Anexo de casos de pruebas

# Servomotor SG90

|  |  |
| --- | --- |
| **Caso de prueba** | Probar el funcionamiento del servomotor SG90 |
| **Identificador caso de prueba/s** | SG90-01-funcionamiento |
| **Función probar** | Funcionamiento del servomotor SG90 |
| **Objetivo** | Determinar el funcionamiento del servomotor |
| **Descripción** | Se desea conectar el servomotor SG90 a un Arduino UNO para determinar su correcto funcionamiento y ángulos de rotación con la precisión de 1° cada 20 ms (de fabrica) |
| **Criterios de éxito** | Funcionamiento correcto del servomotor en sus posibles ángulos de giro (90° a - 90°) con la precisión deseada |
| **Criterios de falla** | No alcanzar ángulos de giros correctos, fallas en conexiones |
| **Precondiciones** | Probar sin obstruir el servomotor con objetos |
| **Necesidades para el caso de prueba** | Módulo Arduino UNO  SG90 |
| **Autor** | Schlapp-Mansilla |
| **Fecha de creación** | 25-04-2017 |
| **Resultados** | [1] Se obtienen los ángulos de giros con la precisión correspondiente |
| **Código fuente/s** | [1] sg90-01-funcionamiento.ino |
| **Imágenes** |  |
| **Sketch** |  |

# Código sg90-01-funcionamiento

#include <Servo.h>

Servo myservo; // create servo object to control a servo

// twelve servo objects can be created on most boards

int pos = 0; // variable to store the servo position

void setup()

{

myservo.attach(9); // attaches the servo on pin 9 to the servo object

}

void loop()

{

for(pos = 0; pos <= -180; pos -= 1) // goes from 0 degrees to 180 degrees

{ // in steps of 1 degree

myservo.write(pos); // tell servo to go to position in variable 'pos'

delay(15); // waits 15ms for the servo to reach the position

}

}

# Pruebas en el sensor de Monóxido de Carbono

|  |  |
| --- | --- |
| **Caso de prueba** | Probar la funcionalidad del sensor de monóxido de carbono MQ7 |
| **Identificador caso de prueba/s** | MQ7-01-funcionamiento |
| **Función probar** | Funcionamiento del sensor MQ7 |
| **Objetivo** | Determinar el funcionamiento correcto del sensor |
| **Descripción** | Se desea conectar el sensor de monóxido de carbono MQ7 con un Arduino UNO para verificar su correcta detección del gas CO |
| **Criterios de éxito** | Obtener la correcta existencia, o no, de gas CO en un ambiente determinado |
| **Criterios de falla** | No obtener la correcta existencia, o no, de gas CO |
| **Precondiciones** | Testear en entornos donde se esté seguro que los niveles de CO sean bajos o inexistentes  Testear en entornos donde se esté seguro que existan al menos pocos niveles de CO |
| **Necesidades para el caso de prueba** | Módulo Arduino UNO  MQ7  Cables Hembra-Macho (x3) |
| **Autor** | Schlapp-Mansilla |
| **Fecha de creación** | 25-04-2017 |
| **Resultados** | [1] Se obtuvieron niveles de CO esperados según los ambientes testeados. |
| **Código fuente/s** | [1]MQ7-01-funcionamientoi.ino |
| **Imágenes** |  |
| **Sketch** |  |

# Código MQ7-01-funcionamiento

void setup() {

Serial.begin(9600);

}

void loop() {

int adc\_MQ = analogRead(A0); //Lemos la salida analógica del MQ

float voltaje = adc\_MQ \* (5.0 / 1023.0); //Convertimos la lectura en un valor de voltaje

Serial.print("adc:");

Serial.print(adc\_MQ);

Serial.print(" voltaje:");

Serial.println(voltaje);

delay(100);

}

# Caso de prueba N 1 Módulo WIFI ESP8266 Velocidad

|  |  |
| --- | --- |
| **Caso de prueba** | Probar la velocidad del módulo Wifi |
| **Identificador caso de prueba/s** | WifiESP8266-01-pruebaVelocidad  WifiESP8266-02-pruebaVelocidad  WifiESP8266-03-pruebaVelocidad  WifiESP8266-04-pruebaVelocidad  WifiESP8266-05-pruebaVelocidad |
| **Función probar** | Comunicación por Wifi |
| **Objetivo** | Determinar la velocidad máxima de transferencia |
| **Descripción** | Se desea verificar la velocidad de conectividad que se puede alcanzar entre una computadora con Wifi y el Arduino conectado al ESP8266 |
| **Criterios de éxito** | Alcanzar una velocidad que permita transmitir 10 fps con un tamaño de 300kb por segundo, mínimamente |
| **Criterios de falla** | No alcanzar la velocidad requerida de fps |
| **Precondiciones** | Testear un entorno sin obstáculos y línea visual.  Establecer la mayor velocidad posible de paquetes de transmisión  [1] ESP8266 a 115200 baudios  [2] ESP8266 a 921600 baudios  [3] ESP8266 a 2500000 baudios  [4] ESP8266 a 5000000 baudios  [5] ESP8266 a 4500000 baudios |
| **Necesidades para el caso de prueba** | Módulo Arduino UNO  ESP8266  Cables Hembra-Macho (x5) |
| **Autor** | Schlapp-Mansilla |
| **Fecha de creación** | 28-3-2017 |
| **Resultados** | [1]Se consigue una velocidad de 10kb/sg. Falla la prueba.  [2]Se consigue una velocidad de 30kb/sg. Falla la prueba.  [3]Se consigue una velocidad de 54kb/sg. Falla la prueba.  [4] No se puede cumplir la prueba, dado que no es posible configurar la velocidad  [5]Se consigue una velocidad de 56kb/sg. Falla la prueba. |
| **Código fuente/s** | [1]pruebaVelocidad-configuraciónWifi.ino  [2]pruebaVelocidad2-configuraciónWifi.ino  [3]pruebaVelocidad3-configuraciónWifi.ino  [4]pruebaVelocidad4-configuraciónWifi.ino  [5]pruebaVelocidad5-configuraciónWifi.ino |
| **Imágenes** |  |
| **Sketch** |  |

# Caso de prueba N 2 Módulo WIFI ESP8266 Velocidad

|  |  |
| --- | --- |
| **Caso de prueba** | Probar la velocidad del módulo Wifi |
| **Identificador caso de prueba/s** | WifiESP8266-01-pruebaVelocidad  WifiESP8266-02-pruebaVelocidad  WifiESP8266-03-pruebaVelocidad  WifiESP8266-04-pruebaVelocidad  WifiESP8266-05-pruebaVelocidad |
| **Función probar** | Comunicación por Wifi |
| **Objetivo** | Determinar la velocidad máxima de transferencia |
| **Descripción** | Se desea verificar la velocidad de conectividad que se puede alcanzar entre una computadora con Wifi y el Arduino conectado al ESP8266 |
| **Criterios de éxito** | Alcanzar una velocidad que permita transmitir 10 fps con un tamaño de 300kb por segundo, mínimamente |
| **Criterios de falla** | No alcanzar la velocidad requerida de fps |
| **Precondiciones** | Testear un entorno sin obstáculos y línea visual.  Establecer la mayor velocidad posible de paquetes de transmisión  [1] ESP8266 a 115200 baudios  [2] ESP8266 a 921600 baudios  [3] ESP8266 a 2500000 baudios  [4] ESP8266 a 5000000 baudios  [5] ESP8266 a 4500000 baudios |
| **Necesidades para el caso de prueba** | Módulo arduino UNO  ESP8266  Cables Hembra-Macho (x5) |
| **Autor** | Schlapp-Mansilla |
| **Fecha de creación** | 28-3-2017 |
| **Resultados** | [1]Se consigue una velocidad de 10kb/sg. Falla la prueba.  [2]Se consigue una velocidad de 30kb/sg. Falla la prueba.  [3]Se consigue una velocidad de 54kb/sg. Falla la prueba.  [4] No se puede cumplir la prueba, dado que no es posible configurar la velocidad  [5]Se consigue una velocidad de 56kb/sg. Falla la prueba. |
| **Código fuente/s** | [1]pruebaVelocidad-configuraciónWifi.ino  [2]pruebaVelocidad2-configuraciónWifi.ino  [3]pruebaVelocidad3-configuraciónWifi.ino  [4]pruebaVelocidad4-configuraciónWifi.ino  [5]pruebaVelocidad5-configuraciónWifi.ino |
| **Imágenes** |  |
| **Sketch** |  |

# Caso de prueba Módulo WIFI ESP8266 Velocidad y configuración AP

|  |  |
| --- | --- |
| **Caso de prueba** | Probar la velocidad del módulo Wifi |
| **Identificador caso de prueba/s** | WifiESP8266-01-ComandosAt-configuracionWifi |
| **Función probar** | Configurar módulo ESP8266 modo AP |
| **Objetivo** | Configurar el módulo ESP8266 para conocer la mayor velocidad alcanzable |
| **Descripción** | Se desea configurar el módulo como modo AP, con ssid:”SAR” sin contraseña y sin codificación. Aceptando 4 clientes simultáneos. Activando servidor DHCP. Habilitando el puerto 80 para el envío de caracteres entre PC<->Arduino a través de Putty. Comprobar los baudios, mínimos y máximos, posibles dentro del rango del Serial y Wifi |
| **Criterios de éxito** | Lograr configuración con los cambios solicitados en la descripción |
| **Criterios de falla** | No lograr la configuración deseada |
| **Precondiciones** | Testear un entorno sin obstáculos y línea visual.  Actualizar el firmware del módulo a su última versión |
| **Necesidades para el caso de prueba** | Módulo arduino UNO  ESP8266  Cables Hembra-Macho (x5)  Un dispositivo con terminal (Putty) para conectarse en modo RAW a la ip proporcionada por el ESP8266 |
| **Autor** | Schlapp-Mansilla |
| **Fecha de creación** | 28-3-2017 |
| **Resultados** | La configuración es posible, pero con errores en los comandos AT. El rango en baudios permitido del Serial[9600 - 115200] el más efectivo es el 19200  El rango en baudios permitido del módulo para transmisión es [9600 - 921600] teórico. En la práctica fue posible llevarlo hasta 4.500.000 |
| **Código fuente/s** | comandosAT-configuracionWifi.ino |
| **Imágenes** |  |
| **Sketch** |  |

# Código comandosAT-configuracionWIfi.ino

#include <SoftwareSerial.h>

SoftwareSerial ESP(9, 10); // RX | TX

/\*

Enviar comando al esp8266 y verificar la respuesta del módulo, todo esto dentro del tiempo timeout

\*/

void sendData(String comando, const int timeout)

{

long int time = millis(); // medir el tiempo actual para verificar timeout

ESP.print(comando); // enviar el comando al ESP8266

while( (time+timeout) > millis()) //mientras no haya timeout

{

while(ESP.available()) //mientras haya datos por leer

{

// Leer los datos disponibles

char c = ESP.read(); // leer el siguiente caracter

Serial.print(c);

}

}

return;

}

void setup()

{ Serial.begin(9600);

ESP.begin(19200);

sendData("AT+CIPSTART='UDP','192.168.4.2',52485",1000);

sendData("AT+CIPSTATUS",1000);

{ Serial.begin(19200);

ESP.begin(19200);

sendData("AT+CIPMUX=1\r\n",1000); // configurar para multiples conexiones

sendData("AT+CIPSERVER=1,80\r\n",1000);// Configurar el servidor en el puerto 80

}

void loop(){

String B= "." ;

if (ESP.available())

{ char c = ESP.read() ;

Serial.print(c);

}

if (Serial.available())

{ char c = Serial.read();

ESP.print(c);

}

}

# Caso de prueba N 3 Módulo WIFI ESP8266 Velocidad

|  |  |
| --- | --- |
| **Caso de prueba** | Probar la velocidad del módulo Wifi |
| **Identificador caso de prueba/s** | WifiESP8266-02-Pruebas-configuracionWifi |
| **Función probar** | Configurar módulo ESP8266 modo SOF AP |
| **Objetivo** | Configurar el módulo ESP8266 para conocer la mayor velocidad alcanzable |
| **Descripción** | Se desea configurar el módulo como modo AP, con ssid:”SAR” sin contraseña y sin codificación. Activando servidor DHCP. Habilitando el puerto para UDP y realizar el envío de caracteres entre PC<->Arduino a través de PacketSender. Comprobar los baudios, mínimos y máximos, posibles dentro del rango del Serial y Wifi y los distintos Buffers. |
| **Criterios de éxito** | Lograr configuración con los cambios solicitados en la descripción |
| **Criterios de falla** | No lograr la configuración deseada |
| **Precondiciones** | Testear un entorno sin obstáculos y línea visual. |
| **Necesidades para el caso de prueba** | Módulo Arduino UNO  ESP8266  Cables Hembra-Macho (x5)  Un dispositivo con PacketSender para generar un servidor UDP y recibir paquetes proporcionados por el ESP8266 |
| **Autor** | Mansilla |
| **Fecha de creación** | 30-3-2017 |
| **Resultados** |  |
| **Código fuente/s** | pruebaVelocidad6-configuracionWifi.ino |
| **Imágenes** |  |
| **Sketch** |  |
| **Prueba1 - Buffer 64bytes**  **PacketSender recibe 98 paquetes**  **CIOBAUD=115200** |  |
| **Prueba 2 - Buffer 8 bytes**  **PacketSender recibe 81 paquetes**  **CIOBAUD=115200** |  |
| **Prueba 3 - Buffer 128 bytes**  **PacketSender**  **recibe 78 paquetes**  **CIOBAUD=115200** |  |
| **Prueba 4 - Buffer**  **256 bytes**  **PacketSender**  **110 paquetes}CIOBAUD=115200** |  |
| **Prueba 5**  **Buffer 512 bytes**  **PacketSender 107**  **CIOBAUD=115200** |  |
| **Prueba 6**  **Buffer 1024 bytes**  **PacketSender 48**  **ERROR**  **CIOBAUD=115200** |  |
| **Prueba 7**  **Buffer 512 bytes**  **PacketSender 6**  **CIOBAUD=9600** |  |
| **Prueba 8**  **Buffer 512 bytes**  **PacketSender 140**  **CIOBAUD=250000** |  |
| **Prueba 9**  **Buffer 512 bytes**  **CIOBAUD=500000** |  |
| **Prueba 10**  **Buffer 512**  **CIOBAUD= 4000000** |  |
| **Prueba 11**  **4608000 baudios Buffer 512 bytes** |  |
| **Prueba 12**  **4608000 baudios y 256 bytes buffer** |  |
| **Prueba 13**  **4608000 baudios y 768 bytes de buffer** |  |

# Código pruebaVelocidad6-configuracionWifi

#include <SoftwareSerial.h>

#define MAX 128

#define VELORIGINAL 115200

#define VELNUEVA 19200

//velocidad maxima 4608000

/\*\*

\* Configuramos el BT a 4500000 y conseguimos una velocidad de 55 KB/sg

\*/

SoftwareSerial ESP(3,2); // RX | TX

/\*

Enviar comando al esp8266 y verificar la respuesta del módulo, todo esto dentro del tiempo timeout

\*/

void sendData(String comando, const int timeout)

{

long int time = millis(); // medir el tiempo actual para verificar timeout

ESP.print(comando); // enviar el comando al ESP8266

while( (time+timeout) > millis()) //mientras no haya timeout

{

while(ESP.available()) //mientras haya datos por leer

{

// Leer los datos disponibles

char c = ESP.read(); // leer el siguiente caracter

Serial.print(c);

}

}

return;

}

//Funcion para llenar un buffer con 1024 elementos

void armarBuffer(char buf[], int inicio, int fin){

for(int i=inicio; i<=fin; i++){

buf[i]='1';

}

}

char frame[MAX]; //En 2048 se queda con problemas Arduino UNO, por quedarse sin espacio

//Se configura el serial para imprimir las opciones

//con un buffer de 512, y misma velo va a 20kb/sg

//con un buffer de 256, misma velo va a 25-38kb/sg

//Con un buffer de 128, la misma velo va a 45-66kb/sg

//TODO hay que testear esto con el ESP, a otra velocidad

//Probando sin CIOMUX=0, CIPMODE=1, CIPSERVER=0, TCP rafagas de 20ms buffer 2k

void setup()

{ Serial.begin(9600);

Serial.println("Las opciones son: 1 para comenzar y 2 para finalizar");

ESP.begin(115200);

sendData("AT+CIOBAUD="+String(VELNUEVA)+"\r\n", 3000);

Serial.println("-------FIN CONFIG VELNUEVA------");

ESP.begin(VELNUEVA);

sendData("AT+CIPSTART='UDP','192.168.4.2',56011",1000);

Serial.println("-------conexion UDP------");

sendData("AT+CIPSTATUS",1000);

Serial.println("-------STATUS------");

armarBuffer(frame,0,MAX-1);

}

//Este toma de la entrada estandar un 1 para comenzar la prueba de transmitir un buffer de 1kb, en 1 sg, por medio del wifi.

//De esta forma, se puede terminar la máxima velocidad reduciendo el timeout.

long minimo=0;

long maximo=0;

long paquetesEnviados;

long tiempoInicio;

long tiempoFinal;

bool primero=true;

void loop()

{

if(Serial.available()){

//Si esta disponible leo del buffer

char opc = Serial.read();

if(opc == '1'){

bool detener=false;

tiempoInicio=millis();

paquetesEnviados=0;

while(!detener){

long tiempoAnterior=millis();

sendData("AT+CIPSEND="+String(MAX)+"\r\n",0);

ESP.print(frame);

paquetesEnviados++;

long tiempoActual = millis()-tiempoAnterior;

Serial.println("---------El tiempo de transmisión es:"+String(((1000/tiempoActual)\*MAX)/1024)+ "KB/sg");

if (primero){

primero=false;

minimo=tiempoActual;

maximo=tiempoActual;

}else{

if (minimo>tiempoActual){

minimo= tiempoActual;

}

if(maximo< tiempoActual){

maximo= tiempoActual;

}

}

opc = Serial.read();

if (opc=='2'){

detener = true;

tiempoFinal = millis();

primero=true;

Serial.println("Rango de velocidad medido con Buffer="+String(MAX)+"KB Tiempo minimo entre paquetes:"+String(minimo)+"ms<->Tiempo maximo entre paquetes:"+String(maximo)+"ms");

Serial.println("Tiempo total de prueba:"+String((tiempoFinal-tiempoInicio)/1000)+" segundos");

Serial.println("Paquetes enviados"+String(paquetesEnviados));

float paqTiem=paquetesEnviados/((tiempoFinal-tiempoInicio)/1000);

Serial.println("Media:"+String(paqTiem)+"paq/sg");

Serial.println("Transf:"+String((paqTiem\*MAX)/1024)+"KB/SG");

sendData("AT+CIOBAUD="+String(VELORIGINAL)+"\r\n", 3000);

ESP.begin(VELORIGINAL);

}

}

}

}

if(ESP.available()){ //Hay algo en el buffer del wifi

while(ESP.available()){

char c= ESP.read();

Serial.print(c);

}

}

}

# Caso de prueba Módulo GPS

|  |  |
| --- | --- |
| **Caso de prueba** | Probar la conectividad del módulo GPS |
| **Identificador caso de prueba/s** | GPS-NEO6-01-conectividad |
| **Función probar** | Recepción por GPS |
| **Objetivo** | Determinar la recepción |
| **Descripción** | Se desea conectar el módulo NEO6 con el Arduino UNO, para probar la recepción de datos desde los satélites. |
| **Criterios de éxito** | Obtener una trama correctamente |
| **Criterios de falla** | No obtener una trama |
| **Precondiciones** | Testear un entorno sin obstáculos y campo abierto |
| **Necesidades para el caso de prueba** | Módulo arduino UNO  NEO6-GPS  Cables Hembra-Macho (x4) |
| **Autor** | Schlapp-Mansilla |
| **Fecha de creación** | 25-3-2017 |
| **Resultados** | [1]Se consigue la trama con el posicionamiento correspondiente en un tiempo prudente. |
| **Código fuente/s** | [1]GPS-NEO6-01-conectividad.ino |
| **Imágenes** |  |
| **Sketch** |  |

# Código GPS-NEO6-01Conectividad

#include <SoftwareSerial.h>

SoftwareSerial gps(4,3);

char dato=' ';

void setup()

{

Serial.begin(9600);

gps.begin(9600);

}

void loop()

{

if(gps.available())

{

dato=gps.read();

Serial.print(dato);

}

}

# Caso de prueba Módulo microSD Card Adapter

|  |  |
| --- | --- |
| **Caso de prueba** | Probar el funcionamiento del microSD Card Adapter |
| **Identificador caso de prueba/s** | microSD-01-leerEscribir |
| **Función probar** | Almacenar y recuperar información en microsd de 16GB |
| **Objetivo** | Determinar velocidad de lectura y escritura desde Arduino |
| **Descripción** | Se desea almacenar y recuperar datos almacenados en una memoria microSD de 16GB conectada a un Arduino UNO. Además, tomar almacenar datos en ésta para visualizarlos en una PC |
| **Criterios de éxito** | Poder almacenar un/varios archivos/s y leerlos desde la PC |
| **Criterios de falla** | No poder almacenar y/o recuperar datos/archivos |
| **Precondiciones** | Se trabajará con el protocolo SPI |
| **Necesidades para el caso de prueba** | Módulo arduino UNO  microSD Card Adapter  Cables Hembra-Macho (x6) |
| **Autor** | Schlapp-Mansilla |
| **Fecha de creación** | 25-4-2017 |
| **Resultados** | [1]Se consigue almacenar y escribir a una microSD de 8GB |
| **Código fuente/s** | [1]microSD-01-leerEscribir.ino |
| **Imágenes** |  |
| **Sketch** |  |

# Código microSD-01-LeerEscribir

#include <SPI.h>

#include <SD.h>

File myFile;

void setup() {

// Open serial communications and wait for port to open:

Serial.begin(9600);

while (!Serial) {

; // wait for serial port to connect. Needed for native USB port only

}

Serial.print("Initializing SD card...");

if (!SD.begin(4)) {

Serial.println("initialization failed!");

return;

}

Serial.println("initialization done.");

// open the file. note that only one file can be open at a time,

// so you have to close this one before opening another.

myFile = SD.open("test.txt", FILE\_WRITE);

// if the file opened okay, write to it:

if (myFile) {

Serial.print("Writing to test.txt...");

myFile.println("testing 1, 2, 3.");

// close the file:

myFile.close();

Serial.println("done.");

} else {

// if the file didn't open, print an error:

Serial.println("error opening test.txt");

}

// re-open the file for reading:

myFile = SD.open("test.txt");

if (myFile) {

Serial.println("test.txt:");

// read from the file until there's nothing else in it:

while (myFile.available()) {

Serial.write(myFile.read());

}

// close the file:

myFile.close();

} else {

// if the file didn't open, print an error:

Serial.println("error opening test.txt");

}

}

void loop() {

// nothing happens after setup

}

# Caso de prueba Integración WIFI y Cámara

|  |  |
| --- | --- |
| **Caso de prueba** | Fase 1 - Módulo WIFI ESP8266 y Cámara OV7670 |
| **Identificador caso de prueba/s** | integración-fase1-transmisión |
| **Función probar** | Transmisión de imágenes a la PC |
| **Objetivo** | Determinar desempeño en la transmisión de imágenes y correcta comunicación entre el ESP8266 y OV7670 mediante un Arduino UNO. |
| **Descripción** | Se desea conectar el módulo ESP8266, y el OV7670 a un mismo Arduino UNO, para probar la transmisión, vía Wifi, de una imágen a la PC. |
| **Criterios de éxito** | Poder enviar al menos una imagen desde el Arduino UNO, a la PC. |
| **Criterios de falla** | Mala conexión o ensamblado, errores en transmisión |
| **Precondiciones** | Es necesario alimentar el ESP8266 por separado con 3.5V |
| **Necesidades para el caso de prueba** | Módulo Arduino UNO  OV7670  ESP8266  Cables Hembra-Macho (18 pines)  PC  Portapilas 3 x AA  Protoboard |
| **Autor** | Schlapp-Mansilla |
| **Fecha de creación** | 14-04-2017 |
| **Resultados** | [1]Falla, se supone que uno de los motivos es el alto procesamiento que efectúa el Arduino UNO, haciendo buffering de la OV7670, la cual no cuenta con chip propio. Además para optimizar la velocidad se utilizan los registros a bajo nivel del Arduino UNO, lo que genera problemas en la transmisión al ESP8266, de esta forma éste no puede cumplir la entrega de los paquetes por WIFI. |
| **Código fuente/s** | [1]integración-fase1-transmisión.ino |
| **Imágenes** |  |
| **Sketch** |  |

# Caso de prueba Cámara OV 7670

|  |  |
| --- | --- |
| **Caso de prueba** | Probar la conectividad de la Cámara OV 7670 con un Arduino Uno |
| **Identificador caso de prueba/s** | CAMOV7670-01-conectividad |
| **Función probar** | Comunicación del Arduino UNO con la Cámara |
| **Objetivo** | Determinar el funcionamiento del módulo |
| **Descripción** | Se desea conectar el módulo OV7670 por medio de un módulo Arduino UNO a la PC. |
| **Criterios de éxito** | Poder enviar al menos una imagen desde el módulo, a través del Arduino UNO, a la PC. |
| **Criterios de falla** | Mala conexión o ensamblado, errores en transmisión |
| **Precondiciones** | Testear la transferencia de imágenes con una velocidad en el puerto serie de  [1]CAM=OV7670 a 9200 baudios |
| **Necesidades para el caso de prueba** | Módulo Arduino UNO  OV7670  Cables Hembra-Macho (18 pines)  PC |
| **Autor** | Schlapp-Mansilla |
| **Fecha de creación** | 8-3-2017 |
| **Resultados** | [1]Se consigue transmitir una imagen con éxito. |
| **Código fuente/s** | [1]camaraOV7670.ino → En Arduino  BMP.java, SimpleRead.java→ En PC (Capturar y armar la imagen) |
| **Imágenes** |  |
| **Sketch** |  |

# Código OV7670

#include <stdint.h>

#include <avr/io.h>

#include <util/twi.h>

#include <util/delay.h>

#include <avr/pgmspace.h>

#include <SoftwareSerial.h>

#define MAX 8

#define VELORIGINAL 115200

#define VELNUEVA 19200

#define F\_CPU 16000000UL

#define vga 0

#define qvga 1

#define qqvga 2

#define yuv422 0

#define rgb565 1

#define bayerRGB 2

#define camAddr\_WR 0x42

#define camAddr\_RD 0x43

/\* Registers \*/

#define REG\_GAIN 0x00 /\* Gain lower 8 bits (rest in vref) \*/

#define REG\_BLUE 0x01 /\* blue gain \*/

#define REG\_RED 0x02 /\* red gain \*/

#define REG\_VREF 0x03 /\* Pieces of GAIN, VSTART, VSTOP \*/

#define REG\_COM1 0x04 /\* Control 1 \*/

#define COM1\_CCIR656 0x40 /\* CCIR656 enable \*/

#define REG\_BAVE 0x05 /\* U/B Average level \*/

#define REG\_GbAVE 0x06 /\* Y/Gb Average level \*/

#define REG\_AECHH 0x07 /\* AEC MS 5 bits \*/

#define REG\_RAVE 0x08 /\* V/R Average level \*/

#define REG\_COM2 0x09 /\* Control 2 \*/

#define COM2\_SSLEEP 0x10 /\* Soft sleep mode \*/

#define REG\_PID 0x0a /\* Product ID MSB \*/

#define REG\_VER 0x0b /\* Product ID LSB \*/

#define REG\_COM3 0x0c /\* Control 3 \*/

#define COM3\_SWAP 0x40 /\* Byte swap \*/

#define COM3\_SCALEEN 0x08 /\* Enable scaling \*/

#define COM3\_DCWEN 0x04 /\* Enable downsamp/crop/window \*/

#define REG\_COM4 0x0d /\* Control 4 \*/

#define REG\_COM5 0x0e /\* All "reserved" \*/

#define REG\_COM6 0x0f /\* Control 6 \*/

#define REG\_AECH 0x10 /\* More bits of AEC value \*/

#define REG\_CLKRC 0x11 /\* Clocl control \*/

#define CLK\_EXT 0x40 /\* Use external clock directly \*/

#define CLK\_SCALE 0x3f /\* Mask for internal clock scale \*/

#define REG\_COM7 0x12 /\* Control 7 \*/ //REG mean address.

#define COM7\_RESET 0x80 /\* Register reset \*/

#define COM7\_FMT\_MASK 0x38

#define COM7\_FMT\_VGA 0x00

#define COM7\_FMT\_CIF 0x20 /\* CIF format \*/

#define COM7\_FMT\_QVGA 0x10 /\* QVGA format \*/

#define COM7\_FMT\_QCIF 0x08 /\* QCIF format \*/

#define COM7\_RGB 0x04 /\* bits 0 and 2 - RGB format \*/

#define COM7\_YUV 0x00 /\* YUV \*/

#define COM7\_BAYER 0x01 /\* Bayer format \*/

#define COM7\_PBAYER 0x05 /\* "Processed bayer" \*/

#define REG\_COM8 0x13 /\* Control 8 \*/

#define COM8\_FASTAEC 0x80 /\* Enable fast AGC/AEC \*/

#define COM8\_AECSTEP 0x40 /\* Unlimited AEC step size \*/

#define COM8\_BFILT 0x20 /\* Band filter enable \*/

#define COM8\_AGC 0x04 /\* Auto gain enable \*/

#define COM8\_AWB 0x02 /\* White balance enable \*/

#define COM8\_AEC 0x01 /\* Auto exposure enable \*/

#define REG\_COM9 0x14 /\* Control 9- gain ceiling \*/

#define REG\_COM10 0x15 /\* Control 10 \*/

#define COM10\_HSYNC 0x40 /\* HSYNC instead of HREF \*/

#define COM10\_PCLK\_HB 0x20 /\* Suppress PCLK on horiz blank \*/

#define COM10\_HREF\_REV 0x08 /\* Reverse HREF \*/

#define COM10\_VS\_LEAD 0x04 /\* VSYNC on clock leading edge \*/

#define COM10\_VS\_NEG 0x02 /\* VSYNC negative \*/

#define COM10\_HS\_NEG 0x01 /\* HSYNC negative \*/

#define REG\_HSTART 0x17 /\* Horiz start high bits \*/

#define REG\_HSTOP 0x18 /\* Horiz stop high bits \*/

#define REG\_VSTART 0x19 /\* Vert start high bits \*/

#define REG\_VSTOP 0x1a /\* Vert stop high bits \*/

#define REG\_PSHFT 0x1b /\* Pixel delay after HREF \*/

#define REG\_MIDH 0x1c /\* Manuf. ID high \*/

#define REG\_MIDL 0x1d /\* Manuf. ID low \*/

#define REG\_MVFP 0x1e /\* Mirror / vflip \*/

#define MVFP\_MIRROR 0x20 /\* Mirror image \*/

#define MVFP\_FLIP 0x10 /\* Vertical flip \*/

#define REG\_AEW 0x24 /\* AGC upper limit \*/

#define REG\_AEB 0x25 /\* AGC lower limit \*/

#define REG\_VPT 0x26 /\* AGC/AEC fast mode op region \*/

#define REG\_HSYST 0x30 /\* HSYNC rising edge delay \*/

#define REG\_HSYEN 0x31 /\* HSYNC falling edge delay \*/

#define REG\_HREF 0x32 /\* HREF pieces \*/

#define REG\_TSLB 0x3a /\* lots of stuff \*/

#define TSLB\_YLAST 0x04 /\* UYVY or VYUY - see com13 \*/

#define REG\_COM11 0x3b /\* Control 11 \*/

#define COM11\_NIGHT 0x80 /\* NIght mode enable \*/

#define COM11\_NMFR 0x60 /\* Two bit NM frame rate \*/

#define COM11\_HZAUTO 0x10 /\* Auto detect 50/60 Hz \*/

#define COM11\_50HZ 0x08 /\* Manual 50Hz select \*/

#define COM11\_EXP 0x02

#define REG\_COM12 0x3c /\* Control 12 \*/

#define COM12\_HREF 0x80 /\* HREF always \*/

#define REG\_COM13 0x3d /\* Control 13 \*/

#define COM13\_GAMMA 0x80 /\* Gamma enable \*/

#define COM13\_UVSAT 0x40 /\* UV saturation auto adjustment \*/

#define COM13\_UVSWAP 0x01 /\* V before U - w/TSLB \*/

#define REG\_COM14 0x3e /\* Control 14 \*/

#define COM14\_DCWEN 0x10 /\* DCW/PCLK-scale enable \*/

#define REG\_EDGE 0x3f /\* Edge enhancement factor \*/

#define REG\_COM15 0x40 /\* Control 15 \*/

#define COM15\_R10F0 0x00 /\* Data range 10 to F0 \*/

#define COM15\_R01FE 0x80 /\* 01 to FE \*/

#define COM15\_R00FF 0xc0 /\* 00 to FF \*/

#define COM15\_RGB565 0x10 /\* RGB565 output \*/

#define COM15\_RGB555 0x30 /\* RGB555 output \*/

#define REG\_COM16 0x41 /\* Control 16 \*/

#define COM16\_AWBGAIN 0x08 /\* AWB gain enable \*/

#define REG\_COM17 0x42 /\* Control 17 \*/

#define COM17\_AECWIN 0xc0 /\* AEC window - must match COM4 \*/

#define COM17\_CBAR 0x08 /\* DSP Color bar \*/

/\*

\* This matrix defines how the colors are generated, must be

\* tweaked to adjust hue and saturation.

\*

\* Order: v-red, v-green, v-blue, u-red, u-green, u-blue

\* They are nine-bit signed quantities, with the sign bit

\* stored in0x58.Sign for v-red is bit 0, and up from there.

\*/

#define REG\_CMATRIX\_BASE 0x4f

#define CMATRIX\_LEN 6

#define REG\_CMATRIX\_SIGN 0x58

#define REG\_BRIGHT 0x55 /\* Brightness \*/

#define REG\_CONTRAS 0x56 /\* Contrast control \*/

#define REG\_GFIX 0x69 /\* Fix gain control \*/

#define REG\_REG76 0x76 /\* OV's name \*/

#define R76\_BLKPCOR 0x80 /\* Black pixel correction enable \*/

#define R76\_WHTPCOR 0x40 /\* White pixel correction enable \*/

#define REG\_RGB444 0x8c /\* RGB 444 control \*/

#define R444\_ENABLE 0x02 /\* Turn on RGB444, overrides 5x5 \*/

#define R444\_RGBX 0x01 /\* Empty nibble at end \*/

#define REG\_HAECC1 0x9f /\* Hist AEC/AGC control 1 \*/

#define REG\_HAECC2 0xa0 /\* Hist AEC/AGC control 2 \*/

#define REG\_BD50MAX 0xa5 /\* 50hz banding step limit \*/

#define REG\_HAECC3 0xa6 /\* Hist AEC/AGC control 3 \*/

#define REG\_HAECC4 0xa7 /\* Hist AEC/AGC control 4 \*/

#define REG\_HAECC5 0xa8 /\* Hist AEC/AGC control 5 \*/

#define REG\_HAECC6 0xa9 /\* Hist AEC/AGC control 6 \*/

#define REG\_HAECC7 0xaa /\* Hist AEC/AGC control 7 \*/

#define REG\_BD60MAX 0xab /\* 60hz banding step limit \*/

#define REG\_GAIN 0x00 /\* Gain lower 8 bits (rest in vref) \*/

#define REG\_BLUE 0x01 /\* blue gain \*/

#define REG\_RED 0x02 /\* red gain \*/

#define REG\_VREF 0x03 /\* Pieces of GAIN, VSTART, VSTOP \*/

#define REG\_COM1 0x04 /\* Control 1 \*/

#define COM1\_CCIR656 0x40 /\* CCIR656 enable \*/

#define REG\_BAVE 0x05 /\* U/B Average level \*/

#define REG\_GbAVE 0x06 /\* Y/Gb Average level \*/

#define REG\_AECHH 0x07 /\* AEC MS 5 bits \*/

#define REG\_RAVE 0x08 /\* V/R Average level \*/

#define REG\_COM2 0x09 /\* Control 2 \*/

#define COM2\_SSLEEP 0x10 /\* Soft sleep mode \*/

#define REG\_PID 0x0a /\* Product ID MSB \*/

#define REG\_VER 0x0b /\* Product ID LSB \*/

#define REG\_COM3 0x0c /\* Control 3 \*/

#define COM3\_SWAP 0x40 /\* Byte swap \*/

#define COM3\_SCALEEN 0x08 /\* Enable scaling \*/

#define COM3\_DCWEN 0x04 /\* Enable downsamp/crop/window \*/

#define REG\_COM4 0x0d /\* Control 4 \*/

#define REG\_COM5 0x0e /\* All "reserved" \*/

#define REG\_COM6 0x0f /\* Control 6 \*/

#define REG\_AECH 0x10 /\* More bits of AEC value \*/

#define REG\_CLKRC 0x11 /\* Clocl control \*/

#define CLK\_EXT 0x40 /\* Use external clock directly \*/

#define CLK\_SCALE 0x3f /\* Mask for internal clock scale \*/

#define REG\_COM7 0x12 /\* Control 7 \*/

#define COM7\_RESET 0x80 /\* Register reset \*/

#define COM7\_FMT\_MASK 0x38

#define COM7\_FMT\_VGA 0x00

#define COM7\_FMT\_CIF 0x20 /\* CIF format \*/

#define COM7\_FMT\_QVGA 0x10 /\* QVGA format \*/

#define COM7\_FMT\_QCIF 0x08 /\* QCIF format \*/

#define COM7\_RGB 0x04 /\* bits 0 and 2 - RGB format \*/

#define COM7\_YUV 0x00 /\* YUV \*/

#define COM7\_BAYER 0x01 /\* Bayer format \*/

#define COM7\_PBAYER 0x05 /\* "Processed bayer" \*/

#define REG\_COM8 0x13 /\* Control 8 \*/

#define COM8\_FASTAEC 0x80 /\* Enable fast AGC/AEC \*/

#define COM8\_AECSTEP 0x40 /\* Unlimited AEC step size \*/

#define COM8\_BFILT 0x20 /\* Band filter enable \*/

#define COM8\_AGC 0x04 /\* Auto gain enable \*/

#define COM8\_AWB 0x02 /\* White balance enable \*/

#define COM8\_AEC 0x01 /\* Auto exposure enable \*/

#define REG\_COM9 0x14 /\* Control 9- gain ceiling \*/

#define REG\_COM10 0x15 /\* Control 10 \*/

#define COM10\_HSYNC 0x40 /\* HSYNC instead of HREF \*/

#define COM10\_PCLK\_HB 0x20 /\* Suppress PCLK on horiz blank \*/

#define COM10\_HREF\_REV 0x08 /\* Reverse HREF \*/

#define COM10\_VS\_LEAD 0x04 /\* VSYNC on clock leading edge \*/

#define COM10\_VS\_NEG 0x02 /\* VSYNC negative \*/

#define COM10\_HS\_NEG 0x01 /\* HSYNC negative \*/

#define REG\_HSTART 0x17 /\* Horiz start high bits \*/

#define REG\_HSTOP 0x18 /\* Horiz stop high bits \*/

#define REG\_VSTART 0x19 /\* Vert start high bits \*/

#define REG\_VSTOP 0x1a /\* Vert stop high bits \*/

#define REG\_PSHFT 0x1b /\* Pixel delay after HREF \*/

#define REG\_MIDH 0x1c /\* Manuf. ID high \*/

#define REG\_MIDL 0x1d /\* Manuf. ID low \*/

#define REG\_MVFP 0x1e /\* Mirror / vflip \*/

#define MVFP\_MIRROR 0x20 /\* Mirror image \*/

#define MVFP\_FLIP 0x10 /\* Vertical flip \*/

#define REG\_AEW 0x24 /\* AGC upper limit \*/

#define REG\_AEB 0x25 /\* AGC lower limit \*/

#define REG\_VPT 0x26 /\* AGC/AEC fast mode op region \*/

#define REG\_HSYST 0x30 /\* HSYNC rising edge delay \*/

#define REG\_HSYEN 0x31 /\* HSYNC falling edge delay \*/

#define REG\_HREF 0x32 /\* HREF pieces \*/

#define REG\_TSLB 0x3a /\* lots of stuff \*/

#define TSLB\_YLAST 0x04 /\* UYVY or VYUY - see com13 \*/

#define REG\_COM11 0x3b /\* Control 11 \*/

#define COM11\_NIGHT 0x80 /\* NIght mode enable \*/

#define COM11\_NMFR 0x60 /\* Two bit NM frame rate \*/

#define COM11\_HZAUTO 0x10 /\* Auto detect 50/60 Hz \*/

#define COM11\_50HZ 0x08 /\* Manual 50Hz select \*/

#define COM11\_EXP 0x02

#define REG\_COM12 0x3c /\* Control 12 \*/

#define COM12\_HREF 0x80 /\* HREF always \*/

#define REG\_COM13 0x3d /\* Control 13 \*/

#define COM13\_GAMMA 0x80 /\* Gamma enable \*/

#define COM13\_UVSAT 0x40 /\* UV saturation auto adjustment \*/

#define COM13\_UVSWAP 0x01 /\* V before U - w/TSLB \*/

#define REG\_COM14 0x3e /\* Control 14 \*/

#define COM14\_DCWEN 0x10 /\* DCW/PCLK-scale enable \*/

#define REG\_EDGE 0x3f /\* Edge enhancement factor \*/

#define REG\_COM15 0x40 /\* Control 15 \*/

#define COM15\_R10F0 0x00 /\* Data range 10 to F0 \*/

#define COM15\_R01FE 0x80 /\* 01 to FE \*/

#define COM15\_R00FF 0xc0 /\* 00 to FF \*/

#define COM15\_RGB565 0x10 /\* RGB565 output \*/

#define COM15\_RGB555 0x30 /\* RGB555 output \*/

#define REG\_COM16 0x41 /\* Control 16 \*/

#define COM16\_AWBGAIN 0x08 /\* AWB gain enable \*/

#define REG\_COM17 0x42 /\* Control 17 \*/

#define COM17\_AECWIN 0xc0 /\* AEC window - must match COM4 \*/

#define COM17\_CBAR 0x08 /\* DSP Color bar \*/

#define CMATRIX\_LEN 6

#define REG\_BRIGHT 0x55 /\* Brightness \*/

#define REG\_REG76 0x76 /\* OV's name \*/

#define R76\_BLKPCOR 0x80 /\* Black pixel correction enable \*/

#define R76\_WHTPCOR 0x40 /\* White pixel correction enable \*/

#define REG\_RGB444 0x8c /\* RGB 444 control \*/

#define R444\_ENABLE 0x02 /\* Turn on RGB444, overrides 5x5 \*/

#define R444\_RGBX 0x01 /\* Empty nibble at end \*/

#define REG\_HAECC1 0x9f /\* Hist AEC/AGC control 1 \*/

#define REG\_HAECC2 0xa0 /\* Hist AEC/AGC control 2 \*/

#define REG\_BD50MAX 0xa5 /\* 50hz banding step limit \*/

#define REG\_HAECC3 0xa6 /\* Hist AEC/AGC control 3 \*/

#define REG\_HAECC4 0xa7 /\* Hist AEC/AGC control 4 \*/

#define REG\_HAECC5 0xa8 /\* Hist AEC/AGC control 5 \*/

#define REG\_HAECC6 0xa9 /\* Hist AEC/AGC control 6 \*/

#define REG\_HAECC7 0xaa /\* Hist AEC/AGC control 7 \*/

#define REG\_BD60MAX 0xab /\* 60hz banding step limit \*/

#define MTX1 0x4f /\* Matrix Coefficient 1 \*/

#define MTX2 0x50 /\* Matrix Coefficient 2 \*/

#define MTX3 0x51 /\* Matrix Coefficient 3 \*/

#define MTX4 0x52 /\* Matrix Coefficient 4 \*/

#define MTX5 0x53 /\* Matrix Coefficient 5 \*/

#define MTX6 0x54 /\* Matrix Coefficient 6 \*/

#define REG\_CONTRAS 0x56 /\* Contrast control \*/

#define MTXS 0x58 /\* Matrix Coefficient Sign \*/

#define AWBC7 0x59 /\* AWB Control 7 \*/

#define AWBC8 0x5a /\* AWB Control 8 \*/

#define AWBC9 0x5b /\* AWB Control 9 \*/

#define AWBC10 0x5c /\* AWB Control 10 \*/

#define AWBC11 0x5d /\* AWB Control 11 \*/

#define AWBC12 0x5e /\* AWB Control 12 \*/

#define REG\_GFI 0x69 /\* Fix gain control \*/

#define GGAIN 0x6a /\* G Channel AWB Gain \*/

#define DBLV 0x6b

#define AWBCTR3 0x6c /\* AWB Control 3 \*/

#define AWBCTR2 0x6d /\* AWB Control 2 \*/

#define AWBCTR1 0x6e /\* AWB Control 1 \*/

#define AWBCTR0 0x6f /\* AWB Control 0 \*/

SoftwareSerial ESP(9,10); // RX | TX

/\*

Enviar comando al esp8266 y verificar la respuesta del módulo, todo esto dentro del tiempo timeout

\*/

void sendData(String comando, const int timeout)

{

long int time = millis(); // medir el tiempo actual para verificar timeout

ESP.print(comando); // enviar el comando al ESP8266

while( (time+timeout) > millis()) //mientras no haya timeout

{

while(ESP.available()) //mientras haya datos por leer

{

// Leer los datos disponibles

char c = ESP.read(); // leer el siguiente caracter

Serial.print(c);

}

}

//Serial.println("Se ingreso el comando: " + comando);

return;

}

struct regval\_list{

uint8\_t reg\_num;

uint16\_t value;

};

const struct regval\_list qvga\_ov7670[] PROGMEM = {

{ REG\_COM14, 0x19 },

{ 0x72, 0x11 },

{ 0x73, 0xf1 },

{ REG\_HSTART, 0x16 },

{ REG\_HSTOP, 0x04 },

{ REG\_HREF, 0xa4 },

{ REG\_VSTART, 0x02 },

{ REG\_VSTOP, 0x7a },

{ REG\_VREF, 0x0a },

/\* { REG\_HSTART, 0x16 },

{ REG\_HSTOP, 0x04 },

{ REG\_HREF, 0x24 },

{ REG\_VSTART, 0x02 },

{ REG\_VSTOP, 0x7a },

{ REG\_VREF, 0x0a },\*/

{ 0xff, 0xff }, /\* END MARKER \*/

};

const struct regval\_list yuv422\_ov7670[] PROGMEM = {

{ REG\_COM7, 0x0 }, /\* Selects YUV mode \*/

{ REG\_RGB444, 0 }, /\* No RGB444 please \*/

{ REG\_COM1, 0 },

{ REG\_COM15, COM15\_R00FF },

{ REG\_COM9, 0x6A }, /\* 128x gain ceiling; 0x8 is reserved bit \*/

{ 0x4f, 0x80 }, /\* "matrix coefficient 1" \*/

{ 0x50, 0x80 }, /\* "matrix coefficient 2" \*/

{ 0x51, 0 }, /\* vb \*/

{ 0x52, 0x22 }, /\* "matrix coefficient 4" \*/

{ 0x53, 0x5e }, /\* "matrix coefficient 5" \*/

{ 0x54, 0x80 }, /\* "matrix coefficient 6" \*/

{ REG\_COM13, COM13\_UVSAT },

{ 0xff, 0xff }, /\* END MARKER \*/

};

const struct regval\_list ov7670\_default\_regs[] PROGMEM = {//from the linux driver

{ REG\_COM7, COM7\_RESET },

{ REG\_TSLB, 0x04 }, /\* OV \*/

{ REG\_COM7, 0 }, /\* VGA \*/

/\*

\* Set the hardware window. These values from OV don't entirely

\* make sense - hstop is less than hstart. But they work...

\*/

{ REG\_HSTART, 0x13 }, { REG\_HSTOP, 0x01 },

{ REG\_HREF, 0xb6 }, { REG\_VSTART, 0x02 },

{ REG\_VSTOP, 0x7a }, { REG\_VREF, 0x0a },

{ REG\_COM3, 0 }, { REG\_COM14, 0 },

/\* Mystery scaling numbers \*/

{ 0x70, 0x3a }, { 0x71, 0x35 },

{ 0x72, 0x11 }, { 0x73, 0xf0 },

{ 0xa2,/\* 0x02 changed to 1\*/1 }, { REG\_COM10, 0x0 },

/\* Gamma curve values \*/

{ 0x7a, 0x20 }, { 0x7b, 0x10 },

{ 0x7c, 0x1e }, { 0x7d, 0x35 },

{ 0x7e, 0x5a }, { 0x7f, 0x69 },

{ 0x80, 0x76 }, { 0x81, 0x80 },

{ 0x82, 0x88 }, { 0x83, 0x8f },

{ 0x84, 0x96 }, { 0x85, 0xa3 },

{ 0x86, 0xaf }, { 0x87, 0xc4 },

{ 0x88, 0xd7 }, { 0x89, 0xe8 },

/\* AGC and AEC parameters. Note we start by disabling those features,

then turn them only after tweaking the values. \*/

{ REG\_COM8, COM8\_FASTAEC | COM8\_AECSTEP },

{ REG\_GAIN, 0 }, { REG\_AECH, 0 },

{ REG\_COM4, 0x40 }, /\* magic reserved bit \*/

{ REG\_COM9, 0x18 }, /\* 4x gain + magic rsvd bit \*/

{ REG\_BD50MAX, 0x05 }, { REG\_BD60MAX, 0x07 },

{ REG\_AEW, 0x95 }, { REG\_AEB, 0x33 },

{ REG\_VPT, 0xe3 }, { REG\_HAECC1, 0x78 },

{ REG\_HAECC2, 0x68 }, { 0xa1, 0x03 }, /\* magic \*/

{ REG\_HAECC3, 0xd8 }, { REG\_HAECC4, 0xd8 },

{ REG\_HAECC5, 0xf0 }, { REG\_HAECC6, 0x90 },

{ REG\_HAECC7, 0x94 },

{ REG\_COM8, COM8\_FASTAEC | COM8\_AECSTEP | COM8\_AGC | COM8\_AEC },

{ 0x30, 0 }, { 0x31, 0 },//disable some delays

/\* Almost all of these are magic "reserved" values. \*/

{ REG\_COM5, 0x61 }, { REG\_COM6, 0x4b },

{ 0x16, 0x02 }, { REG\_MVFP, 0x07 },

{ 0x21, 0x02 }, { 0x22, 0x91 },

{ 0x29, 0x07 }, { 0x33, 0x0b },

{ 0x35, 0x0b }, { 0x37, 0x1d },

{ 0x38, 0x71 }, { 0x39, 0x2a },

{ REG\_COM12, 0x78 }, { 0x4d, 0x40 },

{ 0x4e, 0x20 }, { REG\_GFIX, 0 },

/\*{0x6b, 0x4a},\*/{ 0x74, 0x10 },

{ 0x8d, 0x4f }, { 0x8e, 0 },

{ 0x8f, 0 }, { 0x90, 0 },

{ 0x91, 0 }, { 0x96, 0 },

{ 0x9a, 0 }, { 0xb0, 0x84 },

{ 0xb1, 0x0c }, { 0xb2, 0x0e },

{ 0xb3, 0x82 }, { 0xb8, 0x0a },

/\* More reserved magic, some of which tweaks white balance \*/

{ 0x43, 0x0a }, { 0x44, 0xf0 },

{ 0x45, 0x34 }, { 0x46, 0x58 },

{ 0x47, 0x28 }, { 0x48, 0x3a },

{ 0x59, 0x88 }, { 0x5a, 0x88 },

{ 0x5b, 0x44 }, { 0x5c, 0x67 },

{ 0x5d, 0x49 }, { 0x5e, 0x0e },

{ 0x6c, 0x0a }, { 0x6d, 0x55 },

{ 0x6e, 0x11 }, { 0x6f, 0x9e }, /\* it was 0x9F "9e for advance AWB" \*/

{ 0x6a, 0x40 }, { REG\_BLUE, 0x40 },

{ REG\_RED, 0x60 },

{ REG\_COM8, COM8\_FASTAEC | COM8\_AECSTEP | COM8\_AGC | COM8\_AEC | COM8\_AWB },

/\* Matrix coefficients \*/

{ 0x4f, 0x80 }, { 0x50, 0x80 },

{ 0x51, 0 }, { 0x52, 0x22 },

{ 0x53, 0x5e }, { 0x54, 0x80 },

{ 0x58, 0x9e },

{ REG\_COM16, COM16\_AWBGAIN }, { REG\_EDGE, 0 },

{ 0x75, 0x05 }, { REG\_REG76, 0xe1 },

{ 0x4c, 0 }, { 0x77, 0x01 },

{ REG\_COM13, /\*0xc3\*/0x48 }, { 0x4b, 0x09 },

{ 0xc9, 0x60 }, /\*{REG\_COM16, 0x38},\*/

{ 0x56, 0x40 },

{ 0x34, 0x11 }, { REG\_COM11, COM11\_EXP | COM11\_HZAUTO },

{ 0xa4, 0x82/\*Was 0x88\*/ }, { 0x96, 0 },

{ 0x97, 0x30 }, { 0x98, 0x20 },

{ 0x99, 0x30 }, { 0x9a, 0x84 },

{ 0x9b, 0x29 }, { 0x9c, 0x03 },

{ 0x9d, 0x4c }, { 0x9e, 0x3f },

{ 0x78, 0x04 },

/\* Extra-weird stuff. Some sort of multiplexor register \*/

{ 0x79, 0x01 }, { 0xc8, 0xf0 },

{ 0x79, 0x0f }, { 0xc8, 0x00 },

{ 0x79, 0x10 }, { 0xc8, 0x7e },

{ 0x79, 0x0a }, { 0xc8, 0x80 },

{ 0x79, 0x0b }, { 0xc8, 0x01 },

{ 0x79, 0x0c }, { 0xc8, 0x0f },

{ 0x79, 0x0d }, { 0xc8, 0x20 },

{ 0x79, 0x09 }, { 0xc8, 0x80 },

{ 0x79, 0x02 }, { 0xc8, 0xc0 },

{ 0x79, 0x03 }, { 0xc8, 0x40 },

{ 0x79, 0x05 }, { 0xc8, 0x30 },

{ 0x79, 0x26 },

{ 0xff, 0xff }, /\* END MARKER \*/

};

void error\_led(void){

DDRB |= 32;//make sure led is output

while (1){//wait for reset

PORTB ^= 32;// toggle led

\_delay\_ms(100);

}

}

void twiStart(void){

TWCR = \_BV(TWINT) | \_BV(TWSTA) | \_BV(TWEN);//send start

while (!(TWCR & (1 << TWINT)));//wait for start to be transmitted

if ((TWSR & 0xF8) != TW\_START)

error\_led();

}

void twiWriteByte(uint8\_t DATA, uint8\_t type){

TWDR = DATA;

TWCR = \_BV(TWINT) | \_BV(TWEN);

while (!(TWCR & (1 << TWINT))) {}

if ((TWSR & 0xF8) != type)

error\_led();

}

void twiAddr(uint8\_t addr, uint8\_t typeTWI){

TWDR = addr;//send address

TWCR = \_BV(TWINT) | \_BV(TWEN); /\* clear interrupt to start transmission \*/

while ((TWCR & \_BV(TWINT)) == 0); /\* wait for transmission \*/

if ((TWSR & 0xF8) != typeTWI)

error\_led();

}

void wrReg(uint8\_t reg, uint8\_t dat){

//send start condition

twiStart();

twiAddr(camAddr\_WR, TW\_MT\_SLA\_ACK);

twiWriteByte(reg, TW\_MT\_DATA\_ACK);

twiWriteByte(dat, TW\_MT\_DATA\_ACK);

TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);//send stop

\_delay\_ms(1);

}

static uint8\_t twiRd(uint8\_t nack){

if (nack){

TWCR = \_BV(TWINT) | \_BV(TWEN);

while ((TWCR & \_BV(TWINT)) == 0); /\* wait for transmission \*/

if ((TWSR & 0xF8) != TW\_MR\_DATA\_NACK)

error\_led();

return TWDR;

}

else{

TWCR = \_BV(TWINT) | \_BV(TWEN) | \_BV(TWEA);

while ((TWCR & \_BV(TWINT)) == 0); /\* wait for transmission \*/

if ((TWSR & 0xF8) != TW\_MR\_DATA\_ACK)

error\_led();

return TWDR;

}

}

uint8\_t rdReg(uint8\_t reg){

uint8\_t dat;

twiStart();

twiAddr(camAddr\_WR, TW\_MT\_SLA\_ACK);

twiWriteByte(reg, TW\_MT\_DATA\_ACK);

TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);//send stop

\_delay\_ms(1);

twiStart();

twiAddr(camAddr\_RD, TW\_MR\_SLA\_ACK);

dat = twiRd(1);

TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);//send stop

\_delay\_ms(1);

return dat;

}

void wrSensorRegs8\_8(const struct regval\_list reglist[]){

uint8\_t reg\_addr, reg\_val;

const struct regval\_list \*next = reglist;

while ((reg\_addr != 0xff) | (reg\_val != 0xff)){

reg\_addr = pgm\_read\_byte(&next->reg\_num);

reg\_val = pgm\_read\_byte(&next->value);

wrReg(reg\_addr, reg\_val);

next++;

}

}

void setColor(void){

wrSensorRegs8\_8(yuv422\_ov7670);

}

void setRes(void){

wrReg(REG\_COM3, 4); // REG\_COM3 enable scaling

wrSensorRegs8\_8(qvga\_ov7670);

}

void camInit(void){

wrReg(0x12, 0x80);

\_delay\_ms(100);

wrSensorRegs8\_8(ov7670\_default\_regs);

wrReg(REG\_COM10, 32);//PCLK does not toggle on HBLANK.

}

void arduinoUnoInut(void) {

cli();//disable interrupts

/\* Setup the 8mhz PWM clock

\* This will be on pin 11\*/

DDRB |= (1 << 3);//pin 11

ASSR &= ~(\_BV(EXCLK) | \_BV(AS2));

TCCR2A = (1 << COM2A0) | (1 << WGM21) | (1 << WGM20);

TCCR2B = (1 << WGM22) | (1 << CS20);

OCR2A = 0;//(F\_CPU)/(2\*(X+1))

DDRC &= ~15;//low d0-d3 camera

DDRD &= ~252;//d7-d4 and interrupt pins

\_delay\_ms(3000);

//set up twi for 100khz

TWSR &= ~3;//disable prescaler for TWI (Two wire status register) Informa el estado de las acciones TWI

TWBR = 72;//set to 100khz (Two wire bit rate) controla la frecuencia de reloj (SCL)

//enable serial

//UBRR0H = 0; //High -------- INVESTIGAR, buscar como UART o USART de Arduino

//UBRR0L = 1;//Low --------- 0 = 2M baud rate. 1 = 1M baud. 3 = 0.5M. 7 = 250k 207 is 9600 baud rate.

//UBRR0 = 103;

//UCSR0A |= 2;//double speed aysnc

//UCSR0B = (1 << RXEN0) | (1 << TXEN0);//Enable receiver and transmitter

//UCSR0C = 6;//async 1 stop bit 8bit char no parity bits

}

void StringPgm(const char \* str){

do{

while (!(UCSR0A & (1 << UDRE0)));//wait for byte to transmit

UDR0 = pgm\_read\_byte\_near(str);

while (!(UCSR0A & (1 << UDRE0)));//wait for byte to transmit

} while (pgm\_read\_byte\_near(++str));

}

static void captureImg(uint16\_t wg, uint16\_t hg){

uint16\_t y, x;

//StringPgm(PSTR("\*RDY\*"));

while (!(PIND & 8));//wait for high

while ((PIND & 8));//wait for low

y = hg;

while (y--){

x = wg;

//while (!(PIND & 256));//wait for high

while (x--){

while ((PIND & 4));//wait for low

//UDR0 = (PINC & 15) | (PIND & 240);

//UDR0 = 'A';

int datos = (PINC & 15) | (PIND & 240);

sendData("AT+CIPSEND=4\r\n",0);

ESP.print(datos);

//while (!(UCSR0A & (1 << UDRE0)));//wait for byte to transmit

while (!(PIND & 4));//wait for high

while ((PIND & 4));//wait for low

while (!(PIND & 4));//wait for high

}

// while ((PIND & 256));//wait for low

}

\_delay\_ms(100);

}

void setup(){

//sendData("AT+RST\r\n", 0); //Reseteo de modulo

//sendData("AT+CIOBAUD=115200\r\n", 0); // Configuro la velocidad en baudios

//Serial.println("-------FIN CONFIG VELNUEVA------");

//ESP.begin(VELNUEVA);

//sendData("AT+CIPSTART=\"UDP\",\"192.168.4.2\",50494",0); //Establecer conexion con el IP/Puerto

//Serial.println("-------conexion UDP------");

//sendData("AT+CIPSTATUS",1000);

//armarBuffer(frame,0,MAX-1);

// Configuración de camara y captura de imagenes

Serial.begin(19200);

//Configuracion de modulo Wifi

ESP.begin(115200); //Setea la velocidad en la que va a trabajar el modulo

Serial.println("-------Camara configurada------");

}

int comandosAT(int conf){

Serial.println("---------Comandos AT-------");

while (conf ==0){

if (ESP.available())

{ char c = ESP.read() ;

Serial.print(c);

}

if (Serial.available())

{ char c = Serial.read();

ESP.print(c);

if (c=='X'){

conf = 1;

}

}

}

arduinoUnoInut();

Serial.println("-------Paso arduinoUnoInut------");

camInit();

Serial.println("-------Paso camInit------");

setRes();

Serial.println("-------Paso setRes------");

setColor();

Serial.println("-------Paso setColor------");

wrReg(0x11, 11); //Earlier it had the value: wrReg(0x11, 12); New version works better for me :) !!!!

return conf;

}

int conf = 0;

void loop(){

conf = comandosAT(conf);

if (conf == 1){

captureImg(320, 240); // 320x240 formato por defecto

}

conf=0;

}

# Caso de prueba Módulo Bluetooth HC05-01

|  |  |
| --- | --- |
| **Caso de prueba** | Probar la velocidad del Bluetooth |
| **Identificador caso de prueba** | BluetoothHC05-01-pruebaVelocidad |
| **Función probar** | Comunicación por Bluetooth |
| **Objetivo** | Determinar la velocidad máxima de transferencia |
| **Descripción** | Se desea verificar la velocidad de conectividad que se puede alcanzar entre una computadora con Bluetooth y el Arduino conectado al HC05 |
| **Criterios de éxito** | Alcanzar una velocidad que permita transmitir 10 fps con un tamaño de 300kb por segundo, mínimamente |
| **Criterios de falla** | No alcanzar la velocidad requerida de fps |
| **Precondiciones** | Testear un entorno sin obstáculos y línea visual.  Establecer la mayor velocidad posible de baudios de transmisión |
| **Necesidades para el caso de prueba** | Módulo arduino UNO  BT HC05  Cables Hembra-Macho (x6) |
| **Autor** | Schlapp-Mansilla |
| **Fecha de creación** | 8-3-2017 |
| **Resultados** | Se estableció la comunicación entre los módulos Bluetooth, Arduino<-> PC. La prueba no fue exitosa. Se alcanzó una velocidad de transmisión de 1,17kb/sg. |
| **Código fuente** | ComunicaciónBluetooth.ino |
| **Imágenes** |  |
| **Sketch** |  |

# Comunicación Bluetooth.ino

#include <SoftwareSerial.h>

SoftwareSerial bluetooth(10, 11); // RX, TX

long cont;

long tiempoini;

void setup()

{

Serial.begin(9600);

bluetooth.begin(9600);

cont = 0;

}

int paso = 0;

long segundos=0;

long tiempofin;

unsigned long antes=0;

unsigned long despues=0;

void loop()

{

tiempoini = millis();

while(segundos < 1.0){

antes=millis();

if ( bluetooth.available()){

bluetooth.read();

despues = millis() - antes;

cont++;

paso=1;

}

tiempofin = millis()- tiempoini;

segundos = tiempofin/1000;

}

if (paso != 0){

Serial.println("Bytes leidos:"+ String(cont)+" tiempo antesDespues micros:"+String(despues));

paso =0;

}

segundos =0;

cont=0;

}