# Array di caratteri

//char A

{

B00000000,

B00111000,

B01000100,

B01000100,

B01111100,

B01000100,

B01000100,

B01000100,

B00000000,

B00000000,

B00000000

}

Abbiamo rifatto l’array dei caratteri utilizzando stavolta dei byte in modo da utilizzare molto meno spazio.

L’array è bidimensionale, esso contiene infatti degli array di byte e ogni array rappresenta una lettera, come si può vedere qui di seguito con l’array della lettera “A”, ogni bit a 1 rappresenta la lettera a cui è associato l’array di byte.

Per distinguere i caratteri che verranno stampati, viene preso il numero del carattere (seguendo la tabella ASCII), ne viene sottratto il numero di caratteri iniziali mancanti nel nostro array di caratteri e si avrà come risultato l’indice corrispondente nell’array. I caratteri mancanti, i primi 31 della tabella ASCII insieme all’ultimo, non serviva rappresentarli per il nostro FanClock.

Ecco un esempio di come viene scelto un carattere:

* Scelgo il carattere A che corrisponde a 65 (ASCII).
* Sottraggo 31 che sono il numero di caratteri mancanti dall’inizio del tabella ASCII.
* Ottengo 34 che è la posizione all’interno del mio array dei caratteri

Dopodiché abbiamo realizzato il programma di test dell’array che passa tutti i caratteri e li stampa, si chiama charPrinter e si trova nella cartella d’implementazione

//array che contiene tutti i caratteri da noi realizzati

byte chars[96][11] = {

//…

}

//numero del carattere corrente

int nc = -1;

//messaggio da stampare

String message = " !\"#$%&'()\*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^\_`abcdefghijklmbopqrstuvwxyz{|}~";

//numero di caratteri iniziali nella tabella ascii che non vengono utilizzati

int difAscii = 31;

//numero di rige in ogni carattere

int charH = 10;

//numero di colonne per ogni carattere

int charW = 8;

void setup() {

//inizio la comunicazione seriale

Serial.begin(9600);

}

void loop() {

//passo al prossimo carattere

nc++;

//se arrivo alla fine del messaggio ricomincio

if(nc == message.length()){

nc = 0;

}

//stampo il carattere corrente

printChar(message.charAt(nc),0, false);

//aspetto mezzo secondo

delay(500);

}

//il metodo printChar riceve il carattere da stampare,

//in che direzione stamparlo e se stampare lo sfondo o il carattere

void printChar(char c, int dir, bool neg){

//scelgo il bit (0 o 1) in cui disegnero qualcosa in base se neg sia true o false

int drawBit = 1;

if(neg){

drawBit = 0;

}

//in base alla direzione scelta stampo il carattere in modo diverso

if(dir == 1){

//direzione verso destra

for(int i = charW-1; i >= 0; i--){

for(int j = charH-1; j >= 0; j--){

//se il bit letto corrisponde al drawBit disegno un asterisco

if(drawBit == bitRead(chars[((byte)c)-difAscii][j],i)){

Serial.print("\*");

}

else{

Serial.print(" ");

}

}

Serial.println("");

}

Serial.println("");

}

else if(dir == 2){

//direzione verso il basso

//…

}

else if(dir == 3){

//direzione verso sinistra

//…

}

else{

//direzione verso il sopra

//…

}

}

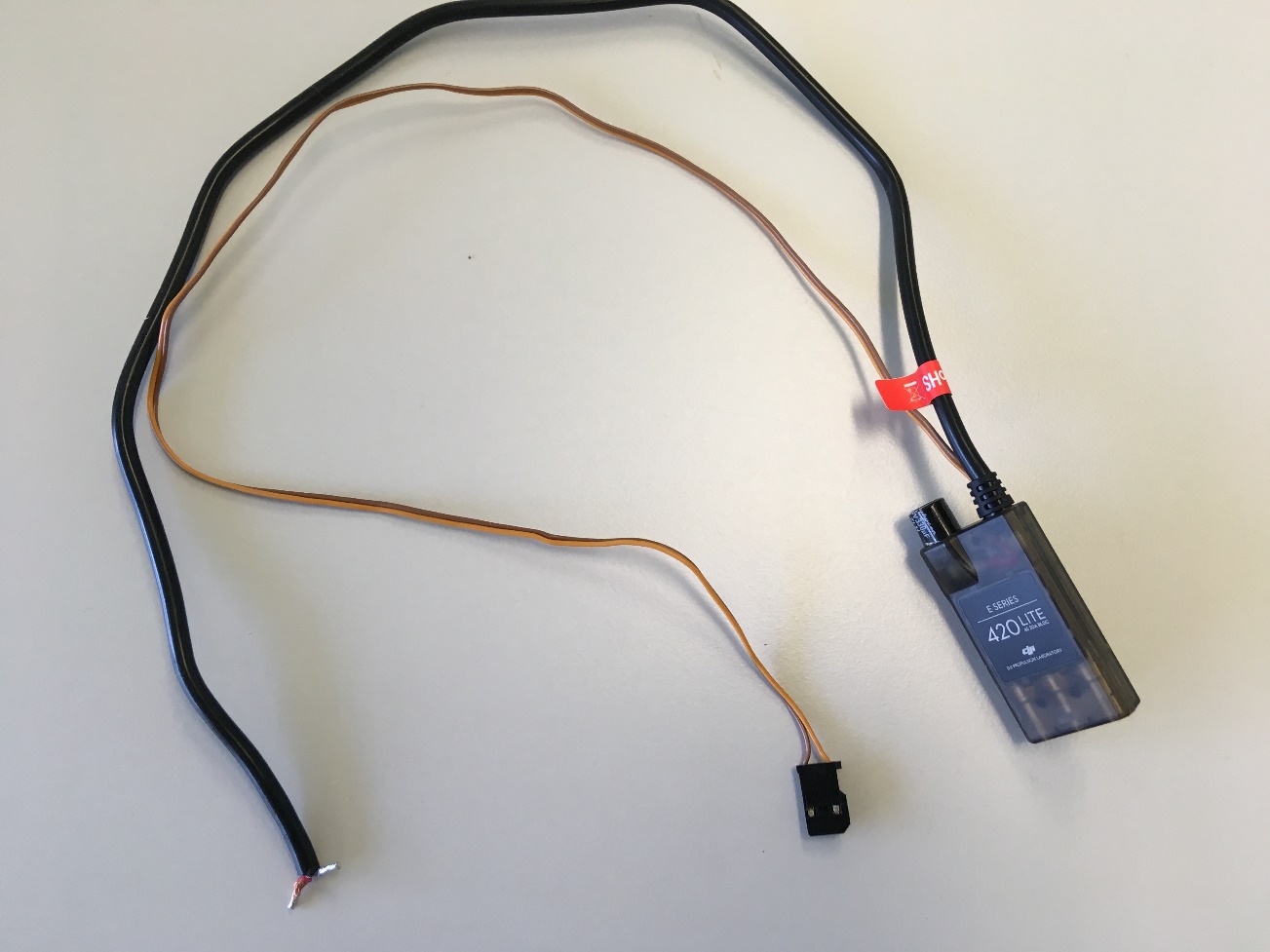
# Motore dji

Per controllare il motore dji 2312E (datasheet disponibile nella cartella documentazione) abbiamo creato il codice contenuto nel file djiMotorTest1.ino contenuto nella cartella implementazione.

Questo codice permette di controllare il motore tramite la porta seriale di Arduino.

Ho incluso la libreria dei Servo per poter controllare il motore tramite l’ESC 420, ogni volta che cambia il valore lo applico al motore. Il valore varia circa da 1000 a 1900 microsecondi, all’inizio applico il valore minimo altrimenti l’ESC non permette al motore di partire.

Ecco come viene collegato l’ESC:



12V (alimentatore)

GND (di arduino)

GND

(alimentatore)

3 output collegati ai 3 input del motore per controllarlo

Input di controllo del motore (collegato al PIN 9 di arduino)

L’alimentatore esterno serve perché arduino non riesce a fornire abbastanza corrente.

Ecco il codice per controllare il motore:

//includo la libreria dei servo

#include <Servo.h>

//il pin di controllo collegato all'esc

int motorPin = 9;

//valo che andrò a passare all'esc

int value = 1050;

//oggetto che rappresenta l'esc

Servo esc;

void setup() {

//associo il pin di contollo all'oggetto esc

esc.attach(motorPin);

//inizio la comunicazione seriale

Serial.begin(9600);

}

void loop() {

Serial.println(value);

//invio all'esc il valore contenuto in value

esc.writeMicroseconds(value);

//se la comunicazione seriale è disponibile inserisco in value il valore passato

if(Serial.available()){

value = Serial.parseInt();

}

}

## Controllo led via wireless

Per poter controllare la striscia di led con Fishino Guppy l’abbiamo collegata nel seguente modo: DATAPIN 6 e CLOCKPIN 7 (secondo PIN di controllo della striscia DotStar che la differenzia dal collegamento della vecchia striscia).

Dopodiché abbiamo realizzato del codice che fa diventare Fishino Guppy raggiungibile nella rete, infatti lo abbiamo collegato via wireless con un access point alla nostra rete di scuola in modo che ricevesse un ip dinamicamente dal DHCP (È anche possibile dargli un ip statico). Quando Fishino riesce a collegarsi alla rete stampa il suo stato in modo che sia possibile sapere come raggiungerlo e si mette in ascolto di pacchetti UDP, così sarà possibile mandare dei messaggi per controllare la striscia.

Ecco la parte del codice dove vengono definite tutte le variabile per collegarsi alla rete, per controllare la striscia e in base a quelle che sia ha definito, Fishino andrà a creare un propria rete wireless o a collegarsi ad un’altra rete (come in questo caso):

#include <Adafruit\_DotStar.h>

#include <Fishino.h>

#include <SPI.h>

#define NUMPIXELS 144 // Number of LEDs in strip (numero di led della striscia)

// Here's how to control the LEDs from any two pins:

// pin di controllo della striscia

#define DATAPIN 6

#define CLOCKPIN 7

// oggetto per poter controllare la striscia

Adafruit\_DotStar strip = Adafruit\_DotStar(

NUMPIXELS, DATAPIN, CLOCKPIN, DOTSTAR\_BRG);

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

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

// CONFIGURATION DATA -- ADAPT TO YOUR NETWORK !!!

// DATI DI CONFIGURAZIONE -- ADATTARE ALLA PROPRIA RETE WiFi !!!

#ifndef \_\_MY\_NETWORK\_H

// here pur SSID of your network

// inserire qui lo SSID della rete WiFi

#define MY\_SSID "FanClock"

// here put PASSWORD of your network. Use "" if none

// inserire qui la PASSWORD della rete WiFi -- Usare "" se la rete non ￨ protetta

#define MY\_PASS "Admin01!"

// comment this line if you want a dynamic IP through DHCP

// obtained IP will be printed on serial port monitor

// commentare la linea seguente per avere un IP dinamico tramite DHCP

// l'IP ottenuto verrà visualizzato sul monitor seriale

//#define IPADDR 169, 254, 0, 40

//#define GATEWAY 192, 168, 1, 1

//#define NETMASK 255, 255, 0, 0

#endif

Qui invece (nel setup), va ad applicare i dati quindi avviare la striscia, collegarsi alla rete e avviare l’ascolto di pacchetti UDP:

void setup()

{

strip.begin(); // Initialize pins for output

strip.show();

// Initialize serial and wait for port to open

// Inizializza la porta seriale e ne attende l'apertura

Serial.begin(115200);

// reset and test WiFi module

// resetta e testa il modulo WiFi

while(!Fishino.reset())

Serial << F("Fishino RESET FAILED, RETRYING...\n");

Serial << F("Fishino WiFi RESET OK\n");

// local port to listen on

// porta UDP sulla quale Fishino attende i pacchetti

unsigned int localPort = 2390;

// END OF CONFIGURATION DATA //

// FINE CONFIGURAZIONE //

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

// define ip address if required

// NOTE : if your network is not of type 255.255.255.0 or your gateway is not xx.xx.xx.1

// you should set also both netmask and gateway

#ifdef IPADDR

IPAddress ip(IPADDR);

#ifdef GATEWAY

IPAddress gw(GATEWAY);

#else

IPAddress gw(ip[0], ip[1], ip[2], 1);

#endif

#ifdef NETMASK

IPAddress nm(NETMASK);

#else

IPAddress nm(255, 255, 255, 0);

#endif

#endif

// buffer to hold incoming packet

// buffer per contenere i pacchetti ricevuti

char packetBuffer[255];

// a string to send back

// una stringa di conferma da spedire al mittente

char ReplyBuffer[] = "acknowledged";

// the UDP client/server

// il client/server UDP

FishinoUDP Udp;

// go into station mode

// imposta la modalità stazione

Fishino.setMode(STATION\_MODE);

// try forever to connect to AP

// tenta la connessione finchè non riesce

Serial << F("Connecting to AP...");

while(!Fishino.begin(MY\_SSID, MY\_PASS))

{

Serial << ".";

delay(2000);

}

Serial << "OK\n";

// setup IP or start DHCP client

// imposta l'IP statico oppure avvia il client DHCP

#ifdef IPADDR

Fishino.config(ip, gw, nm);

#else

Fishino.staStartDHCP();

#endif

// wait till connection is established

Serial << F("Waiting for IP...");

while(Fishino.status() != STATION\_GOT\_IP)

{

Serial << ".";

delay(500);

}

Serial << "OK\n";

// print connection status on serial port

// stampa lo stato della connessione sulla porta seriale

printWifiStatus();

// starts listening on local port

// inizia l'ascolto dei pacchetti UDP alla porta specificata

Serial << F("Starting connection to server...\n");

Udp.begin(localPort);

Quindi ogni volta che arriva un pacchetto UDP a Fishino ne viene passato il contenuto e se rispetta il seguente protocollo andrà ad applicare i dati alla striscia:

* “n” per numero di led da accendere
* “r” per il valore del rosso del colore
* “g” per il valore del verde del colore
* “b” per il valore del blu del colore

Ecco ad esempio con il messaggio “n020r255g000b000” andranno ad accendersi 20 led con il colore rosso.

Ci si deve ricordare che i numeri devono essere composti da tre cifre per poter funzionare in questo programma.

Ecco la parte di codice che si occupa di quanto spiegato prima:

//variabili per controllare se i dati sono già stati letti

bool cLedN = false;

bool cRed = false;

bool cGreen = false;

bool cBlue = false;

//variabili per contenere i diversi dati

int nLed = 0;

int color[3] = {0,0,0};

void loop()

{

//azzero le variabili di controllo

cLedN = false;

cRed = false;

cGreen = false;

cBlue = false;

//accendo i led con gli ultimi dati che ho ricevuto

for(int i = 0; i < nLed; i++){

strip.setPixelColor(i,color[1],color[0],color[2]);

}

strip.show();

// if there's data available, read a packet

// se ci sono dati in arrivo, li stampa

int packetSize = Udp.parsePacket();

if (packetSize)

{

Serial << F("Received packet of size ");

Serial << packetSize << "\n";

Serial << F("From ");

IPAddress remoteIp = Udp.remoteIP();

Serial << remoteIp;

Serial << F(", port ");

Serial << Udp.remotePort() << "\n";

// read the packet into packetBufffer

// legge il pacchetto nel buffer

int len = Udp.read(packetBuffer, 255);

if (len > 0)

packetBuffer[len] = 0;

Serial << F("Contents:\n");

Serial.println(packetBuffer);

//leggo il contenuto del pacchetto

for(int i = 0; i < 255; i++){

String s = "";

if(packetBuffer[i] == 'n' && !cLedN){

//leggo il numero di led da accendere

s = String(packetBuffer[i+1])+String(packetBuffer[i+2])+String(packetBuffer[i+3]);

Serial.println(s.toInt());

nLed = s.toInt();

cLedN = true;

}

else if(packetBuffer[i] == 'r' && !cRed){

//leggo il valore del rosso

s = String(packetBuffer[i+1])+String(packetBuffer[i+2])+String(packetBuffer[i+3]);

color[0] = s.toInt();

Serial.println(color[0]);

cRed = true;

}

else if(packetBuffer[i] == 'g' && !cGreen){

//leggo il valore del verde

s = String(packetBuffer[i+1])+String(packetBuffer[i+2])+String(packetBuffer[i+3]);

color[1] = s.toInt();

Serial.println(color[1]);

cGreen = true;

}

else if(packetBuffer[i] == 'b' && !cBlue){

//leggo il valore del blu

s = String(packetBuffer[i+1])+String(packetBuffer[i+2])+String(packetBuffer[i+3]);

color[2] = s.toInt();

Serial.println(color[2]);

cBlue = true;

}

}

// send a reply, to the IP address and port that sent us the packet we received

// invia una risposta all'indirizzo IP e alla porta da cui proviene la richiesta

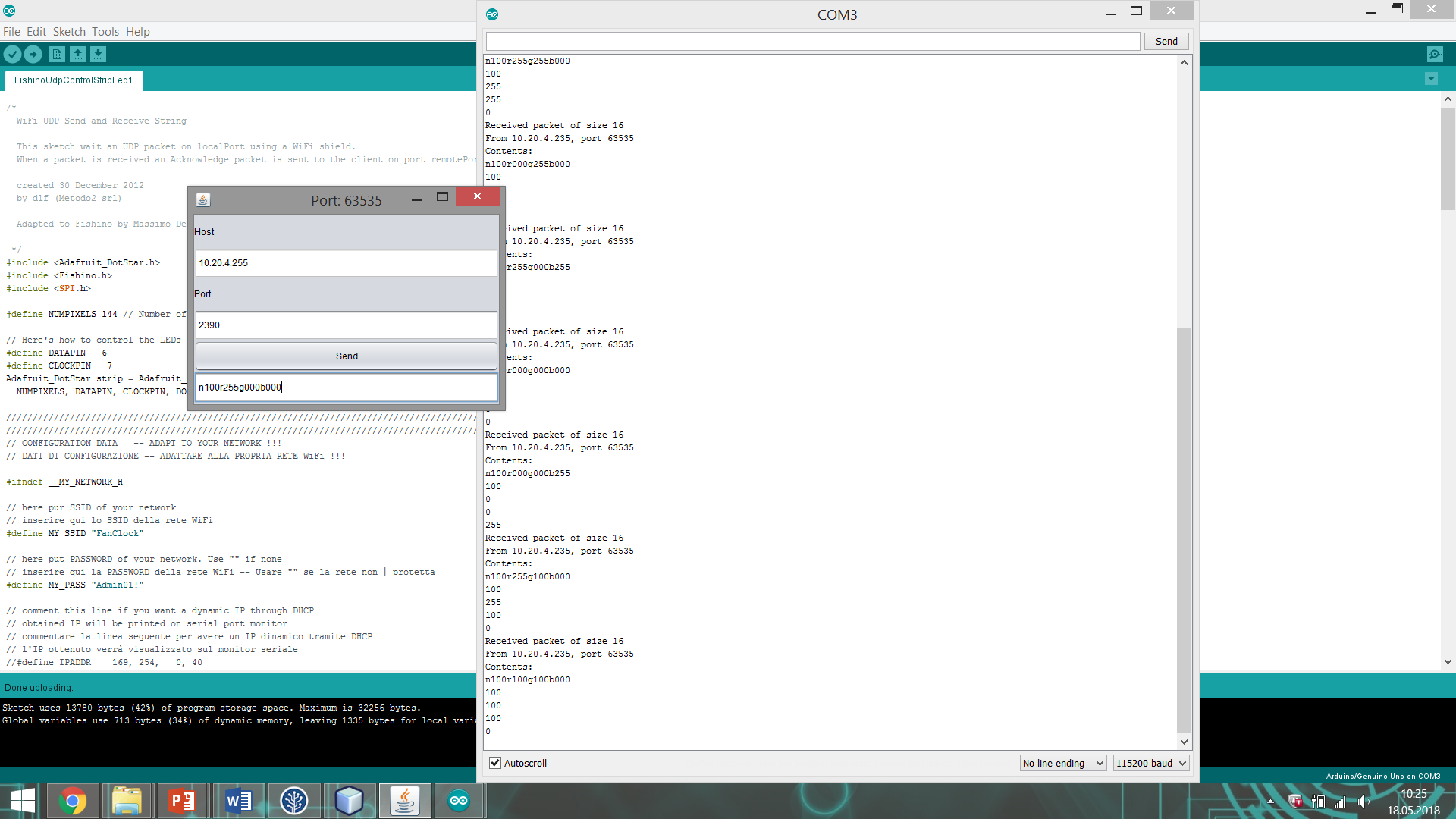
Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());

Udp.write(ReplyBuffer);

Udp.endPacket();

}

}

Naturalmente è possibile inviare i dati con qualsiasi applicazione che manda pacchetti UDP. Ne abbiamo utilizzato una realizzata da noi in java per un’altra occasione molto semplice da usare nella quale si doveva definire host con la sua porta di ascolto a cui inviare il messaggio (pacchetto).

Il programma di controllo dei led si chiama *FishinoUdpControlStripLed1* e si trova nella cartella d’implementazione.

## Connessione con il server NTP

Per quanto riguarda la connessione con il server NTP Fishino si comporta inizialmente quasi nella stessa maniera come nel codice per controllare i led, cioè si collega alla rete di scuola ricevendo un ip dinamico poi però sapendo l’ip del server NTP di scuola (10.20.4.2) gli manda un pacchetto UDP preparato in modo che il server NTP risponda con i dati dell’ora desiderati. Qui di seguito ecco il programma *FishinoUdpNtpClient* del quale abbiamo inserito le parti più importanti cioè dove prepara il pacchetto NTP, lo invia e poi stampa la risposta del server:

// send an NTP request to the time server at the given address

// invia una richiesta al server NTP all'indirizzo fornito

unsigned long sendNTPpacket(IPAddress& address)

{

// set all bytes in the buffer to 0

// azzera il buffer di ricezione NTP

memset(packetBuffer, 0, NTP\_PACKET\_SIZE);

// Initialize values needed to form NTP request

// (see URL above for details on the packets)

// Inizializza i valori da inviare al server NTP

// (vedere URL del server per dettagli sul formato pacchetto)

// LI, Version, Mode

packetBuffer[0] = 0b11100011;

// Stratum, or type of clock

packetBuffer[1] = 0;

// Polling Interval

packetBuffer[2] = 6;

// Peer Clock Precision

packetBuffer[3] = 0xEC;

// 8 bytes of zero for Root Delay & Root Dispersion

packetBuffer[12] = 49;

packetBuffer[13] = 0x4E;

packetBuffer[14] = 49;

packetBuffer[15] = 52;

// all NTP fields have been given values, now

// you can send a packet requesting a timestamp:

// tutti i campi del paccketto NTP sono stati impostati

// è quindi possibile inviare il paccetto di richiesta di data/ora

// NTP requests are to port 123

// beginPacket() just opens the connection

// invia la richiesta NTP alla porta 123

// beginPacket() apre solo la connessione

Udp.beginPacket(address, 123);

// fill UDP buffer with packet data

// riempie il buffer di invio UDP con i dati del pacchetto

Udp.write(packetBuffer, NTP\_PACKET\_SIZE);

// ends and send the UDP packet

// termina ed invia il pacchetto UDP

Udp.endPacket();

return 0;

}

void loop()

{

// send an NTP packet to a time server

Serial << F("Sending UDP request...");

sendNTPpacket(timeServer);

Serial << "OK\n";

// wait to see if a reply is available

delay(1000);

while(Udp.parsePacket())

{

Serial << F("Packet received\n");

// print remote port and IP of incoming packet, just to show them

// stampa IP e porta remoti per mostrare la provenienza del pacchetto

IPAddress remoteIp = Udp.remoteIP();

uint32\_t remotePort = Udp.remotePort();

Serial << F("Remote IP : ") << remoteIp << "\n";

Serial << F("Remote port : ") << remotePort << "\n";

// We've received a packet, read the data from it and put into a buffer

// abbiamo ricevuto un pacchetto, leggiamo i dati ed inseriamoli in un buffer

Udp.read(packetBuffer, NTP\_PACKET\_SIZE);

// the timestamp starts at byte 40 of the received packet and is four bytes,

// or two words, long. First, extract the two words:

// il timestamp inizia dal byte 40 del pacchetto ricevuto, e consiste in 4 bytes

// o due words, long. Innanzitutto estraiamo le due words

unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);

unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);

// combine the four bytes (two words) into a long integer

// this is NTP time (seconds since Jan 1 1900):

// combiniamo i 4 bytes (o 2 words) in un long integer

// che è il tempo NTP (secondi dal primo Gennaio 1900)

unsigned long secsSince1900 = highWord << 16 | lowWord;

Serial << F("Seconds since Jan 1 1900 = ") << secsSince1900 << "\n";

// now convert NTP time into everyday time

// ora convertiamo il tempo NTP in formato leggibile

Serial << F("Unix time = ");

// Unix time starts on Jan 1 1970. In seconds, that's 2208988800

// il tempo Unix inizia dal primo Gennaio 1970. In secondi, sono 2208988800

const unsigned long seventyYears = 2208988800UL;

// subtract seventy years

// sottrae dal tempo NTP la base Unix

unsigned long epoch = secsSince1900 - seventyYears;

// print Unix time:

// stampa il tempo Unix

Serial << epoch << "\n";

// print the hour, minute and second

// stampa ora, minuti e secondi

// UTC is the time at Greenwich Meridian (GMT)

// Tempo UTC (ora al meridiano di Greenwich, GMT)

Serial << F("The UTC time is ");

// print the hour (86400 equals secs per day)

// stampa l'ora (contando 86400 secondi al giorno

Serial << ((epoch % 86400L) / 3600);

Serial.print(':');

if (((epoch % 3600) / 60) < 10)

{

// In the first 10 minutes of each hour, we'll want a leading '0'

// nei primi 10 minuti di ogni ora vogiamo uno zero iniziale

Serial << '0';

}

// print the minute (3600 equals secs per minute)

// stampa i minuti (contando 3600 secondi per minuto)

Serial << ((epoch % 3600) / 60);

Serial << ':';

if ((epoch % 60) < 10)

{

// In the first 10 seconds of each minute, we'll want a leading '0'

// nei primi 10 secondi di ogni minuto vogliamo lo zero iniziale

Serial << '0';

}

// print the second

// stampa i secondi

Serial << epoch % 60 << "\n";

}

// wait ten seconds before asking for the time again

// attende 10 secondi prima di effettuare una nuova richiesta

delay(10000);

}

## Riferimenti

Molte parti dei codici che abbiamo realizzato come test vengono dagli esempi dalle librerie di Fishino come ad esempio *FishinoUdpNtpClient,* che è un esempio delle librerie, solo che ne abbiamo cambiato la configurazione della rete a cui si collega per testarne il funzionamento.

## Conclusioni

Tutte queste parti di codice saranno da mettere insieme per realizzare il FanClock abbiamo già cominciato con il programma *FishinoUdpControlFanClock1* che ha già riscontrato dei problemi che non riusciamo a risolvere. Però la maggiore parte della lavoro è completo.