Sensor ultrasonico

/\*

Sensor de proximidad y al ser inferior a 10cm

envia un pulso de alarma por el pin 13

HC-SR04 conexiones:

VCC al arduino 5v

GND al arduino GND

Echo al Arduino pin 6

Trig al Arduino pin 7

Descargar planos de conexiones en http://elprofegarcia.com/

\*/

#define Pecho 6

#define Ptrig 7

long duracion, distancia;

void setup() {

Serial.begin (9600); // inicializa el puerto seria a 9600 baudios

pinMode(Pecho, INPUT); // define el pin 6 como entrada (echo)

pinMode(Ptrig, OUTPUT); // define el pin 7 como salida (triger)

pinMode(13, 1); // Define el pin 13 como salida

}

void loop() {

digitalWrite(Ptrig, LOW);

delayMicroseconds(2);

digitalWrite(Ptrig, HIGH); // genera el pulso de triger por 10ms

delayMicroseconds(10);

digitalWrite(Ptrig, LOW);

duracion = pulseIn(Pecho, HIGH);

distancia = (duracion/2) / 29; // calcula la distancia en centimetros

if (distancia >= 500 || distancia <= 0){ // si la distancia es mayor a 500cm o menor a 0cm

Serial.println("---"); // no mide nada

}

else {

Serial.print(distancia); // envia el valor de la distancia por el puerto serial

Serial.println("cm"); // le coloca a la distancia los centimetros "cm"

digitalWrite(13, 0); // en bajo el pin 13

}

if (distancia <= 10 && distancia >= 1){

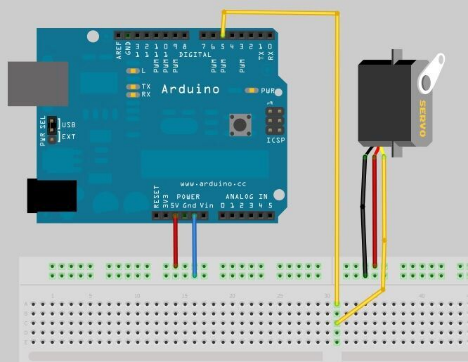
digitalWrite(13, 1); // en alto el pin 13 si la distancia es menor a 10cm

Serial.println("Alarma......."); // envia la palabra Alarma por el puerto serial

}

delay(400); // espera 400ms para que se logre ver la distancia en la consola

}

Mover Servo de extemo a extremo

// Mover servo de extremo a extremo con parada en el centro

// Descargar el programa y plano en www.elprofegarcia.com

#include <Servo.h> // Incluye la libreria Servo

Servo servo1; // Crea el objeto servo1 con las caracteristicas de Servo

void setup()

{

servo1.attach(5,600,1500); // Asocia el servo1 al pin 3, define el min y max del ancho del pulso

} // eso depende del fabricante del servo

void loop()

{

servo1.write(0); // Gira el servo a 0 grados

delay(700); // Espera 700 mili segundos a que el servo llegue a la posicion

servo1.write(90); // Gira el servo a 90 grados

delay(700);

servo1.write(180); //Gira el servo a 180 grados

delay(700);

}

Giro de servo en carro

// Mover servo de derecha a izquierda al oprimir el pulsador de cada lado

// el servo regresa automaticamente al centro (Direccion de Carro)

// Descargar el programa y plano en www.elprofegarcia.com

#include <Servo.h> // Incluye la libreria Servo

Servo servo1; // Crea el objeto servo1 con las caracteristicas de Servo

int pulsador\_der = 1; // Fijamos el pulsador de la derecha en 0

int pulsador\_izq = 1; // Fijamos el pulsador de la izquierda en 0

void setup()

{

servo1.attach(5,600,1500); // Asocia el servo1 al pin 3, define el min y max del ancho del pulso

// eso depende del fabricante del servo

pinMode(2, INPUT); // Configuramos el pin 2 como entrada

pinMode(3, INPUT); // Configuramos el pin 3 como entrada

}

void loop()

{

pulsador\_der = digitalRead(2); //lee el valor del pin 2 si esta en alto o bajo

pulsador\_izq = digitalRead(3); //lee el valor del pin 3 si esta en alto o bajo

if (pulsador\_der == LOW) { // si se oprimio el plsador de la derechao

servo1.write(180); // Gira el servo a 1800 grados

delay(15); // Espera 15 mili segundos a que el servo llegue a la posicion

}

if (pulsador\_izq == LOW) { // si se oprimio el pulsador de la izquierda

servo1.write(0); // Gira el servo a 0 grados

delay(15); // Espera 15 mili segundos a que el servo llegue a la posicion

}

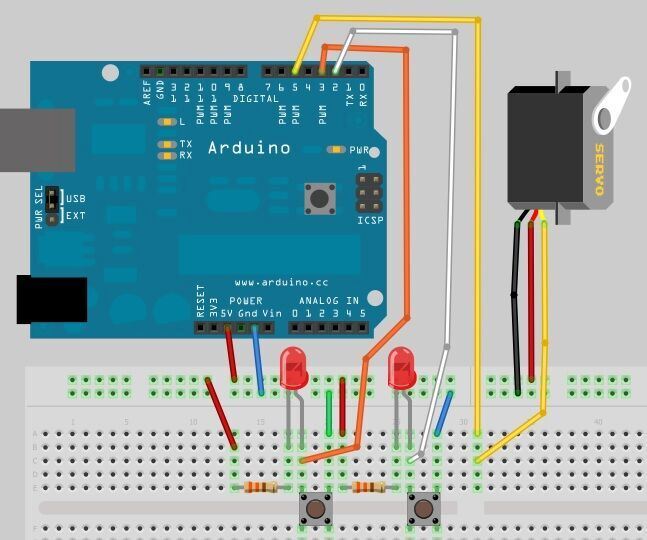
if(pulsador\_izq == HIGH && pulsador\_der == HIGH){ // Si se dejan de oprimir los dos pulsadores

servo1.write(90); // Gira el servo a 90 grados

delay(15); // Espera 15 mili segundos a que el servo llegue a la posicion

}

}



**//Luces estroboscopicas con variacion de tiempo**

//Luces estroboscopicas con variacion de tiempo

// COPIE ESTE PROGRAMA AL IDE DE ARDUINO

//salidas digitales 8 y 7 , entrada pin 2 para variar el tiempo

//descargar programa y plano electrico en www.elprofegarcia.com

int ledazul = 8;

int ledrojo = 7;

int entrada = 2;

int pulsador = 0;

int tiempo = 30;

void setup() {

pinMode(ledazul, OUTPUT);

pinMode(ledrojo, OUTPUT);

pinMode(entrada, INPUT);

}

void loop() {

digitalWrite(ledazul, HIGH); //prende led azul

delay(tiempo); // espera un tiempo de 200 o 30ms

digitalWrite(ledazul, LOW); //apaga el led azul

delay(tiempo); // espera un tiempo de 200 o 30ms

digitalWrite(ledazul, HIGH); //prende led azul

delay(tiempo); // espera un tiempo de 200 o 30ms

digitalWrite(ledazul, LOW); //apaga el led azul

delay(150); // espera un tiempo de 150ms

digitalWrite(ledrojo, HIGH); //prende led rojo

delay(tiempo); // espera un tiempo de 200 o 30 mili segundos

digitalWrite(ledrojo, LOW); //apaga el led rojo

delay(tiempo); // espera un tiempo de 200 o 30 mili segundos

digitalWrite(ledrojo, HIGH); //prende led rojo

delay(tiempo); // espera un tiempo de 200 o 30 mili segundos

digitalWrite(ledrojo, LOW); //apaga el led rojo

delay(150); // espera un tiempo de 150 mili segundos

pulsador = digitalRead(entrada); //lee el valor del pin 2 si esta en alto o bajo

if (pulsador == HIGH) { // si esta en alto el tiempo es de 30ms de lo contrario es 200ms

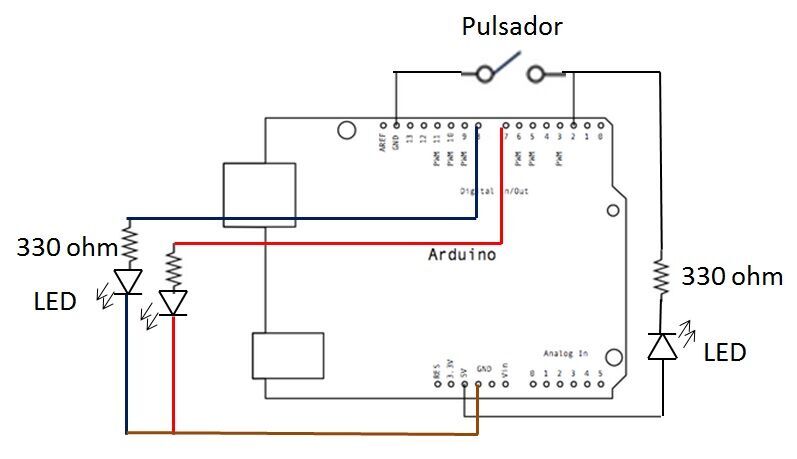
tiempo=30;

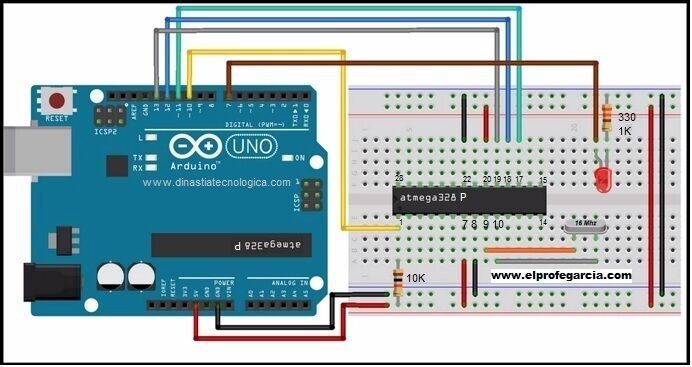
}

else {

tiempo=200;

}

 }

Bootloader Arduino Plano

// ArduinoISP version 04m3

// Copyright (c) 2008-2011 Randall Bohn

// If you require a license, see

// http://www.opensource.org/licenses/bsd-license.php

//

// This sketch turns the Arduino into a AVRISP

// using the following arduino pins:

//

// pin name: not-mega: mega(1280 and 2560)

// slave reset: 10: 53

// MOSI: 11: 51

// MISO: 12: 50

// SCK: 13: 52

//

// Put an LED (with resistor) on the following pins:

// 9: Heartbeat - shows the programmer is running

// 8: Error - Lights up if something goes wrong (use red if that makes sense)

// 7: Programming - In communication with the slave

//

// 23 July 2011 Randall Bohn

// -Address Arduino issue 509 :: Portability of ArduinoISP

// http://code.google.com/p/arduino/issues/detail?id=509

//

// October 2010 by Randall Bohn

// - Write to EEPROM > 256 bytes

// - Better use of LEDs:

// -- Flash LED\_PMODE on each flash commit

// -- Flash LED\_PMODE while writing EEPROM (both give visual feedback of writing progress)

// - Light LED\_ERR whenever we hit a STK\_NOSYNC. Turn it off when back in sync.

// - Use pins\_arduino.h (should also work on Arduino Mega)

//

// October 2009 by David A. Mellis

// - Added support for the read signature command

//

// February 2009 by Randall Bohn

// - Added support for writing to EEPROM (what took so long?)

// Windows users should consider WinAVR's avrdude instead of the

// avrdude included with Arduino software.

//

// January 2008 by Randall Bohn

// - Thanks to Amplificar for helping me with the STK500 protocol

// - The AVRISP/STK500 (mk I) protocol is used in the arduino bootloader

// - The SPI functions herein were developed for the AVR910\_ARD programmer

// - More information at http://code.google.com/p/mega-isp

#include "pins\_arduino.h"

#define RESET SS

#define LED\_HB 9

#define LED\_ERR 8

#define LED\_PMODE 7

#define PROG\_FLICKER true

#define HWVER 2

#define SWMAJ 1

#define SWMIN 18

// STK Definitions

#define STK\_OK 0x10

#define STK\_FAILED 0x11

#define STK\_UNKNOWN 0x12

#define STK\_INSYNC 0x14

#define STK\_NOSYNC 0x15

#define CRC\_EOP 0x20 //ok it is a space...

void pulse(int pin, int times);

void setup() {

Serial.begin(19200);

pinMode(LED\_PMODE, OUTPUT);

pulse(LED\_PMODE, 2);

pinMode(LED\_ERR, OUTPUT);

pulse(LED\_ERR, 2);

pinMode(LED\_HB, OUTPUT);

pulse(LED\_HB, 2);

}

int error=0;

int pmode=0;

// address for reading and writing, set by 'U' command

int here;

uint8\_t buff[256]; // global block storage

#define beget16(addr) (\*addr \* 256 + \*(addr+1) )

typedef struct param {

uint8\_t devicecode;

uint8\_t revision;

uint8\_t progtype;

uint8\_t parmode;

uint8\_t polling;

uint8\_t selftimed;

uint8\_t lockbytes;

uint8\_t fusebytes;

int flashpoll;

int eeprompoll;

int pagesize;

int eepromsize;

int flashsize;

}

parameter;

parameter param;

// this provides a heartbeat on pin 9, so you can tell the software is running.

uint8\_t hbval=128;

int8\_t hbdelta=8;

void heartbeat() {

if (hbval > 192) hbdelta = -hbdelta;

if (hbval < 32) hbdelta = -hbdelta;

hbval += hbdelta;

analogWrite(LED\_HB, hbval);

delay(20);

}

void loop(void) {

// is pmode active?

if (pmode) digitalWrite(LED\_PMODE, HIGH);

else digitalWrite(LED\_PMODE, LOW);

// is there an error?

if (error) digitalWrite(LED\_ERR, HIGH);

else digitalWrite(LED\_ERR, LOW);

// light the heartbeat LED

heartbeat();

if (Serial.available()) {

avrisp();

}

}

uint8\_t getch() {

while(!Serial.available());

return Serial.read();

}

void fill(int n) {

for (int x = 0; x < n; x++) {

buff[x] = getch();

}

}

#define PTIME 30

void pulse(int pin, int times) {

do {

digitalWrite(pin, HIGH);

delay(PTIME);

digitalWrite(pin, LOW);

delay(PTIME);

}

while (times--);

}

void prog\_lamp(int state) {

if (PROG\_FLICKER)

digitalWrite(LED\_PMODE, state);

}

void spi\_init() {

uint8\_t x;

SPCR = 0x53;

x=SPSR;

x=SPDR;

}

void spi\_wait() {

do {

}

while (!(SPSR & (1 << SPIF)));

}

uint8\_t spi\_send(uint8\_t b) {

uint8\_t reply;

SPDR=b;

spi\_wait();

reply = SPDR;

return reply;

}

uint8\_t spi\_transaction(uint8\_t a, uint8\_t b, uint8\_t c, uint8\_t d) {

uint8\_t n;

spi\_send(a);

n=spi\_send(b);

//if (n != a) error = -1;

n=spi\_send(c);

return spi\_send(d);

}

void empty\_reply() {

if (CRC\_EOP == getch()) {

Serial.print((char)STK\_INSYNC);

Serial.print((char)STK\_OK);

}

else {

error++;

Serial.print((char)STK\_NOSYNC);

}

}

void breply(uint8\_t b) {

if (CRC\_EOP == getch()) {

Serial.print((char)STK\_INSYNC);

Serial.print((char)b);

Serial.print((char)STK\_OK);

}

else {

error++;

Serial.print((char)STK\_NOSYNC);

}

}

void get\_version(uint8\_t c) {

switch(c) {

case 0x80:

breply(HWVER);

break;

case 0x81:

breply(SWMAJ);

break;

case 0x82:

breply(SWMIN);

break;

case 0x93:

breply('S'); // serial programmer

break;

default:

breply(0);

}

}

void set\_parameters() {

// call this after reading paramter packet into buff[]

param.devicecode = buff[0];

param.revision = buff[1];

param.progtype = buff[2];

param.parmode = buff[3];

param.polling = buff[4];

param.selftimed = buff[5];

param.lockbytes = buff[6];

param.fusebytes = buff[7];

param.flashpoll = buff[8];

// ignore buff[9] (= buff[8])

// following are 16 bits (big endian)

param.eeprompoll = beget16(&buff[10]);

param.pagesize = beget16(&buff[12]);

param.eepromsize = beget16(&buff[14]);

// 32 bits flashsize (big endian)

param.flashsize = buff[16] \* 0x01000000

+ buff[17] \* 0x00010000

+ buff[18] \* 0x00000100

+ buff[19];

}

void start\_pmode() {

spi\_init();

// following delays may not work on all targets...

pinMode(RESET, OUTPUT);

digitalWrite(RESET, HIGH);

pinMode(SCK, OUTPUT);

digitalWrite(SCK, LOW);

delay(50);

digitalWrite(RESET, LOW);

delay(50);

pinMode(MISO, INPUT);

pinMode(MOSI, OUTPUT);

spi\_transaction(0xAC, 0x53, 0x00, 0x00);

pmode = 1;

}

void end\_pmode() {

pinMode(MISO, INPUT);

pinMode(MOSI, INPUT);

pinMode(SCK, INPUT);

pinMode(RESET, INPUT);

pmode = 0;

}

void universal() {

int w;

uint8\_t ch;

fill(4);

ch = spi\_transaction(buff[0], buff[1], buff[2], buff[3]);

breply(ch);

}

void flash(uint8\_t hilo, int addr, uint8\_t data) {

spi\_transaction(0x40+8\*hilo,

addr>>8 & 0xFF,

addr & 0xFF,

data);

}

void commit(int addr) {

if (PROG\_FLICKER) prog\_lamp(LOW);

spi\_transaction(0x4C, (addr >> 8) & 0xFF, addr & 0xFF, 0);

if (PROG\_FLICKER) {

delay(PTIME);

prog\_lamp(HIGH);

}

}

//#define \_current\_page(x) (here & 0xFFFFE0)

int current\_page(int addr) {

if (param.pagesize == 32) return here & 0xFFFFFFF0;

if (param.pagesize == 64) return here & 0xFFFFFFE0;

if (param.pagesize == 128) return here & 0xFFFFFFC0;

if (param.pagesize == 256) return here & 0xFFFFFF80;

return here;

}

void write\_flash(int length) {

fill(length);

if (CRC\_EOP == getch()) {

Serial.print((char) STK\_INSYNC);

Serial.print((char) write\_flash\_pages(length));

}

else {

error++;

Serial.print((char) STK\_NOSYNC);

}

}

uint8\_t write\_flash\_pages(int length) {

int x = 0;

int page = current\_page(here);

while (x < length) {

if (page != current\_page(here)) {

commit(page);

page = current\_page(here);

}

flash(LOW, here, buff[x++]);

flash(HIGH, here, buff[x++]);

here++;

}

commit(page);

return STK\_OK;

}

#define EECHUNK (32)

uint8\_t write\_eeprom(int length) {

// here is a word address, get the byte address

int start = here \* 2;

int remaining = length;

if (length > param.eepromsize) {

error++;

return STK\_FAILED;

}

while (remaining > EECHUNK) {

write\_eeprom\_chunk(start, EECHUNK);

start += EECHUNK;

remaining -= EECHUNK;

}

write\_eeprom\_chunk(start, remaining);

return STK\_OK;

}

// write (length) bytes, (start) is a byte address

uint8\_t write\_eeprom\_chunk(int start, int length) {

// this writes byte-by-byte,

// page writing may be faster (4 bytes at a time)

fill(length);

prog\_lamp(LOW);

for (int x = 0; x < length; x++) {

int addr = start+x;

spi\_transaction(0xC0, (addr>>8) & 0xFF, addr & 0xFF, buff[x]);

delay(45);

}

prog\_lamp(HIGH);

return STK\_OK;

}

void program\_page() {

char result = (char) STK\_FAILED;

int length = 256 \* getch();

length += getch();

char memtype = getch();

// flash memory @here, (length) bytes

if (memtype == 'F') {

write\_flash(length);

return;

}

if (memtype == 'E') {

result = (char)write\_eeprom(length);

if (CRC\_EOP == getch()) {

Serial.print((char) STK\_INSYNC);

Serial.print(result);

}

else {

error++;

Serial.print((char) STK\_NOSYNC);

}

return;

}

Serial.print((char)STK\_FAILED);

return;

}

uint8\_t flash\_read(uint8\_t hilo, int addr) {

return spi\_transaction(0x20 + hilo \* 8,

(addr >> 8) & 0xFF,

addr & 0xFF,

0);

}

char flash\_read\_page(int length) {

for (int x = 0; x < length; x+=2) {

uint8\_t low = flash\_read(LOW, here);

Serial.print((char) low);

uint8\_t high = flash\_read(HIGH, here);

Serial.print((char) high);

here++;

}

return STK\_OK;

}

char eeprom\_read\_page(int length) {

// here again we have a word address

int start = here \* 2;

for (int x = 0; x < length; x++) {

int addr = start + x;

uint8\_t ee = spi\_transaction(0xA0, (addr >> 8) & 0xFF, addr & 0xFF, 0xFF);

Serial.print((char) ee);

}

return STK\_OK;

}

void read\_page() {

char result = (char)STK\_FAILED;

int length = 256 \* getch();

length += getch();

char memtype = getch();

if (CRC\_EOP != getch()) {

error++;

Serial.print((char) STK\_NOSYNC);

return;

}

Serial.print((char) STK\_INSYNC);

if (memtype == 'F') result = flash\_read\_page(length);

if (memtype == 'E') result = eeprom\_read\_page(length);

Serial.print(result);

return;

}

void read\_signature() {

if (CRC\_EOP != getch()) {

error++;

Serial.print((char) STK\_NOSYNC);

return;

}

Serial.print((char) STK\_INSYNC);

uint8\_t high = spi\_transaction(0x30, 0x00, 0x00, 0x00);

Serial.print((char) high);

uint8\_t middle = spi\_transaction(0x30, 0x00, 0x01, 0x00);

Serial.print((char) middle);

uint8\_t low = spi\_transaction(0x30, 0x00, 0x02, 0x00);

Serial.print((char) low);

Serial.print((char) STK\_OK);

}

//////////////////////////////////////////

//////////////////////////////////////////

////////////////////////////////////

////////////////////////////////////

int avrisp() {

uint8\_t data, low, high;

uint8\_t ch = getch();

switch (ch) {

case '0': // signon

error = 0;

empty\_reply();

break;

case '1':

if (getch() == CRC\_EOP) {

Serial.print((char) STK\_INSYNC);

Serial.print("AVR ISP");

Serial.print((char) STK\_OK);

}

break;

case 'A':

get\_version(getch());

break;

case 'B':

fill(20);

set\_parameters();

empty\_reply();

break;

case 'E': // extended parameters - ignore for now

fill(5);

empty\_reply();

break;

case 'P':

start\_pmode();

empty\_reply();

break;

case 'U': // set address (word)

here = getch();

here += 256 \* getch();

empty\_reply();

break;

case 0x60: //STK\_PROG\_FLASH

low = getch();

high = getch();

empty\_reply();

break;

case 0x61: //STK\_PROG\_DATA

data = getch();

empty\_reply();

break;

case 0x64: //STK\_PROG\_PAGE

program\_page();

break;

case 0x74: //STK\_READ\_PAGE 't'

read\_page();

break;

case 'V': //0x56

universal();

break;

case 'Q': //0x51

error=0;

end\_pmode();

empty\_reply();

break;

case 0x75: //STK\_READ\_SIGN 'u'

read\_signature();

break;

// expecting a command, not CRC\_EOP

// this is how we can get back in sync

case CRC\_EOP:

error++;

Serial.print((char) STK\_NOSYNC);

break;

// anything else we will return STK\_UNKNOWN

default:

error++;

if (CRC\_EOP == getch())

Serial.print((char)STK\_UNKNOWN);

else

Serial.print((char)STK\_NOSYNC);

}

}