usa ||.**

for… **Ya sabes cómo es y para qué es… Es como un “loop”. TambiÉn sabes de while, y do-while.**

Serial.begin(9600); **Inicializa la comunicación serial. Va dentro de setup. El baudrate tú lo elijes: 9600, 115200, etc.**

Serial.print(“texto”); **Escribe el texto en el puerto serial. Si se escribe “Serial.println(blabla)” se saltará una línea hacia abajo.**

Serial.print(variable); **Manda el valor de la variable al puerto serial.**

long nombre de la variable; **Maneja números enteros como int, pero tiene un espacio mucho más grande para guardar variables. También al hacer multiplicaciones generalmente que generan números muy grandes para ser int, se pone por ejemplo en voltaje:**

*miliVolts= (analogRead()\*5000****L****)/1023;*

map(variable que vamos a escalar, valor mínimo de la variable, valor máximo de la variable, 0, 255); **Mapea o cambia la escala de los datos. Puse 0 a 255 porque generalmente se usa ese en PWM. La mayor parte de las veces va acompañada de la siguiente función, “constrain”.**

constrain(variable, 0, 255); **Restringe los límites de la función map.**

**Funciones: Son un pedazo de código que está tanto fuera del void setup, como del void loop. El objetivo de las funciones es que se puedan mandar llamar cada vez que se necesiten (desde void loop). Para crear una función que devuelva valores enteros, sería algo parecido a esto:**

int nombre de la función(aquí va todo lo que necesita tu función){

} **También puede ser de tipo booleano, vacío (void, que no devuelve nada), etc. Para devolver el valor se usa “return”.**

**Ejemplo:**

*int suma(int x, y) {*

*int resultado;*

*resultado= x+y;*

*return resultado;*

*void loop() {*

*int valor= suma(4,6);* **Aquí se manda llamar a la función suma, y opera con los números establecidos.**

*}*

return variable; **Se escribe al final de una función. Devuelve el valor de la variable especificada.**

void función(){

}; **Si no va a devolver nada, se escribe void, que significa “vacío”, por eso los paréntesis están vacíos, y no hay ninguna variable adentro.**

Ping nombre del sensor ultrasónico = Ping(# de pin/ nombre del pin); **Estamos usando uno de los métodos de la librería Ping. Es necesaria para usar el sensor ultrasónico. Afuera de loop o setup.**

nombre del sensor ultrasónico.fire(); **Con esto estamos activando nuestro sensor ultrasónico, es necesario para que funcione. Va en loop.**

variable de distancia= nombre del sensor ultrasónico.centimeters(); **Nos va a devolver la distancia en centímetros. Va en loop.**

#include <Servo.h> **Esta librería es necesaria para usar Servos, que por cierto desactiva las funciones PWM para los pines 9 y 10 de Arduino UNO.**

Servo nombre del servo; **Crea un objeto de la librería Servo.h, para decirle al arduino que usarás un servo . Fuera de Setup y Loop.**

Nombre de servo.attach(pin del servo, pulso mínimo, pulso máximo); **Configuración del servo. Va en Setup.**

Nombre del servo.write(variable donde esta guardado el ángulo); **Activa el servo. En Loop.**

**Datos curiosos BREAK;**

**Si se tiene por ejemplo:**

**while(true){**

**while(true){**

**break;**

**digitalWrite(13,HIGH);}**

**}**

**El break que está ahí, romperá únicamente el while al que corresponde, que sería el segundo.**

**Además, la instrucción digitalWrite(13,HIGH); nunca se ejecutará, porque el break está antes de ella, y rompe todo el while desde ahí.**

**Funcionará de la misma forma con un for. El break sólo rompe con uno de estos ciclos.**

**PINES ARDUINO**

DIGITAL (PWM~)= Significa que en los pines con el símbolo ~ podemos hacerlos salidas analógicas.

ANALOG IN= Son las entradas analógicas que por default ya están declaradas como entradas, no es necesario declarar como entrada estos pines.

Comunicación Serial= Sólo se da en los pines 0 y 1 del arduino.

Se tienen 19 posibles entradas digitales, y 18 salidas digitales.

I2C = Protocolo, está en los pines A4 y A5.

Reset = Basta con ponerle un push button a Tierra (GND) para que se reinicie cada vez que se pulsa.

Vin = Entra al regulador de voltaje.

**HARDWARE**

Nota1: Los transistores o sensores en forma de transistores siempre se van a ver desde la vista plana y de izquierda a derecha.

**SENSOR DE TEMPERATURA LM35**

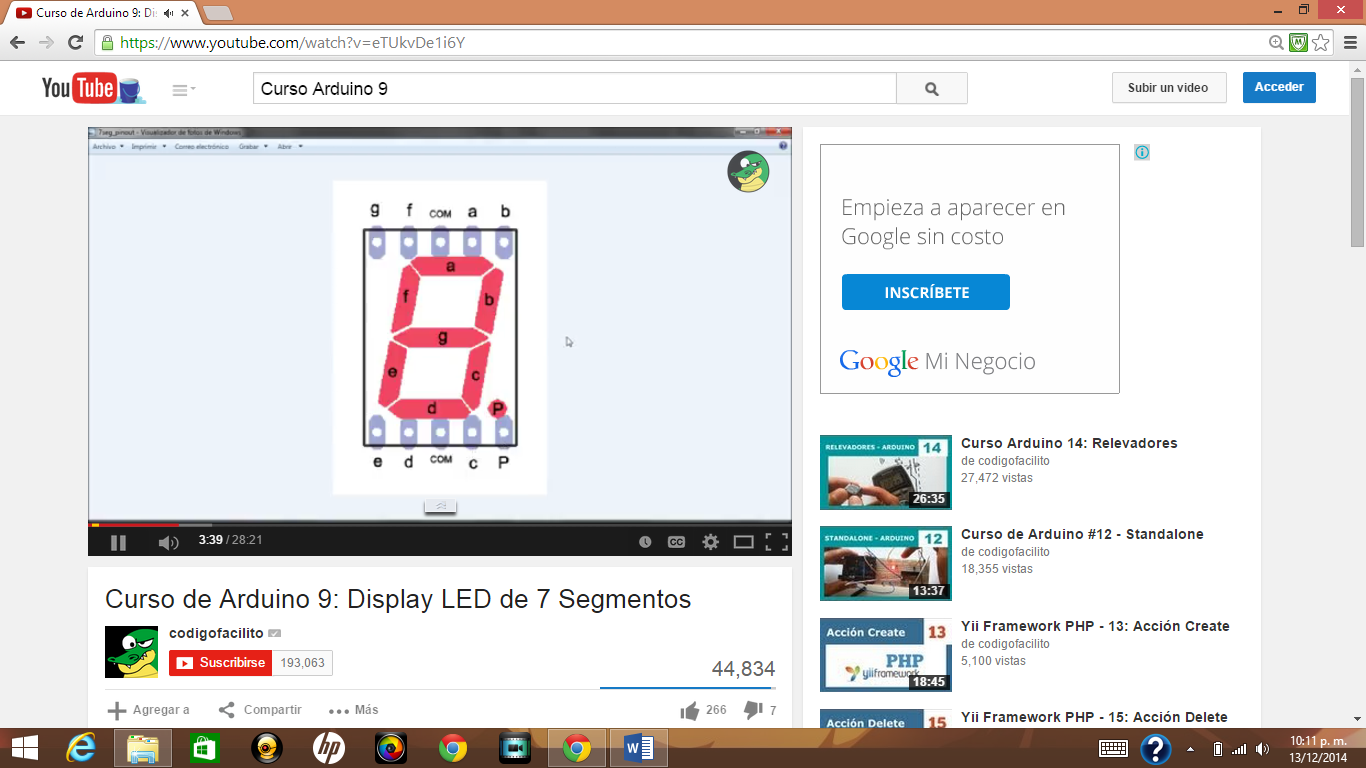
(1= +5V; 2= AnalogPin; 3= GND), 10mV/°C, va de 0 a 5000mV, en el código se hace una regla de 3 para obtener los mV, ya que dará un dato entre 0 y 1023 del analogRead(), y esos mV se dividen entre 10, y esa es la temperatura resultante.

**PUSHBOTTONS**

Usan resistencias de 10KΩ o 1KΩ En pull-up o pull-down. Yo casi siempre uso pull-up.

**DISPLAY DE 7 SEGMENTOS DE CÁTODO COMÚN**

**com=** Punto en común, en este caso es GND. No es necesario conectar ambos a GND, con uno es suficiente para que sirva todo el display. Cada led necesita una resistencia, no pueden tener una resistencia en común. Todos los LED’s usan resistencias de 220Ω.



**ZUMBADOR O BUZZER**

(+= DigitalPin; -= GND) Produce sonido, es un tipo micrófono-ruedita, cómpratelo! ;)

Sí tienen polaridad.

**SENSOR ULTRASÓNICO**

[Viéndolo de frente] (1= GND; 2= DigitalPin (trigger), 3= DigitalPin (echo), 4= +5V)

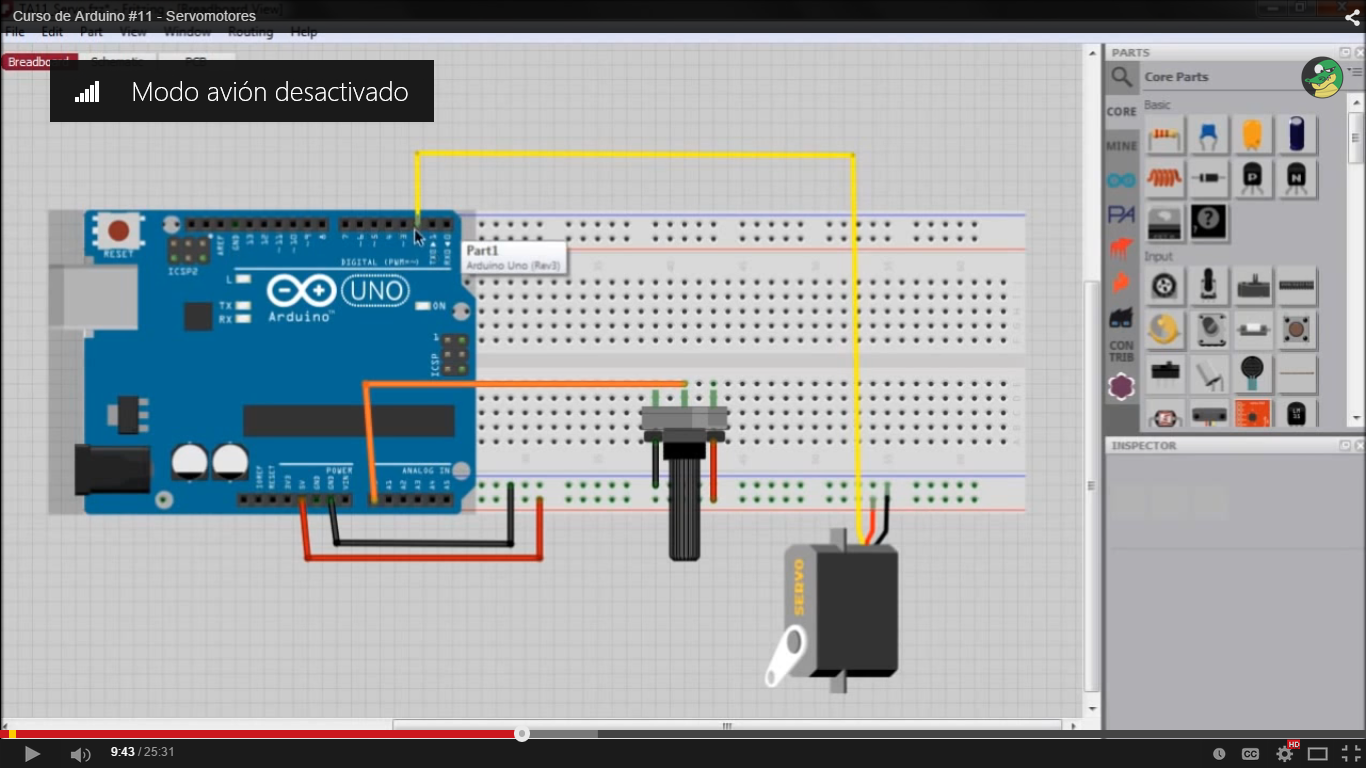
**SERVOMOTORES**

(Rojo generalmente en medio= +5V; Negro/Café= GND; Amarillo/Naranja= DigitalPin)

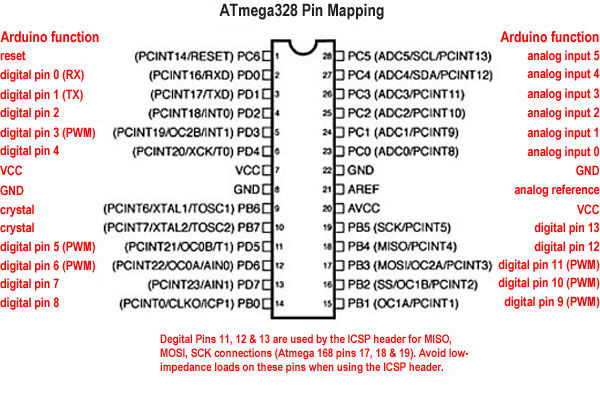
Doble fuente de energía para el servo, la del USB y la de plug-in, ya que se puede llegar a quemar tu arduino o tu computadora, +9V- 1A o 1.5ª. Se controla a base de pulsos de 1 milisegundo (0°) hasta 2 milisegundos (180°) aproximadamente, tienes que experimentarle. **No es necesario usar ni Analog, ni PWM del arduino, ya que se puede dañar.** Usaremos microsegundos. Para usarlo se guarda: valor= analogRead(pindelpotenciometro);

angulo= map(valor, 0, 1023, 0, 180); en 0 hasta 180 es el ángulo de giro que quiera. En dado caso que el servo gire en sentido horario, hay que poner primero 180 y luego 0. Va en loop esto.

Al final ponle un delay(20); para darle oportunidad al servo de reaccionar bien. Varía ese 20, depende del peso que cargue, a más peso, más delay.

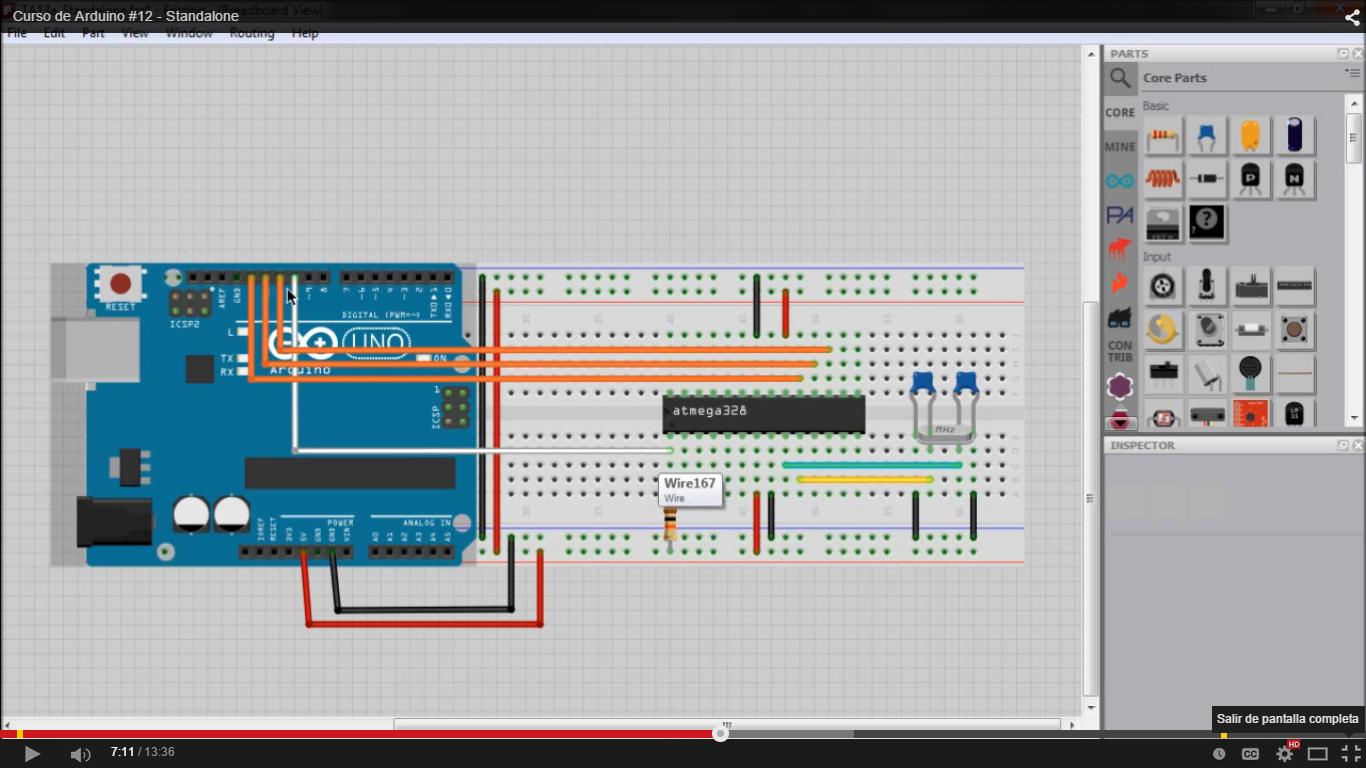


**MICROCONTROLADOR ATMEGA328P (PARA ARDUINO UNO)**



**STANDALONE**

Paso 1: Para poder hacer proyectos permanentes, hay que trabajar sin sacar el microcontrolador del Arduino UNO al principio para poder cargar el bootloader al otro ATMEGA328P-PU que tengo fuera para que pueda entender el lenguaje C como se muestra en la siguiente figura. Compilar y cargarle el ArduinoISP que se encuentra en los programas ejemplo. Checar que en Tools Board esté Arduino UNO y en Programmer esté en Arduino as ISP después de cargarle el programa. Y para terminar darle en Burn Bootloader.



**Pines ATMEGA328P-PU para bootloader**

Pin 1 (reset) = Resistencia de 10k a +5V y Pin 10 Arduino UNO

Pin 7 (VCC) = +5V

Pin 8 (GND) = GND

Pin 9 y 10 (crystal) = Cristal de 16MHz, no importa la polaridad

Pin 9 y 10(cystal) = Un capacitor por cada pin de 18 a 22 picofaradios a GND

Pin 17 (Digital Pin 11 PWM) = Pin 11 Arduino

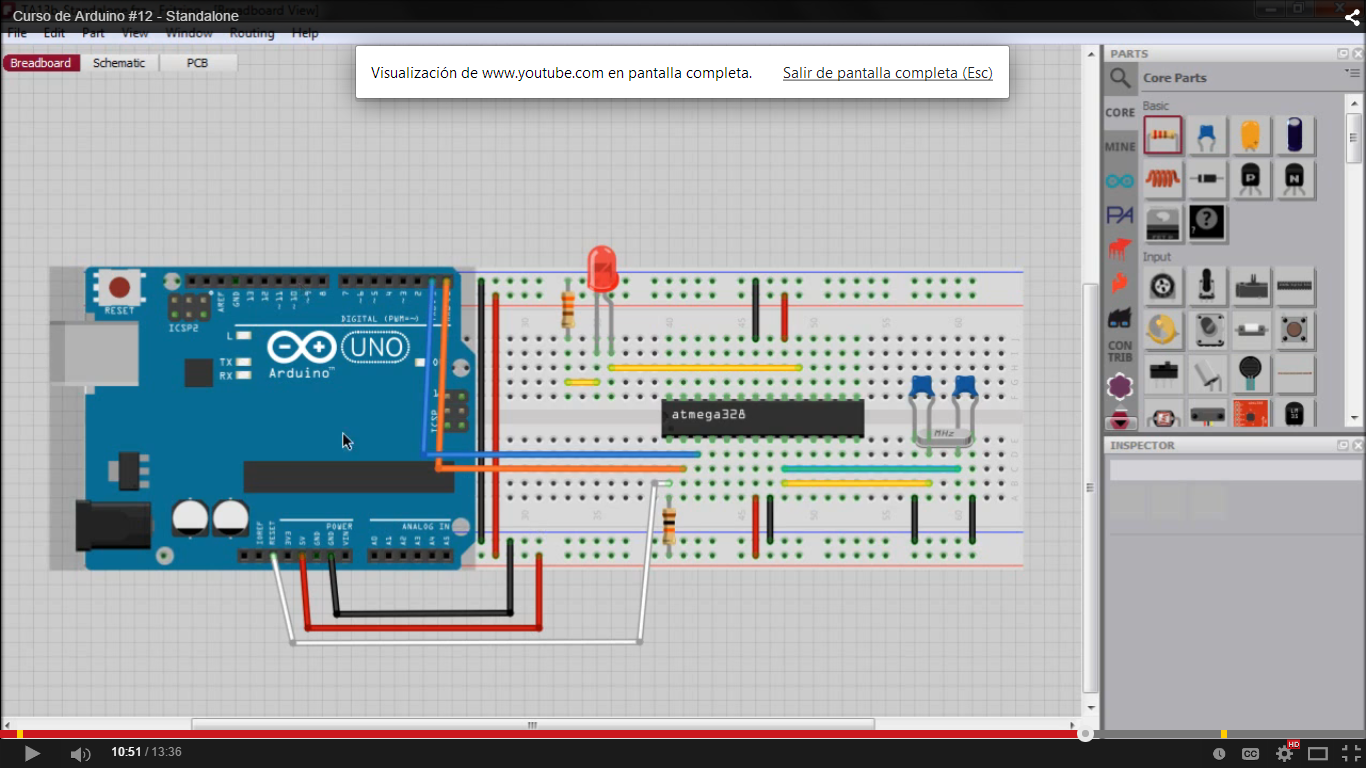
Pin 18 (Digital Pin 12) = Pin 12 Arduino

Pin 19 (Digital Pin 13) = Pin 13 Arduino

Pin 20 (VCC) = +5V

Pin 22 (GND) = GND

Paso 2: Quitando el microcontrolador del Arduino UNO, ya podemos cargar el programa que deseemos como comúnmente se carga, con “AVRISP mkll”. Recordar que la imagen tiene un error, el reloj va como en la imagen anterior. Es estrictamente necesario que el de RESET esté conectado (el blanco como en la imagen).



**Pines básicos atmega328 para cargar el programa (ya sin el ATMEGA328P-PU del Arduino UNO adentro)**

Pin 1 (reset) = Resistencia de 10k a +5V y Reset Arduino

Pin 2 (Digital Pin 0 (RX)) = Digital Pin 0 (RX) Arduino

Pin 3 (Digital Pin 1 (TX)) = Digital Pin 1 (TX) Arduino

Pin 7 (VCC) = +5V

Pin 8 (GND) = GND

Pin 9 y 10 (crystal) = Cristal de 16MHz, no importa la polaridad

Pin 9 y 10(cystal) = Un capacitor por cada pin de 18 a 22 picofaradios a GND

Pin 20 (VCC) = +5V

Pin 22 (GND) = GND

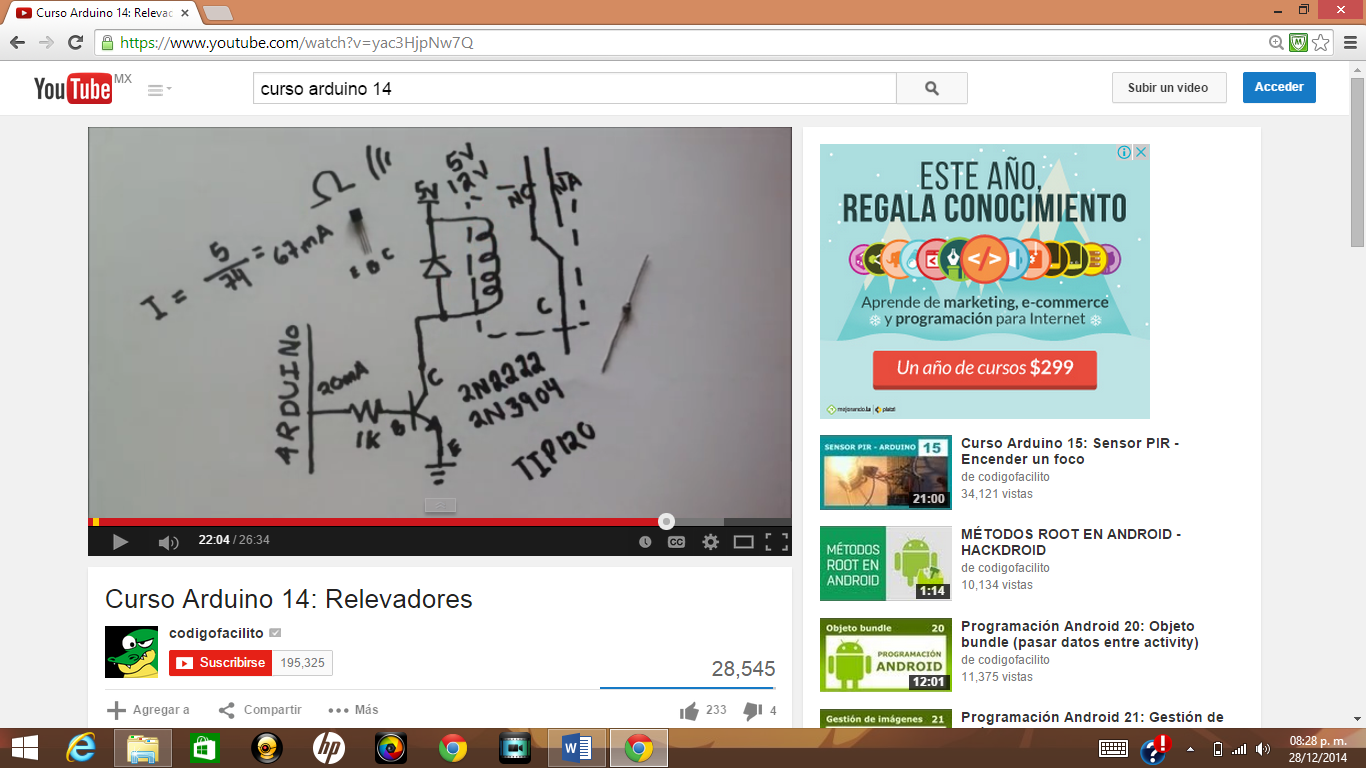
Nota: hay un pequeño error en el cristal, va en los pines 9 y 10, no solo en el 9.

**En este paso, ya puedo conectar todo lo que yo quiera a mi ATMEGA Standalone para poder hacer el proyecto que necesito permanente. Se recomienda poner un push-button a tierra del pin 1 MCLR (Master Clear) para poder resetearlo, y si se usa PCB (circuito impreso), poner los 5 pines (Rx/Tx/+5V/GND/Reset) como bus de datos en header macho para poder programarlo fácilmente.**

**Recomendación: Para ver que ya funciona mi nuevo atmega, le cargo el programa blink en 0 basics en examples, y conecto el led con su resistencia al pin 13 (patita 19), y listo, tiene que parpadear..**

**RELEVADORES**

**Es necesario usa uno o más transistores, ya que la corriente que usa un relevador es aproximadamente de 67mA si usamos +5V, y en caso de arduino, los pines solo resisten o da más bien 20mA, por lo cual puede causar corto circuito.**



**TRANSISTORES**

**Generalmente, en la base se conecta con una resistencia lo que controla el circuito, en sentido de arduino, sería el pin con una resistencia de 1K, el emisor a tierra y el colector a un relevador por ejemplo, y el relevador a +5V junto con un diodo en paralelo para evitar que se queme el circuito. Y así, amplificamos la corriente.**

**Más usados para unos cientos de mA:**

**2N2222 (1= Emisor ; 2= Base; 3= Colector)**

**2N3904**

**PN2222**

**BC547-548**

**Más usados para mayor corriente aún:**

**TIP120**

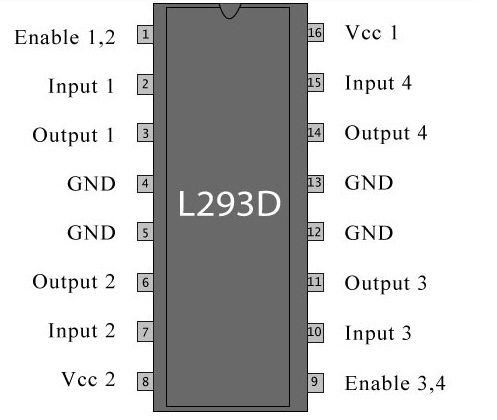
**DIODOS**

Técnicamente lo que dicen es que dejan fluir corriente (ánodo) de un lado a otro pero no en sentido contrario (cátodo). Para saber cómo orientarlo, tiene una línea, esa línea indica el cátodo

**Modelos:**

**1N4001, 1N4002, 1N4003 ó 1N4004…**

**PUENTE H L293D- Driver para motores de DC**



Vcc 1  Voltaje lógico (Con el que va a trabajar el Puente H), generalmente es de +5V

Vcc  Voltaje de Motores, este rango varía entre +7V y +14V

Input 1, 2, 3 y 4  Entradas Digitales

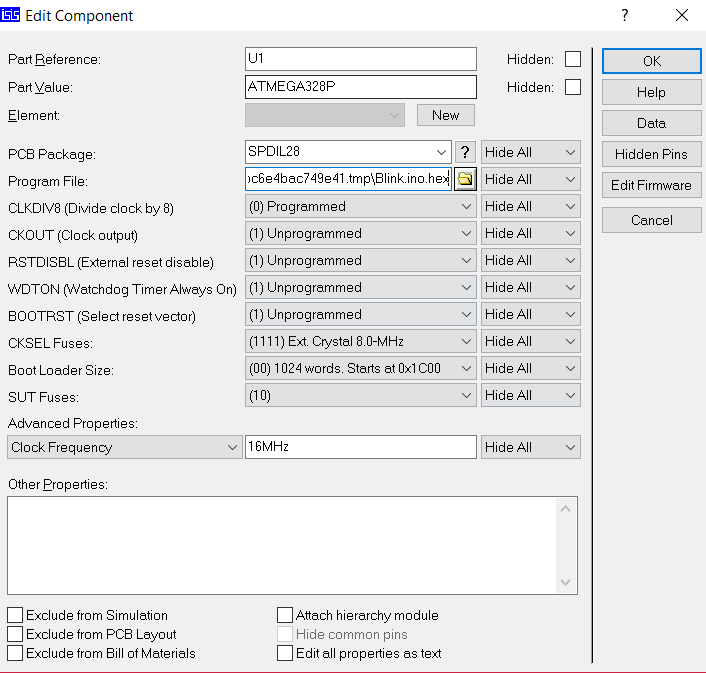
Output 1, 2, 3 y 4  Salidas Digitales o Análogas (Dependiendo si se usan los Enables)

Enable 1,2 y 3,4  Para el control de velocidades, si se va a usar totalmente digital el puente H, éstas van conectadas a VCC 1, si no, van conectadas a alguna salida PWM~ del Arduino usando analogWrite( #pinPWM~, rango de 0 a 255).

**Simular ATMega 328P en Proteus**

Configuración Básica para Blink y hacer parpadear un LED:

Para hacer parpadear el LED necesitamos la siguiente configuración en el chip:



Cuando carguemos el archivo, buscamos la siguiente dirección en el buscador de archivos:

C:\Users\Sigma17\AppData\Local\Temp\**build9c9ef3bdfe2fccb480bc6e4bac749e41.tmp**

En este caso, la última carpeta se puede buscar/sustituir básicamente por la carpeta creada más recientemente, y luego seleccionamos el archivo .HEX que corresponde al programa en instrucciones entendibles por el simulador.

Nota: En el programa principal de Arduino IDE utilizar delays muy bajos para poder notar los cambios, ya que va muy lenta la simulación en Proteus.

**Para agregar bibliotecas extras a la plataforma Arduino IDE**

<https://www.arduino.cc/en/Guide/Libraries>