INTERNAL REPORT

Sistema di monitoraggio in RT della frequenza di rotazione

Report N. 63,
released: 09 03 2017

Alessandro Cabras, Pierluigi Ortu, Andrea Saba, Enrico Urru, Tonino Pisanu

Revisore: Raimondo Concu

OAC Osservatorio Astronomico di Cagliari

Sommario

Sommario 2
1. Introduzione. 4
2. Architettura e schema a blocchi 6
2.1 Introduzione ad Arduino 6
2.3 Schema a blocchi del progetto 6
3. Progettazione e realizzazione dell'hardware. 8
3.1 Il circuito di condizionamento 8
4. Firmware del frequenzimetro 13
4.1 Introduzione sui contatori 13
4.2 Base dei tempi 13
4.3 Blocco di conteggio 15
4.4 Blocco porta o gate 15
5 Output su display LCD 17
5.1 Hardware 17
5.2 Firmware 17
6.1 Comunicazione Ethernet 19
6.1 Hardware 19
6.2 Firmware 20
7. Comunicazione wireless 24
7.1 Hardware 24
7.2 Firmware 24
8. Estensione della memoria 27
8.1 Hardware 27
8.2 Firmware: 27
9. Comunicazione Client/Server 31
9.1 Introduzione ad AJAX 31
9.2 Implementazione lato Arduino 31

9.3 Implementazione lato browser 33
10. Realizzazione della dashboard 36
10.1 Introduzione a Bootstrap 36
10.2 Struttura della pagina 37
10.2.1 Header 37
10.2.2 Control panel 38
10.2.3 Alert banner 38
10.2.4 Body 38
10.2.5 Footer 39
10.3 Script per il controllo dei componenti dinamici 40
10.3.1 Utility.js 40
10.3.2 Graph.js 41
10.3.3 Main.js 42
Conclusioni 44
Riferimenti 45

1. Introduzione.

Questo rapporto interno ha come obiettivo quello di descrivere le fasi di progettazione, realizzazione e test di un sistema di monitoraggio per la misura della velocità di rotazione di motori brushless e/o brushed. Tale dispositivo trova applicazione in diversi ambiti, in SRT ad esempio molti automatismi utilizzano motori di questo tipo e l'analisi della frequenza di rotazione permette, tra le altre cose, di valutarne le prestazioni. Sistemi di questo tipo si trovano anche in dispositivi di dimensioni maggiori come ad esempio sulle eliche di un UAV[1]. Il sistema realizzato comprende un contagiri ottico e una web application per la visualizzazione dei dati in tempo reale, l'uso di tale interfaccia può a sua volta essere esteso a tutti i tipi di dispositivi che necessitano la visione dei dati in real time.

Nello specifico il contagiri non è altro che un frequenzimetro digitale, esso fornisce il valore misurato tramite un circuito in parte digitale e in parte analogico. I vantaggi di questa tipologia di frequenzimetri vanno dalla loro dimensione ridotta al range di misurazione molto esteso.

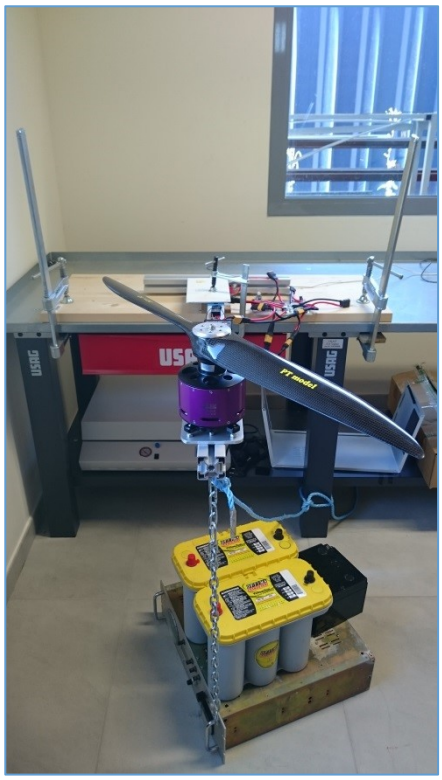

Figura 1: Prototipo dell'elica dell'UAV

[1] Un aeromobile a pilotaggio remoto, comunemente noto come drone, è un velivolo caratterizzato dall'assenza del pilota umano a bordo. Il suo volo è controllato dal computer a bordo del velivolo, sotto il controllo remoto di un navigatore o pilota, sul terreno o in un altro veicolo.

Nel nostro caso l'evento meccanico usato per testare il dispositivo è un'elica collegata ad un motore elettrico come quella in

Figura 1 e la misurazione viene fatta utilizzando un laser e un fotodiodo che posti perpendicolarmente all'elica, inviano un impulso ogni qualvolta il raggio del laser viene interrotto dall'elica che passa sopra di esso, un funzionamento analogo a quello delle comuni fotocellule usate nei cancelli automatici o nei cronometri sportivi.

Questo segnale sarà prima opportunamente condizionato tramite un piccolo circuito e poi inviato a un microcontrollore che eseguirà le elaborazioni necessarie fornendo il risultato finale.

2. Architettura e schema a blocchi.

2.1 Introduzione ad Arduino

Arduino[2] è una scheda elettronica di piccole dimensioni con un microcontrollore ATmega, sviluppata da alcuni membri dell'Interaction Design Institute di Ivrea, ideata come strumento hardware per la prototipazione rapida e per scopi hobbistici, didattici e professionali.

Con Arduino si possono realizzare in maniera relativamente rapida e semplice piccoli dispositivi come controllori di luci, di velocità per motori, sensori di luce, temperatura e umidità e molti altri progetti che utilizzano sensori, attuatori e comunicazione con altri dispositivi. È fornito di un semplice ambiente di sviluppo integrato per la programmazione. Tutto il software a corredo è libero, e gli schemi circuitali sono distribuiti come hardware libero.

2.3 Schema a blocchi del progetto

Nello schema in Figura 2 si può osservare l'architettura di massima del sistema che ha come elemento centrale il controllore Arduino il quale riceve il segnale condizionato da un fotodiodo, lo elabora e lo visualizza su un display.

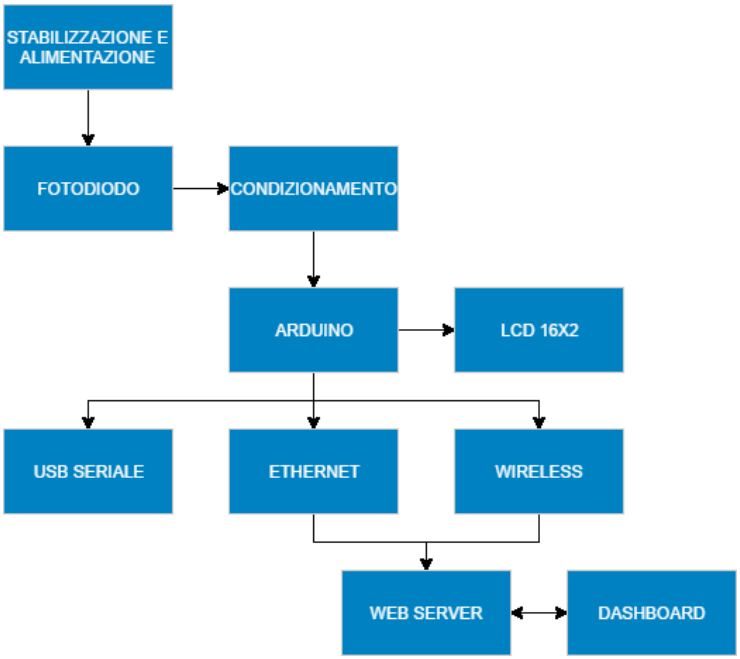

Figura 2: Schema a blocchi

[2] Il nome della scheda deriva da quello di un bar di Ivrea frequentato da alcuni dei fondatori del progetto (che richiama a sua volta il nome di Arduino d'Ivrea, Re d'Italia nel 1002).

In uscita il dato viene mandato via USB in seriale (per debugging) e via ethernet e wireless ad un web server nella rete locale. I dati sul web server possono essere visualizzati grazie ad una web application che fa da cruscotto.

3. Progettazione e realizzazione dell'hardware.

3.1 Il circuito di condizionamento

Come anticipato nello schema in Figura 2, il segnale proveniente dal fotodiodo non può essere utilizzato per la misurazione senza che prima venga opportunamente condizionato.

Per condizionamento del segnale si intende quell'insieme di operazioni (es: amplificazione, filtraggio, adattamento di livello) che occorre effettuare su un segnale elettrico per renderlo adatto all'elaborazione successiva.

Il condizionamento dei segnali è principalmente utilizzato nel campo dell'acquisizione dati, in cui i segnali provenienti da un sensore devono essere normalizzati e filtrati a livelli che li rendano compatibili per la successiva conversione analogico-digitale in maniera che possano essere letti tramite strumenti software.

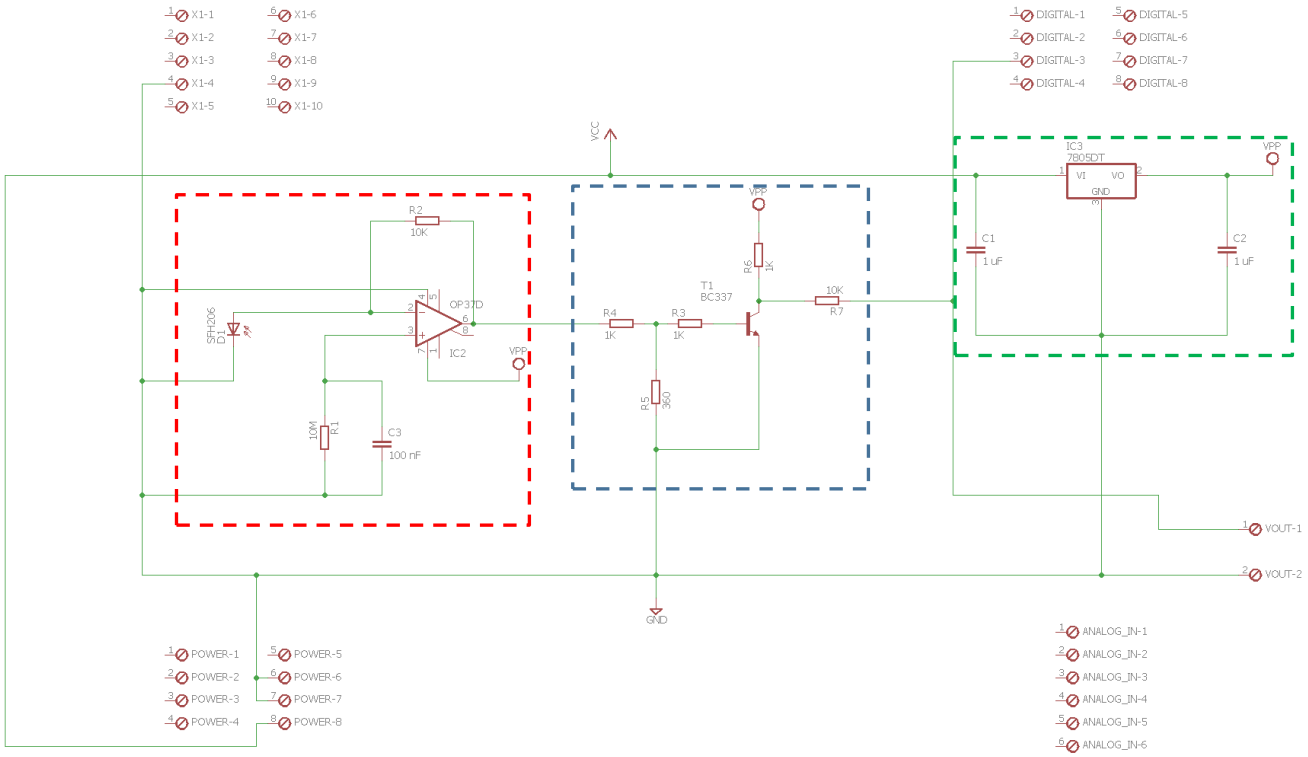

Figura 3: Schematico del circuito.

In Figura 3 è riportato lo schema complessivo del circuito di condizionamento, realizzato utilizzando il software di progettazione elettronica Eagle[1] e in esso sono evidenziati i 3 blocchi principali che saranno analizzati singolarmente nei paragrafi successivi.

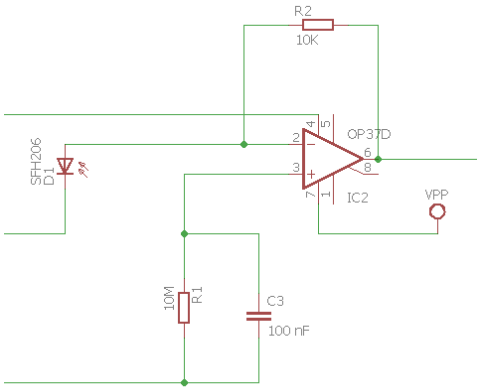

Figura 4: Amplificazione

La prima parte del circuito riportata in Figura 4, ha la funzione di prelevare il segnale, convertirlo e amplificarlo. Al suo interno il fotodiodo costituisce il sensore di misurazione vero e proprio tramite il quale si apprezzeranno le interruzioni del fascio luminoso e un amplificatore operazionale del tipo OPA37 della Texas Instruments[2], amplificherà il livello del segnale affinché esso possa essere elaborato.

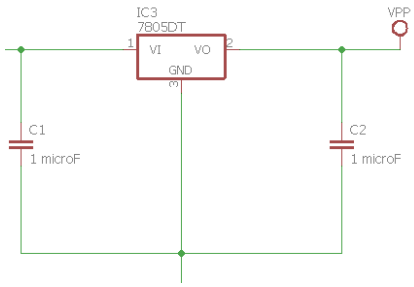

Figura 5: Stabilizzazione

Il secondo blocco del circuito è stato aggiunto successivamente ed ha il compito di stabilizzare la tensione di alimentazione a 5V, in modo da fornire la tensione di alimentazione corretta al microcontrollore.

Lo stabilizzatore utilizzato è della serie KA78XXE della Fairchild[3] e garantisce una tensione tra 4.80V e 5.20V in uscita a fronte di una tensione di ingresso compresa tra 5V e 24V, questo in combinazione ai condensatori C1 e C2 di filtraggio garantisce sempre la giusta alimentazione nel range specificato.

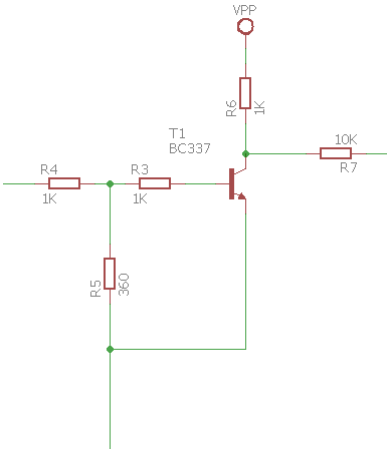

Figura 6: Adattamento di livello

Il terzo e ultimo blocco quello in Figura 6, è stato inserito dopo i primi test che hanno rilevato una tensione in uscita di 1.6V con sensore libero e di 4.3V con sensore eccitato dal laser.

Il valore di 1.6V è troppo elevato e viene letto da Arduino come valore logico alto. Il circuito in questione effettua una taratura dei due valori ovvero modifica il range portando a 0.05V il valore basso e a 4.9V il valore alto, garantendo che questi valori siano letti dal microcontrollore rispettivamente come valore logico basso e valore logico alto.

Per la fase di test iniziale, il circuito è stato realizzato su breadboard utilizzando un'alimentazione continua a 9V come si può vedere in Figura 7. La prima modifica apportata, vista l'assenza di variazioni apprezzabili sui valori di uscita, è stata quella di sostituire il primo fotodiodo quadrato di lato 5mm con uno quadrato di lato 10mm del tipo PDB-C613-2[4] per avere meno problemi con il puntamento del laser.

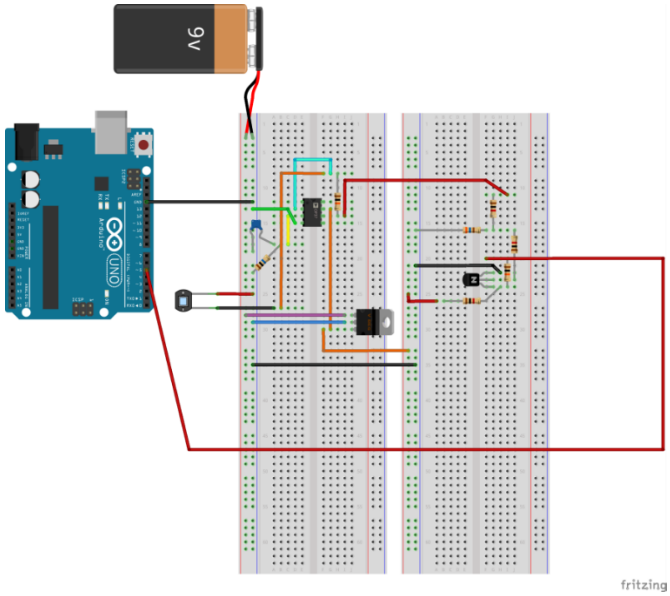

Figura 7: Circuito di test su breadboard

Una volta terminati i test con la breadboard si ha la certezza che il circuito realizzato si comporterà esattamente come previsto in fase di progettazione, perciò si può passare alla realizzazione della versione definitiva sotto forma di chip, Eagle permette di ricavare dallo schema la scheda PCB, lasciando all'utente la possibilità di posizionare i componenti e di realizzare le piste di collegamento, operazioni che dovranno essere fatte in vista della successiva stampa e tenendo quindi conto dei limiti della stampante stessa. La scheda è stata dimensionata tenendo conto delle dimensioni di Arduino in modo che le due schede potessero essere poi sovrapposte

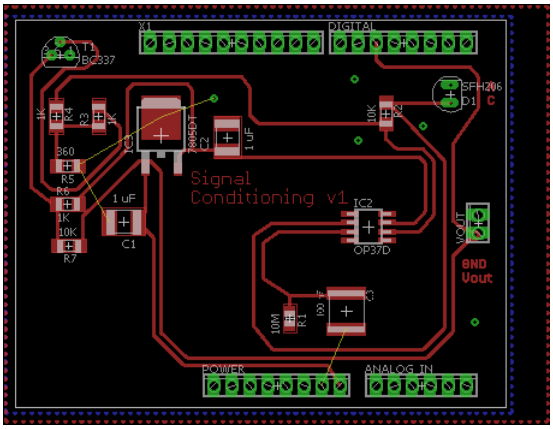

Figura 8: Progetto scheda PCB.

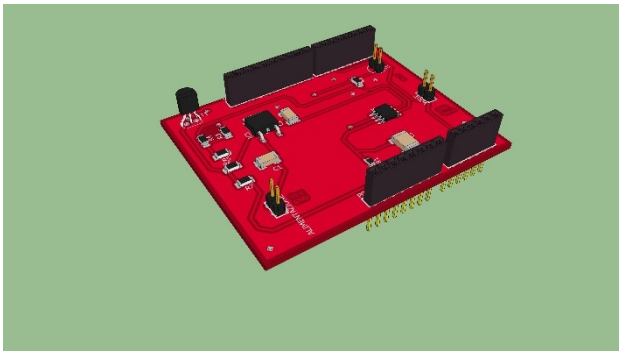

Figura 9: Rendering 3D del circuito.

Per avere un'idea migliore dell'ingombro dei componenti, verificarne il corretto posizionamento e l'interazione con gli altri elementi del progetto, è stato realizzato anche un modello 3D della scheda, tramite il software Google Sketchup[5] illustrato in Figura 9.

Terminata la fase di progettazione, si passa alla stampa della scheda. Tramite Eagle si generano i file Gerber[3], i quali vengono letti dal software che comunica direttamente con la fresatrice LPKF C100HF[6] presente presso i laboratori di elettronica dell'Osservatorio Astronomico di Cagliari. La fresatrice, realizza su rame le piste e i fori per i connettori.

[3] Il formato di file Gerber è lo standard de-facto utilizzato per la produzione di circuiti stampati (PCB) per tracciare le connessioni elettriche quali piste, vias, e piazzole. In aggiunta, il file contiene informazioni per la foratura e la fresatura del circuito stampato.

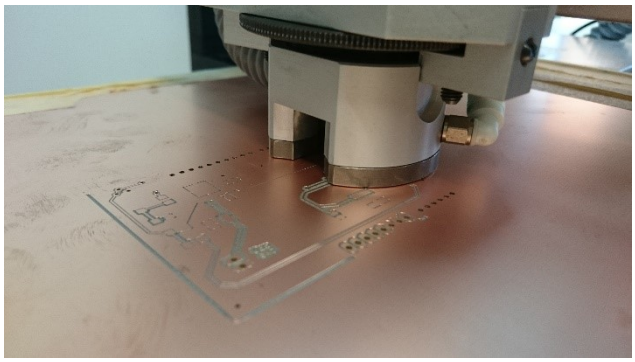

Figura 10: Stampa della scheda

A stampa finita, la scheda viene tagliata, pulita e su di essa vengono saldati a mano i componenti circuitali previsti in fase di progettazione.

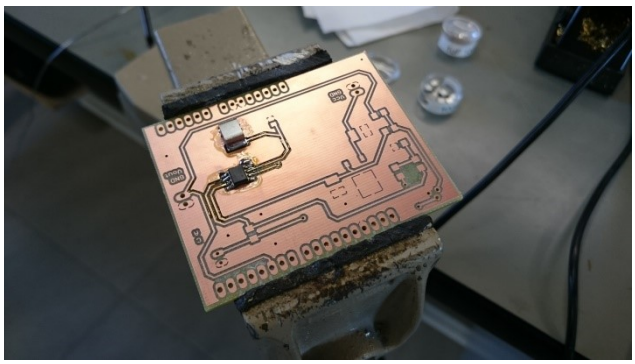

Figura 11: Saldatura dei componenti.

Al termine, si ripetono le misurazioni utilizzando la scheda e si verifica che i valori ottenuti siano coerenti con quelli osservati sul circuito di test nella breadbord.

Si conclude la realizzazione dell'hardware, connettendo il circuito all'Arduino e si passa alla fase di realizzazione del firmware.

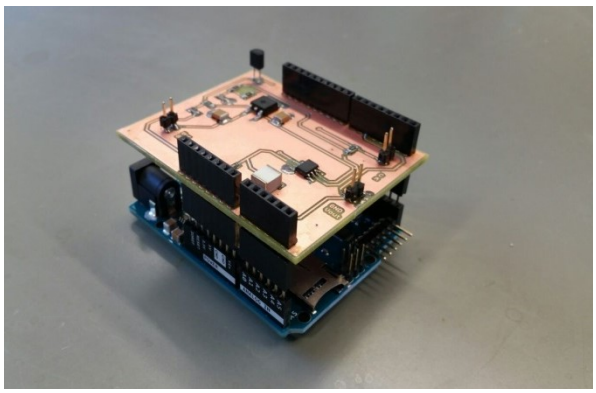

Figura 12 Hardware completo

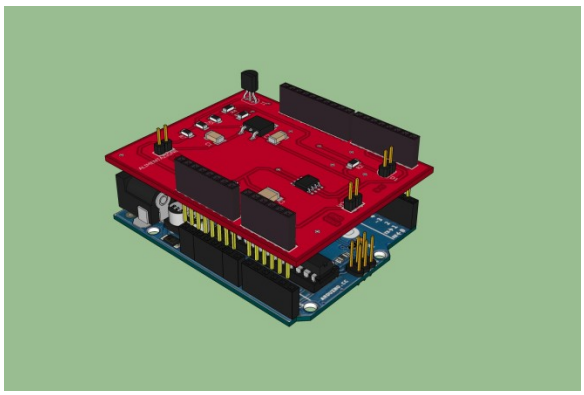

Figura 13 Rendering 3D

4. Firmware del frequenzimetro

4.1 Introduzione sui contatori

In un contatore numerico sono presenti i seguenti blocchi funzionali fondamentali:

1. blocco di ingresso o condizionamento;
2. blocco porta o gate;
3. base dei tempi e riferimento di frequenza;
4. blocco di conteggio;
5. blocco di visualizzazione.

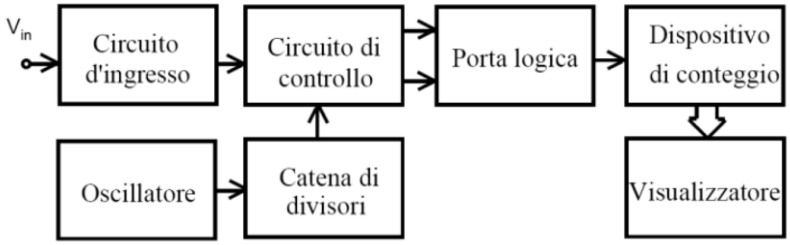

Figura 14: Schema semplificato di un contatore numerico

Il codice sottostante invece illustra lo spazio dei nomi utilizzato per la realizzazione del firmware, che contiene le variabili utili per il calcolo e la dichiarazione della funzione che esegue le operazioni, quella che dovrà essere richiamata nello sketch Arduino.

```
1.  #include <avr/io.h>
2.  #include <util/delay.h>
3.  #include <avr/interrupt.h>
4.  #include "Arduino.h"
5.
6.  namespace FreqCounter
7.  {
8.      extern  unsigned int freq;
9.      extern volatile unsigned int ready;
10.     extern volatile unsigned int overflows;
11.     extern volatile unsigned int ticks;
12.     extern volatile unsigned int period;
13.     extern volatile unsigned int calibrate;
14.
15.     void start(int ms);
16. }
```

4.2 Base dei tempi

Per misurare la durata di un tempo, un contatore deve disporre di un riferimento temporale interno. Esso è ottenuto mediante un blocco denominato base dei tempi; tale struttura produce un segnale con frequenza di riferimento generata da un oscillatore interno e

permette di ottenere segnali aventi frequenza multipla o sottomultipla della stessa, mediante opportune operazioni di moltiplicazione o divisione realizzate anche con tecniche di modulazione.

L'accuratezza di un contatore è fortemente dipendente dalla stabilità nel tempo dell'oscillatore interno la cui realizzazione richiede particolare cura. A tale scopo vengono utilizzati oscillatori al quarzo con frequenza di oscillazione compresa tra 1 e 10 MHz, per questo per la base dei tempi ci affideremo al timer0 di Arduino a 8bit. I 16 MHz di Arduino saranno scalati mediante un prescaler[4] con valore 64 fornendo quindi un clock da 250 KHz, inoltre il timer sarà impostato in modalità "*CTC"* (Clear timer on compare match) in modo tale che si resetti ogni volta che il registro "*TCNT0"* raggiunge il valore contenuto in "*OCR0A"* (250) e servirà per misurare il gate time come vedremo nel paragrafo successivo. Tale evento si verificherà quindi ogni millisecondo, infatti:

$$16MHz/64/250 = 1KHz$$

La routine di sistema "ISR" con parametro "TIMER1_OVF_vect" va in esecuzione ogni volta che il timer1 va in overflow ovvero raggiunge il valore di 2^{16} quando questo succede viene incrementa la variabile "*overflows*".

```
1.      // Resetto i registri del timer0
2.      TCCR0A=0;
3.      TCCR0B=0;
4.
5.      // Imposto il prescaler
6.      TCCR0B &= ~(1<<CS02) ;
7.      TCCR0B |=  (1<<CS01) ;
8.      TCCR0B |=  (1<<CS00) ;
9.
10.     //Imposto la modalità ctc
11.     TCCR0A &= ~(1<<WGM20) ;
12.     TCCR0A |=  (1<<WGM21) ;
13.     TCCR0A &= ~(1<<WGM22) ;
14.     OCR0A = 249;
15.
16.     ISR(TIMER1_OVF_vect)
17.     {
18.        FreqCounter::overflows++;
19.     }
```

[4] Il prescaler è un contatore elettronico che viene utilizzato per ridurre i segnali in alta frequenza in segnali a frequenza minore tramite una divisione fra interi

4.3 Blocco di conteggio

Tale blocco ha come scopo principale il conteggio degli impulsi di tensione, compresi tra il segnale di start e quello di stop. Nel nostro caso per il conteggio si utilizzerà il timer1 di Arduino a 16 bit. Imposteremo a tale scopo il Timer1 in modalità conteggio sul pin 5, quello scelto in fase di progettazione per l'ingresso del segnale, con il clock sul fronte alto.

```
1.  //resetto i registri del timer1
2.      TCCR1A=0;
3.      TCCR1B=0;
4.      TCNT1=0;
5.  // il clock del timer1 è sul pin 5 e il conteggio sul fronte alto
6.      TCCR1B |= (1<<CS12) | (1<<CS11) | (1<<CS10);
```

4.4 Blocco porta o gate

Tale blocco ha la funzione di discriminare tra gli impulsi forniti dal blocco di ingresso e quelli restituiti dal contatore situato nel blocco successivo.

I segnali di start e stop individuano il periodo temporale nel quale è attivo il contatore. Il comando di inizio conteggio viene dato abilitando il timer0 a inviare interrupt.

```
1.   TIMSK0 |=(1<<OCIE0A);     // segnale di start abilito gli interrupt del timer0
```

L'interrupt di tipo compare match va in esecuzione quando il registro *TCNT0* raggiunge il valore contenuto in "*OCR0A",* tale evento che come abbiamo visto si verifica ogni 1ms, può essere catturato usando la routine "ISR" chiamata con parametro "TIMER0_COMPA_vect".

Il valore della variabile *"ticks"*, che viene usata come contatore per misurare il tempo trascorso, viene confrontato ad ogni chiamata con il valore della variabile *"period",* che rappresenta l'ampiezza del gate time ed è inizializzata col valore 1000 corrispondente a 1 ms. Quando viene raggiunto tale valore si invia il segnale di stop: il conteggio viene bloccato, vengono disabilitati gli interrupt e viene calcolata la frequenza, in caso contrario il valore di *"ticks"* viene incrementato.

La frequenza risulta essere uguale a:

$$F = n \times 2^{16} + t1$$

Dove "n" è il numero di overflows del timer1 moltiplicato a 2^{16} che è il valore di overflow e sommato a "t1" che è il suo valore attuale.

Infine azzeriamo le variabili utilizzate e ci prepariamo per il successivo conteggio.

```
ISR(TIMER0_COMPA_vect)
{
    if (FreqCounter::ticks >= FreqCounter::period)
    {
        delayMicroseconds(FreqCounter::calibrate); //Calibra il conteggio
        TCCR1B = TCCR1B & ~7;               // Fermo il timer1
        TIMSK0 &= ~(1<<OCIE0A);               // Disabilito gli interrupt del timer0

        FreqCounter::ready=1;             // Il valore è pronto per la lettura
        //calcolo il valore della frequenza

        FreqCounter::freq=((65536*FreqCounter::overflows)+TCNT1);
        //resetto le variabili di conteggio
        FreqCounter::overflows=0;
        TCNT1 = 0;
    }
    FreqCounter::ticks++;               //conta il numero di overflows
}
```

È possibile aumentare ulteriormente la precisione del conteggio inserendo un ritardo prima del calcolo della frequenza, tale ritardo espresso in microsecondi va calibrato confrontando i valori ottenuti con quelli di uno strumento professionale.

5 Output su display LCD

5.1 Hardware

Per il blocco di visualizzazione si è scelto di utilizzare un display LCD Digilent Pmod[7] a 16x2 caratteri, pilotato da un microcontrollore integrato Atmel ATmega48[8].

Il protocollo di comunicazione per il collegamento tra il display e dispositivo è di tipo I²C[9] il quale per essere utilizzato con questo display necessita il settaggio dei jumper "MD2", "MD1", "MD0" rispettivamente a 1, 0, 0, (come da datasheet) il dato si troverà all'indirizzo I²C "0x48".

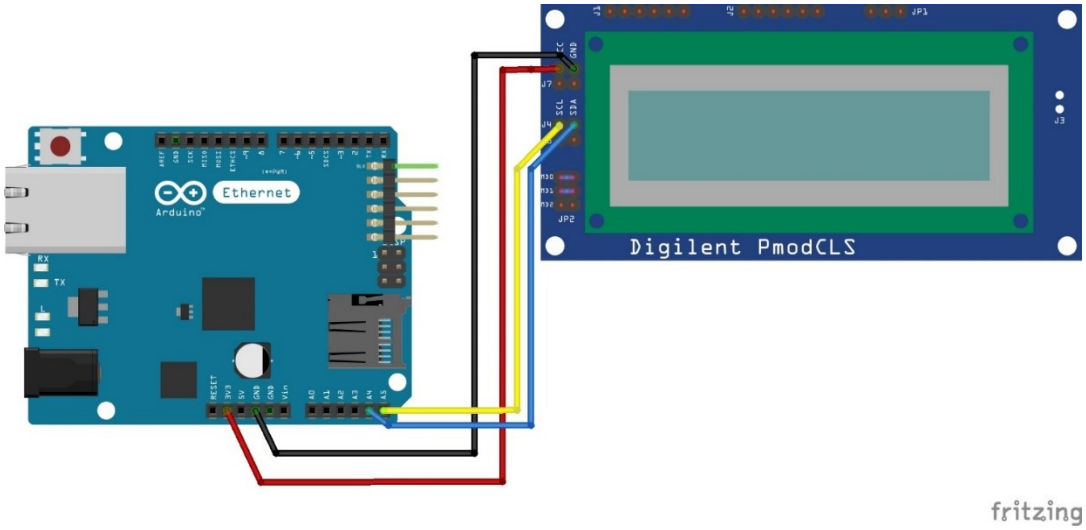

Figura 15: Schema di collegamento del display

5.2 Firmware

Per quanto riguarda il firmware è stata realizzata una libreria con le funzioni di scrittura. Al display corrisponde una classe *"DisplayI2C"*. In questa classe sono definiti, l'indirizzo utilizzato dal protocollo I2C, e due funzioni pubbliche: la prima invia la stringa passata come argomento al display, mentre la seconda serve per inviargli comandi specifici (settare la posizione, cancellare ecc.) Per eseguire un'operazione deve essere inviato al controllore il codice dell'operazione preceduto dai caratteri 'H' (0x1b) e '[' (0x5B).

```
#include "Arduino.h"
#include <Wire.h>

class DisplayI2C
{
 public:
    DisplayI2C(int address);
    void write(String str);
    void writeCommand(char* command);

 private:
    int address;

};
```

Le due funzioni utilizzano per la comunicazione con il display la libreria *"Wire.h"*, la quale consente di comunicare con altri dispositivi via I^2C/TWI, in particolare sono utilizzate nel codice le funzioni illustrate in Tabella 1.

Sintassi	Funzione
Wire.begin();	inizializza la libreria wire e imposta il bus I2C con master e slave
Wire.beginTransmission();	apre una connessione con lo slave I2C attraverso l'indirizzo specificato
Wire.write(data);	scrive dati sullo slave
Wire.endTrasmission();	invia gli ultimi byte rimasti e chiude la connessione

Tabella 1: Funzioni della libreria Wire.h

Il codice delle due funzioni è il seguente:

```
void DisplayI2C::write(String str) {
  Wire.beginTransmission(address);
  Wire.print(str);
  Wire.endTransmission();
}


void DisplayI2C::writeCommand(char* command) {
  Wire.beginTransmission(address);
  Wire.write(27);
  Wire.write("[");
  Wire.write(command);
  Wire.endTransmission();
}
```

6.1 Comunicazione Ethernet

In una prima versione del firmware i dati prodotti dal frequenzimetro vengono trasmessi solamente via USB oltre che visualizzati sul display. La versione successiva ha visto l'aggiunta delle funzionalità per l'invio dei dati ad un web server che permette di elaborarli, modificarli o anche solo semplicemente visualizzarli.

6.1 Hardware

Arduino mette a disposizione uno shield Ethernet, basata su microcontrollore ATmega328[10] e su chip WIZnet W5100[11]. Nella board i pin 10,11,12 e 13 sono dedicati al modulo ethernet e non possono essere utilizzati.

La scheda è dotata dei pin SDA e SCL per la comunicazione I^2C e di un lettore per schede SD.

Per quanto riguarda le altre caratteristiche è identica alla scheda Arduino uno utilizzata in precedenza perciò inseriamo il circuito di condizionamento sulla scheda e procediamo.

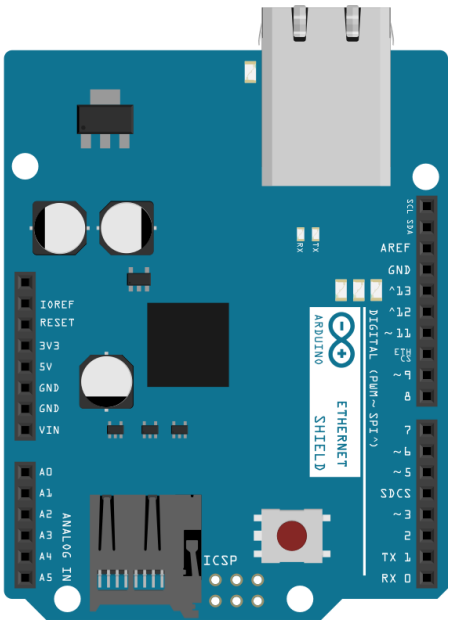

Figura 16: Shield ethernet

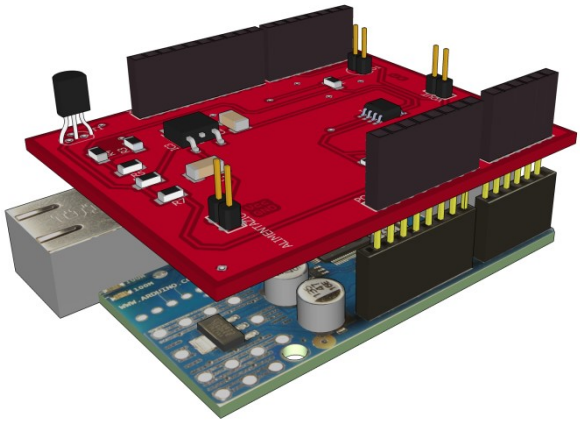

Figura 17: Rendering 3D

In Figura 18 viene illustrata la configurazione della rete LAN utilizzata per la comunicazione tra Arduino e i computer sui quali, tramite browser, vengono inviate le richieste HTTP per recuperare i dati.

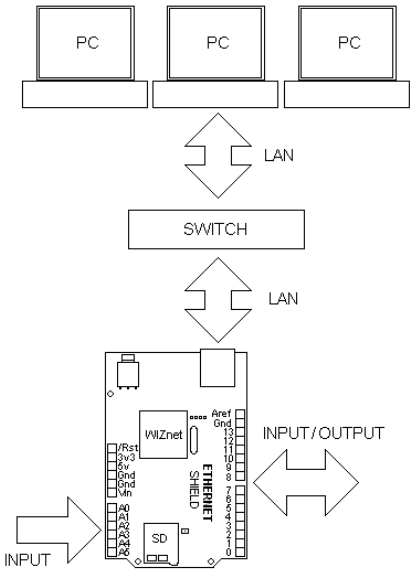

Figura 18: Schema di collegamento alla rete

6.2 Firmware

In questa versione del firmware vengono definite le variabili necessarie per configurare la comunicazione:

- l'array per il MAC address di Arduino;
- l'indirizzo IP;
- l'oggetto server che rimane in ascolto sulla porta 80 utilizzata dal protocollo HTTP.

```
1.  #include <SPI.h>
2.  #include <Ethernet.h>
3.  byte mac[] = {0x90, 0xA2, 0xDA, 0x0F, 0xA1, 0xA1};
4.  IPAddress ip(192,168,145,64);
5.  EthernetServer server(80);
```

Nel blocco setup vengono utilizzate due funzioni della libreria Ethernet.h, la prima *Ethernet.begin(mac, ip)* inizializza il chip WIZnet con l'indirizzo MAC e l'indirizzo IP, la seconda *Server.begin()* avvia il server e lo mette in ascolto sulla porta 80 in attesa di eventuali richieste da parte dei client, si invia inoltre sulla seriale una stringa con l'indirizzo del server creato.

```
1.    Ethernet.begin(mac, ip);
2.    server.begin();
3.    Serial.print("server is at ");
4.    Serial.println(Ethernet.localIP());
```

All'interno del blocco *loop()* il server aspetta la connessione di un client con la funzione *"Server.available()"*, controlla se ci sono dati da leggere e assegna il valore restituito ad una variabile di tipo *"EthernetClient"*

Questa funzione non è bloccante quindi viene eseguita ciclicamente all'interno del blocco *loop()*, di conseguenza il valore restituito dalla funzione in mancanza di una richiesta client è false. È necessario quindi controllare se l'oggetto client istanziato sia diverso da false prima di poter instaurare la connessione.

Fintanto che la connessione con il client è attiva e se effettivamente ci sono byte da leggere, all'interno del blocco IF, si utilizza un ciclo WHILE per eseguire il codice necessario a scambiare dati tra client e server.

I dati scambiati sono richieste/risposte HTTP, il messaggio di richiesta è composto da 3 parti: una riga di richiesta (request line), una sezione header con informazioni aggiuntive e un corpo del messaggio:

```
GET /wiki.com/Pagina_principale HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.2; Linux) (KHTML, like Gecko)
Accept: text/html, image/jpeg, image/png, text/*, image/*, */*
Accept-Encoding: x-gzip, x-deflate, gzip, deflate, identity
Accept-Charset: iso-8859-1, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: it.wikipedia.org
```

La riga di richiesta è composta dal metodo, dall'URI e versione del protocollo mentre gli header di richiesta più comuni sono illustrati in Tabella 2.

Host	Nome del server a cui si riferisce l'URL. È obbligatorio nelle richieste conformi http/1.1 perché permette l'uso dei virtual host basati sui nomi.
User-Agent	Identificazione del tipo di client: tipo browser, produttore, versione.

Tabella 2: Header di richiesta

Ogni byte dei dati in arrivo viene letto sino al carattere "\n", successivamente viene inviata la risposta tramite la funzione *println()* dell'oggetto client.

Nella risposta http devono essere presenti: lo status code del protocollo http, composto dalla versione del protocollo http (HTTP/1.1), dal codice che specifica la corretta elaborazione dei dati *"200 OK"* e il tipo di dati (http Content types) inviati.

```
HTTP/1.0 200 OK
Date: Mon, 28 Jun 2004 10:47:31 GMT
Server: Apache/1.3.29 (Unix) PHP/4.3.4
X-Powered-By: PHP/4.3.4
Vary: Accept-Encoding,Cookie
Cache-Control: private, s-maxage=0, max-age=0, must-revalidate
Content-Language: it
Content-Type: text/html; charset=utf-8
Age: 7673
X-Cache: HIT from wikipedia.org
Connection: close
```

Viene inviata quindi al browser sul client il corpo della risposta, cioè la stringa con il codice html che compone la pagina. Al termine si esce dal blocco WHILE e si chiude la connessione con la funzione *"stop()"*.

Avviando un browser in un pc connesso alla stessa rete in cui è collegato l'Arduino con Ethernet shield e digitando nella barra degli indirizzi l'IP corrispondente, il browser eseguirà una richiesta GET che sarà elaborata.

```
1.  EthernetClient client = server.available();
2.
3.     if (client)
4.     {
5.       boolean currentLineIsBlank = true;
6.       while (client.connected())
7.       {
8.         if (client.available())
9.         {
10.          char c = client.read();
11.          Serial.write(c);
12.        }
13.          if (c == '\n' && currentLineIsBlank)
14.          {
15.            client.println("HTTP/1.1 200 OK");
16.            client.println("Content-Type: text/html");
17.            client.println("Connection: keep-alive");
18.            client.println("<!DOCTYPE HTML>");
19.            client.println("<html>");
20.               //html
21.            client.println("</html>");
22.            break;
23.          }
24.          if (c == '\n')
25.          {
```

```
26.             currentLineIsBlank = true;
27.           }
28.           else if (c != '\r')
29.                {
30.                  currentLineIsBlank = false;
31.                }
32.         }
33.       }
34.       delay(1);
35.       client.stop();
36.     }
```

7. Comunicazione wireless

7.1 Hardware

Per la comunicazione wireless sono state effettuate alcune modifiche hardware che consistono nell'aggiunta di uno shield Wi-Fi.
Per evitare un conflitto sull'utilizzo dei pin dell'Arduino Ethernet si è deciso di usare il modulo Arduino UNO USB connesso allo shield Wi-Fi, complessivamente avremo quindi 3 board sovrapposte come in Figura 19 e Figura 20: l'Arduino, il modulo Wi-Fi e il circuito di condizionamento.

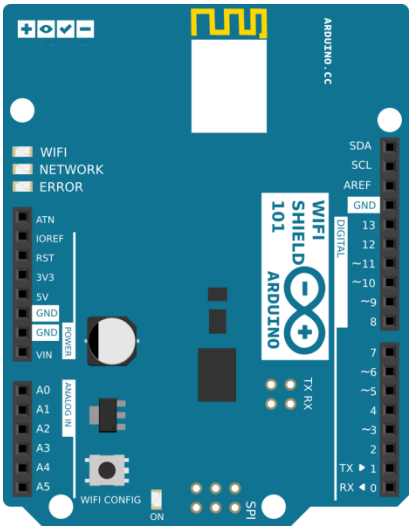

Figura 19: Shield Wi-Fi

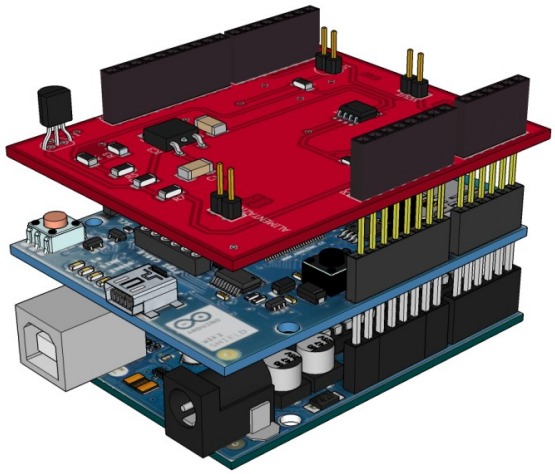

Figura 20: Rendering 3D

7.2 Firmware

Il firmware è basato su quello utilizzato per la comunicazione Ethernet, la libreria *"Wifi.h"* utilizzata fornisce funzioni equivalenti a quelle della libreria Ethernet adattate ad una comunicazione senza fili.
Le variabili definite in questo caso sono:

- un array contente il nome identificativo della rete wireless (SSID[5]);
- un array per la chiave di rete;

[5] Service Set IDentifier è il nome con cui una rete Wi-Fi si identifica ai suoi utenti

- una variabile che memorizza il key index della rete (necessario solo per reti con chiave WEP);
- una variabile intera status per memorizzare lo stato della rete;
- un oggetto server che rimane in ascolto sulla porta 80 utilizzata dal protocollo http.

```
#include <SPI.h>
#include <WiFi.h>

char ssid[] = "<nome rete>";
char pass[] = "<chiave di rete>";
int keyIndex = 0;
int status = WL_IDLE_STATUS;
WiFiServer server(80);
```

Nel blocco setup viene verificato che la shield wireless sia collegata all'Arduino tramite la funzione di libreria *"WiFi.status()"*.

Valore	Descrizione
WL_CONNECTED	Assegnato quando si è connessi ad una rete
WL_NO_SHIELD	Assegnato quando la shield Wi-Fi non è stata rilevata
WL_IDLE_STATUS	È uno stato temporaneo assegnato durante la chiamata della WiFi.begin() in attesa che venga dato l'esito della connessione
WL_CONNECT_FAILED	Assegnato quando la connessione non è avvenuta
WL_CONNECTION_LOST	Assegnato quando la connessione viene persa
WL_DISCONNECTED	Assegnato quando avviene la disconnessione dalla rete,

Tabella 3: Valori di ritorno della WiFi.status()

La funzione *"WiFi.begin(ssid, pass)"* serve per inizializzare la connessione con la rete, ha come parametri l'id e la chiave di rete e restituisce "*WL_CONNECTED"* se la connessione è avvenuta con successo, altrimenti restituisce "*WL_IDLE_STATUS"* se la shield risulta alimentata ma non connessa alla rete.
Durante la fase di debug è stato utile stampare lo stato della rete tramite la funzione "*printWifiStatus()" che* restituisce tramite interfaccia seriale il nome della rete, l'indirizzo IP del server e la potenza del segnale.

```
if (WiFi.status() == WL_NO_SHIELD)
{
    Serial.println("WiFi shield non trovata");
    while (true);
}
```

```
6.
7.  while (status != WL_CONNECTED)
8.  {
9.      Serial.print("Connessione alla rete: ");
10.     Serial.println(ssid);
11.     status = WiFi.begin(ssid, pass);
12.     delay(10000);
13.  }
14.  server.begin();
15.  printWifiStatus();
```

La funzione Loop è simile a quella realizzata per la comunicazione Ethernet con la sola differenza che la variabile client creata per la connessione è di tipo *"WifiClient"* come definito dalla libreria.

```
1.     WiFiClient client = server.available();
```

8. Estensione della memoria

8.1 Hardware

Poiché Arduino riserva solamente 32 Kbyte di memoria per il codice, ed essendo necessaria un'interfaccia web composta da elementi statici e dinamici che richiedono un quantitativo di memoria significativo, si è deciso di utilizzare il lettore SD presente a bordo sia della board Ethernet che dello shield Wifi.
Per la comunicazione tra microcontrollore e lettore di schede, Arduino utilizza il protocollo SPI tramite i pin 11,12,13 e il pin 4 per selezionare la scheda.
la scheda deve essere formattata con filesystem FAT a 16 bit o FAT 32 bit. I file che costituiscono la pagina web devono essere copiati preventivamente.
La scheda utilizzata nel nostro caso è una SDHC da 16Gb.

8.2 Firmware:

Per quanto riguarda la parte software, Arduino fornisce una libreria standard *"SD.h"* che mette a disposizione tutte le funzioni per comunicare con il lettore di schede.

Nella funzione di setup sono inserite le chiamate a due funzioni:

- *SD.begin(<cspin>)* inizializza la scheda e la libreria, il parametro da passare alla funzione indicato come *cspin,* rappresenta il pin da utilizzare per la comunicazione con il lettore di schede nella board e può essere omesso nel caso in cui si decida di utilizzare quello di default, oppure specificarlo come nel nostro caso. La funzione restituisce TRUE in caso di successo, FALSE altrimenti.
- *SD.exists(<nome file>)* controlla all'interno della SD la presenza o meno di un file o di una directory ed ha come parametro il nome del file da ricercare, anche in questo caso il valore di ritorno è TRUE in caso di successo e FALSE altrimenti.

```
#include <SPI.h>
#include <Sd.h>

if (!SD.begin(4))
  {
    Serial.println("ERRORE - Impossibile inizializzare la scheda!");
    return;
  }
  Serial.println("OK - Scheda inizializzata");
  // cerco la pagina web sulla SD
  if (!SD.exists("index.htm"))
  {
    Serial.println("ERRORE - Impossibile trovare il file index.htm");
    return;
  }
  Serial.println("OK - File index.html trovato");
```

Nella funzione loop, la funzione *"SD.open(filepath, mode)"*, restituisce un oggetto di tipo *"File"* corrispondente al file passato. Nel caso in cui il file non esista questo verrà creato.

Alla funzione vengono passati due parametri, il primo è il path del file all'interno della scheda, il secondo (opzionale), indica una delle due modalità di apertura:

FILE_READ	Apre il file in lettura e si posiziona all'inizio dello stesso, (valore di default)
FILE_WRITE	Apre il file in lettura e scrittura e si posiziona alla fine dello stesso

Tabella 4: modalità di apertura della funzione SD.open().

L'oggetto *"File"* restituito si riferisce al file aperto, nel caso in cui il file non possa essere aperto l'oggetto assume il valore FALSE, è perciò buona norma testare il valore di tale oggetto prima di proseguire.

La *"webFile.available()"* restituisce il numero di byte disponibile per la lettura.

Per la lettura e la scrittura sul file è possibile utilizzare le funzioni illustrate nelle Tabella 5: Read & WriteTabella 5:

	Read	Write
Funzione	Legge dal file	Scrive sul file
Sintassi	*file*.read() *file*.read(buf, len)	*file*.write(data) *file*.write(buf, len)
Parametri	File: un'istanza della classe File (restituita dalla SD.open()) Buf: un array di caratteri o di byte Len: il numero di elementi in buf	File: un'istanza della classe File (restituita dalla SD.open()) Data: il byte, il carattere o la stringa (char *) da scrivere Buf: un array di caratteri o byte Len: il numero di elementi in buf
Restituisce	Il byte successivo a quello letto o -1 in caso di EOF	Il numero di byte scritti

Tabella 5: Read & Write

La funzione *'file.close()"*, verifica che il buffer di scrittura del file sia stato fisicamente salvato anche sulla scheda di memoriae procede alla sua chiusura. La funzione non ha parametri e non restituisce alcun valore.

Nel codice sottostante viene illustrato il controllo sul file e all'interno del ciclo la scrittura sul client dei dati letti sul file.

```
1.  File webFile;
2.  webFile = SD.open("index.htm");
3.  if (webFile)
4.  {
5.     while(webFile.available())
6.     {
7.        client.write(webFile.read());
8.     }
9.     webFile.close();
10. }
```

La libreria *"SD.h"* fornita di default da Arduino ha diverse limitazioni, in primo luogo possono nascere problemi di compatibilità secondo il tipo di file system utilizzato per formattare la scheda, perché non è assicurata la completa compatibilità con il formato FAT32, inoltre possono nascere dei problemi per quanto riguarda l'occupazione di memoria RAM.

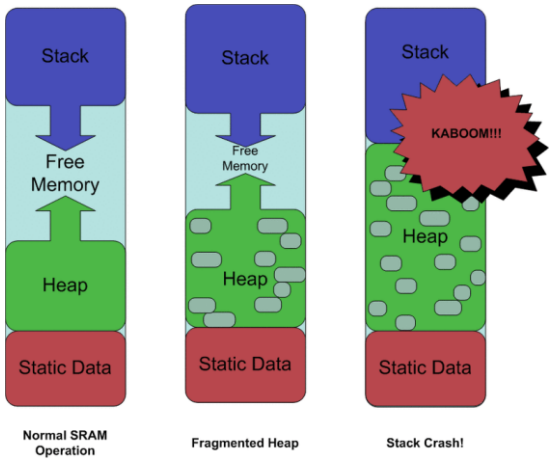

Figura 21: Occupazione RAM, conseguenze

La libreria standard è stata quindi sostituita con la libreria *"SdFat.h"*[12].

```
1.  #include <SdFat.h>
2.  SdFat SD;
```

9. Comunicazione Client/Server

Vista la frequenza di ricezione dei dati è necessario che il browser sia in grado di aggiornarli costantemente in tempo reale.

Per risolvere questo problema è stato scelto di utilizzare AJAX.

9.1 Introduzione ad AJAX

In informatica AJAX[13], acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive. Lo sviluppo di applicazioni HTML con AJAX si basa su uno scambio di dati in background fra web browser e server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente.

AJAX è asincrono nel senso che i dati extra sono richiesti al server e caricati in background senza interferire con il comportamento della pagina esistente.

Può capitare nello scambio di dati tra cliente server che molto codice HTML di una pagina sia identico a quello della seconda, è quindi sconveniente inviare a ogni scambio tutti i dati poiché viene sprecata moltissima banda e l'interfaccia utente diventa molto più lenta di quanto potrebbe essere.

Le applicazioni Ajax possono inviare richieste al web server per ottenere solo i dati che sono necessari ottenendo applicazioni più veloci (perché la quantità di dati inter scambiati fra il browser e il server si riduce) e con un tempo di elaborazione inferiore da parte del web server, poiché la maggior parte dei dati della richiesta sono già stati elaborati.

9.2 Implementazione lato Arduino

In termini software quello che si andrà a implementare prevede poche modifiche al codice già scritto ed è identico per le due versioni ethernet e wireless, nelle due porzioni di codice successive si analizzerà l'integrazione delle nuove parti già viste in precedenza.

Dal client possono arrivare diverse richieste http, è necessario perciò discriminare quelle che riguardano l'invio dei dati delle misure di frequenza dalle altre.

Si definisce quindi un array *"HTTP_req"* con dimensione prefissata con lo scopo di contenere la richiesta http in modo tale che questa possa poi essere analizzata.

Alla connessione di un client, una volta completati i controlli e verificata la connessione, ciclicamente si legge e si inserisce dentro l'array, un byte per volta sino al massimo della sua dimensione.

```
char HTTP_req[REQ_BUF_SZ] = {0};
char req_index = 0;

if (client) {
      boolean currentLineIsBlank = true;
      while (client.connected())
      {
        if (client.available())
        {
          char c = client.read();
          if (req_index < (REQ_BUF_SZ - 1)) {
            HTTP_req[req_index] = c;
            req_index++;
          }
```

Una volta terminata la lettura della richiesta http, viene inviato al client l'header http standard. Viene quindi eseguito un controllo della richiesta, se questa contiene al suo interno la stringa "freq_req" significa che si tratta di una richiesta inviata dalla pagina web la quale richiede il dato della frequenza proveniente dall'Arduino.

```
 if (c == '\n' && currentLineIsBlank)
 {
    <invio un header standard http>
    if (StrContains(HTTP_req, "freq_req"))
    {
       GetFrequency(client);
    }
    else
    {
       <leggo da scheda SD e scrivo sul client>
    }
```

Viene chiamata in questo caso la funzione *"GetFrequency(<client>)"* che, per le comunicazioni di tipo Wi-Fi, è la seguente:

```
void GetFrequency(WiFiClient cl)
{
  cl.println(str);
}
```

Riceve come parametro la variabile client al quale viene passato il valore di frequenza memorizzato al momento della lettura in una variabile globale di tipo *"String"*.

La modifica dello sketch si conclude con il reset dell'array contenente la richiesta e quello degli indici utilizzati, preparandosi per una nuova lettura.

```
req_index = 0;
StrClear(HTTP_req, REQ_BUF_SZ);
```

9.3 Implementazione lato browser

Viene di seguito illustrata la realizzazione della pagina web che, lato browser dovrà, come già accennato, richiedere i dati all'Arduino e visualizzarli.

La richiesta dei dati viene eseguita dalla funzione Java script *"GetFrequency()"* che analizzeremo nei paragrafi successivi la quale viene invocata ogni secondo.

Nella prima riga della funzione viene costruita una stringa composta dalla dicitura "&nocache" e da un numero casuale. Questo deve essere fatto per permettere, con i browser di casa Microsoft, di identificare ogni richiesta GET come univoca.

La prima riga della richiesta http, la request line, avrà quindi la seguente forma:

```
GET /ajax_switch&nocache=29860.903564600583 HTTP/1.1
```

```
1.  nocache = "&nocache=" + Math.random() * 1000000;
2.  var request = new XMLHttpRequest();
3.  request.open("GET", "freq_req" + nocache, true);
4.  request.send(null);
```

Nella seconda riga viene definita la variabile *"request"* e a essa viene assegnato un oggetto di tipo *"XMLHttpRequest"*.

Metodo	Descrizione
open(metodo, URL) **open(metodo, URL, async)** **open(metodo, URL, async, userName)** **open(metodo, URL, async, userName, password)**	Specifica il metodo, l'URL e altri parametri opzionali per la richiesta. Il parametro metodo può assumere valore di "GET", "POST", oppure "PUT" ("GET" è utilizzato quando si richiedono dati, mentre "POST" è utilizzato per inviare dati, specialmente se la lunghezza dei dati da trasmettere è maggiore di 512 byte). Il parametro URL può essere sia relativo sia assoluto.

	Il parametro "async" specifica se la richiesta deve essere gestita in modo asincrono oppure no – "true" significa che lo script può proseguire l'elaborazione senza aspettare la risposta dopo il metodo send(), mentre "false" significa che lo script è costretto ad aspettare una risposta dal server prima di continuare.
send(content)	Invia la richiesta

Tabella 6: Metodi per l'invio della richiesta

Il metodo *"onreadystatechange()"*, chiamato dopo la dichiarazione della variabile gestisce la ricezione della richiesta. Esso rappresenta il concetto precedentemente introdotto di richiesta asincrona, il dato viene cioè richiesto solo al variare dello stato della richiesta, in caso positivo viene richiamata una funzione che effettua un test sugli attributi della richiesta.

Attributo	Descrizione
onreadystatechange	Gestore dell'evento lanciato a ogni cambiamento di stato.
readyState	Restituisce lo stato corrente dell'istanza di XMLHttpRequest: 0 = non inizializzato, 1 = aperto, 2 = richiesta inviata, 3 = risposta in ricezione e 4 = risposta ricevuta.
responseText	Restituisce la risposta del server in formato stringa
Status	Restituisce il codice HTTP restituito dal server (per esempio 404 per "Not Found" e 200 per "OK").

Tabella 7: Attributi della richiesta

Il valore prelevato viene salvato nella variabile *"actualValue"* e il compito di visualizzarlo viene rimandato alle funzioni javascript che analizzeremo in seguito.

```
request.onreadystatechange = function()
            {
               if (this.readyState == 4)
               {
                  if (this.status == 200)
                  {
                     if (this.responseText != null)
                     {
                        actualValue =parseInt(this.responseText);
                     }
                  }
               }
            }
```

10. Realizzazione della dashboard

Il pannello di controllo che è stato realizzato ha un'interfaccia semplice e intuitiva, che permette all'utente finale di visualizzare con rapidità il dato proveniente dal sistema.

Per semplificare ulteriormente la stesura del codice si è scelto di utilizzare il framework Bootstrap[14].

10.1 Introduzione a Bootstrap

Bootstrap è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Essa contiene modelli di progettazione basati su HTML e CSS, sia per la tipografia, che per le varie componenti dell'interfaccia, come moduli, pulsanti e navigazione, così come alcune estensioni opzionali di JavaScript.

Bootstrap è stato sviluppato da Mark Otto e Jacob Thornton presso Twitter come un frame work che uniformasse i vari componenti che ne realizzavano l'interfaccia web, dato che la presenza di diverse librerie aveva portato ad incoerenze ed elevati oneri di manutenzione.

Nell'agosto 2011 Twitter ha rilasciato Bootstrap come open source, invitando tutti gli sviluppatori a partecipare al progetto e a dare il loro contributo alla piattaforma. Nel febbraio 2012, è stato il progetto di sviluppo che sulla piattaforma GitHub ha ricevuto il maggior numero di apprezzamenti, una posizione che detiene ancora nel giugno 2014, quando aveva 71.000 stelle e 26.000 fork.

10.2 Struttura della pagina

La struttura della pagina, illustrata in Figura 22: Struttura della paginaè divisa in blocchi ognuno dei quali con una specifica funzione, ogni blocco sarà analizzato nel dettagli nei paragrafi in seguito e sarà fornita per ciascuno un'implementazione in linguaggio HTML.

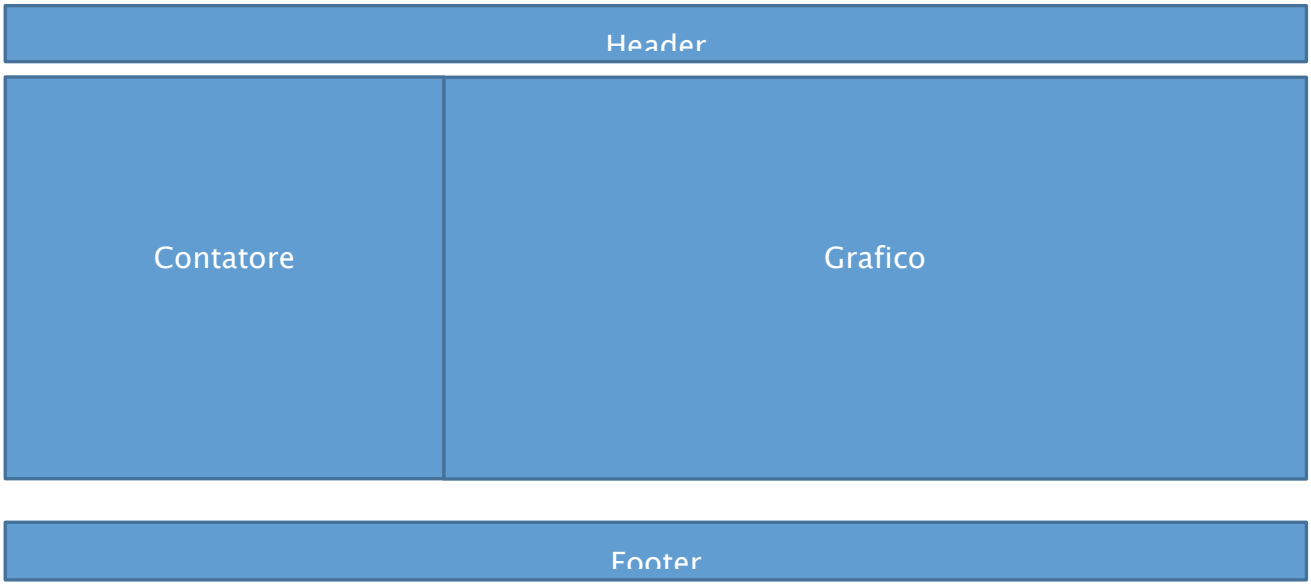

Figura 22: Struttura della pagina

L'header situato nella parte alta della pagina, contiene l'intestazione e quindi titolo ed eventuali menù. Il footer, situato nella parte bassa, contiene le note sui diritti d'autore, informazioni sullo sviluppatore, link a siti correlati. La parte centrale è dedicata alla visualizzazione del dato. Questo viene visualizzato sia sotto forma di valore numerico che come grafico permettendo di apprezzare l'andamento della frequenza nel tempo.

Tutta la pagina è un unico blocco div con nome container

```
1.  <div style= "background-color: white; padding: 0px; width: 100%" class = "container center-
    block">
2.  </div>
```

10.2.1 Header

L'header occupa la parte alta della pagina in tutta la sua larghezza e contiene il titolo perciò lo identifichiamo con la classe bootstrap *container-fluid* definita per elementi di questo tipo.

Avremo quindi un div con all'interno il titolo in un tag H1 e un bottone per l'apertura del pannello delle impostazioni.

```
<div style = "background-color: #369EAD; width: 100%" class = "container-fluid">
   <h1 style = "color: white; auto" class = "text-center inline">FreQmeter</h1>
   <div class = "pull-right" style = "margin-top: -55px; font-size: 16pt;">
      <button style = "background-
color: #369EAD; border: none" id = "option" class = "btn btn-info"
data-placement="top" data-toggle="modal" data-target="#myModal" >
         <span style="font-size: 25px" class = "glyphicon glyphicon-cog"></span>
      </button>
   </div>
</div>
```

10.2.2 Control panel

Per realizzare il pannello di controllo, utilizzeremo il modal un componente di bootstrap semplice e flessibile per costruire finestre di dialogo, il modal quando viene aperto si posiziona sopra la pagina lasciando il contenuto sottostante in trasparenza. Si ha il grosso vantaggio di avere una nuova finestra senza dover realizzare un'altra pagina inoltre il modal ha a disposizione diversi metodi javascript per il controllo e la possibilità di essere completamente personalizzato. Questo verrà lanciato quando l'utente preme il bottone nell'header e permetterà di settare il numero di pale dell'elica e le impostazioni di visualizzazione per il grafico.

10.2.3 Alert banner

E' possibile settare tra le impostazioni di visualizzazione, quella che manda un messaggio visivo quando il dato ricevuto oltrepassa una certa soglia. Il componente alert di bootstrap realizza un banner di questo tipo personalizzabile e flessibile che compare solo al verificarsi di determinate situazioni, daremo quindi al div la classe alert specificando il tipo alert-danger (alert di colore rosso) e l'animazione in ingresso fade in. Il div compare quindi solamente al superamento della soglia e scompare quando viene premuto il bottone per la chiusura.

```
<div class="alert alert-danger fade in" role="alert" style="display: none">
   <button type="button" class="close" onclick="$('.alert').hide()">
      <span aria-hidden="true">×</span>
   </button>
   <strong>Warning!</strong> The frequency value has exceeded the threshold.
</div>
```

10.2.4 Body

La parte centrale della pagina situata subito sotto l'header rappresenta il corpo e sarà divisa in due parti una occupata dal valore numerico per la visualizzazione del dato, l'altra invece più estesa orizzontalmente ospiterà il grafico a linee. Definito quindi un div per contenere il corpo avente classe row, posso sfruttare le proprietà di bootstrap che divide

immaginariamente tale riga in 12 colonne, nel nostro caso il primo div avrà classe col-md-4 e occuperà le prime 4 mentre il secondo di classe col-md-8 le rimanenti 8.

```
<div style = "margin: 30px; auto" class = "row center-block">
    <div class = "col-md-4" style = "color: #666666; font-size: 196pt; text-align: center" id="freq_div">
        0
    </div>
    <div class = "col-md-8" id="chart_div"></div>
</div>
```

10.2.5 Footer

Nel footer a fondo pagina sono state inserite contenente le credenziali dello sviluppatore:

```
<div class = "container-fluid" style="position: fixed; bottom: 0; width: 100%; margin: 0px">
                <hr>
                <p style="margin: 10px">Developed by Alessandro Cabras</p>
            </div>
```

L'aspetto della pagina così come definita nei paragrafi precedenti è illustrato nelle Figura 23 e Figura 24.

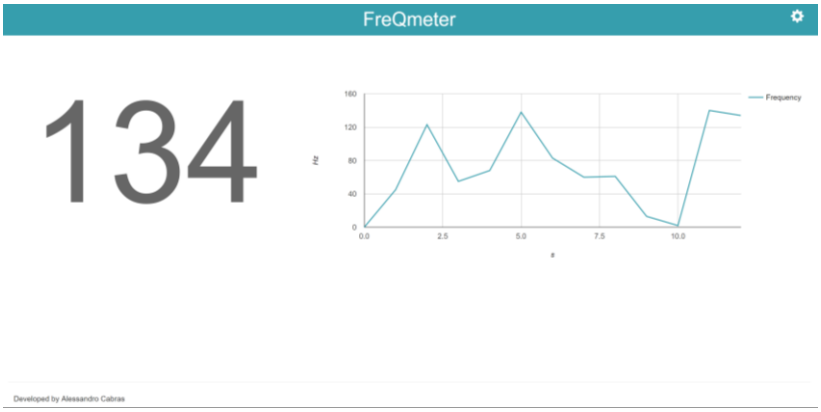

Figura 23: Dashboard completa

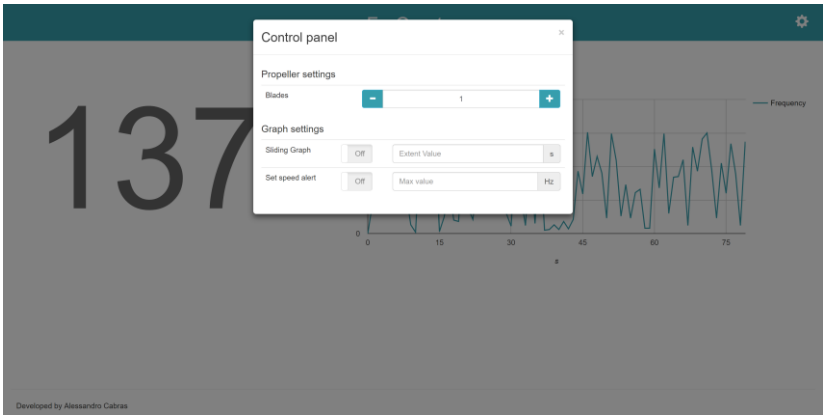

Figura 24: Pannello di controllo

10.3 Script per il controllo dei componenti dinamici

Passiamo ora all'analisi degli script per il controllo dei componenti dinamici, per poter utilizzare le estensioni di Boostrap innanzitutto vanno inclusi uno script e un foglio di stile e lo script jquery[15], gli altri 3 script inclusi verranno analizzati nei paragrafi seguenti

10.3.1 Utility.js

Il primo script che analizzeremo è definito nel file utility.js e contiene una serie di funzioni di utili per la gestione della pagina:

- ***plus():*** questa funzione senza parametri serve per settare il numero di pale dell'elica e viene richiamata con l'evento "*onclick"* dall'apposito bottone nel pannello per le opzioni (vedi paragrafo 10.2.2), incrementa una variabile globale e aggiorna il valore attuale all'interno del div.
- ***minus():*** analoga alla precedente ma decrementa il valore.
- ***ctrlInput():*** Questa funzione esegue il controllo sugli input nel pannello delle impostazioni, in particolare attiva il bottone per ciascuna impostazione solo se il valore inserito in input è valido, viene richiamata con l'evento "*onInput"* da ciascuno degli input nel pannello.
- ***applySettings():*** Questa funzione senza parametri, applica i settaggi del pannello impostazioni e viene richiamata ogni secondo prima della visualizzazione di un nuovo valore. La variabile globale *"annotation"* viene utilizzata per passare alla funzione che disegna il grafico l'annotazione di superamento soglia nel caso questa sia stata impostata

```
1.  var annotation=undefined;
2.
3.          //INCREMENTO IL NUMERO DI PALE
4.          function plus(){
5.             number++;
6.             $(".text-number").html(number);
7.          }
8.
9.          //DECREMENTO IL NUMERO DI PALE
10.         function minus(){
11.            (number >= 2) ? number-- : 1;
12.            $(".text-number").html(number);
13.         }
14.
15.
16.         //CONTROLLO SUGLI INPUT
17.         function ctrlInput(){
18.            if(document.getElementById("alertValue").value.length==0){  //controllo input al
    ert
```

```
19.               $('#alertBt').bootstrapToggle('off');
20.               $('#alertBt').bootstrapToggle('disable');
21.            }
22.            else $('#alertBt').bootstrapToggle('enable');
23.
24.
25.            if(document.getElementById("extentValue").value.length==0){  //controllo input s
    liding
26.               $('#sliding').bootstrapToggle('off');
27.               $('#sliding').bootstrapToggle('disable');
28.            }
29.            else $('#sliding').bootstrapToggle('enable');
30.         }
31.
32.         //APPLICO LE IMPOSTAZIONI DI VISUALIZZAZIONE
33.         function applySettings(){
34.            var alertValue=document.getElementById("alertValue").value;
35.            var alertBt=document.getElementById("alertBt").checked;
36.            var extentValue=document.getElementById("extentValue").value;
37.
38.            if(alertBt && actualValue>alertValue){  //alert sul contatore
39.               document.getElementById("freq_div").style="color: #ba4444; align: center; fon
    t-size: 196pt";
40.                $(".alert").show();
41.                annotation="!";
42.            }
43.            else {document.getElementById("freq_div").style="color: #666666; align: center;
    font-size: 196pt";
44.                  annotation=undefined;
45.            }
46.
47.
48.            if(document.getElementById("sliding").checked==true)   //grafico scorrevole, vis
    ualizzo solo gli ultimi n valori
49.               if (data.getNumberOfRows() >  extentValue )
50.                  data.removeRows(0,data.getNumberOfRows()-extentValue);
51.         }
```

10.3.2 Graph.js

Il secondo file graph.js contiene le funzioni per la creazione del grafico, esse utilizzano lo script loader.js di Google chart un servizio web interattivo per la creazione di grafici a partire dai dati forniti dall'utente con una specifica formattazione espressa in JavaScript, possono essere realizzati grafici di tutti i tipi e personalizzati grazie ad un'ampia collezione di funzioni.

- ***drawBasic():*** Questa funzione senza parametri disegna il grafico e viene richiamata ogni volta che deve essere inserito un nuovo valore nel nostro, se deve essere inserito il primo punto vengono creati gli assi con le etichette e le colonne, dalla seconda chiamata in poi viene solo inserito il nuovo punto. Il grafico creato viene collocato dentro il div apposito (paragrafo 10.2.4)

```
google.charts.load('current', {packages:['corechart', 'table', 'gauge', 'controls','table']
});
        google.charts.setOnLoadCallback(drawBasic);
    var data;
    var chart;  //grafico online
    var xVal=0;  //valore x
    var yVal=0;  //valore x
    var options;


    function drawBasic(){

       if(xVal == 0){//se è la prima volta crea le colonne e il grafico
          data = new google.visualization.DataTable();
             data.addColumn('number', 'Time');
             data.addColumn('number', 'Frequency');
          data.addColumn({type:'string', role:'annotation'}); // annotation role col.
          chart = new google.visualization.LineChart(document.getElementById('chart_div'));

          }
       else {
          yVal = actualValue; //la y prende il nuovo valore
          applySettings();//applico le impostazioni di visualizzazione
       }

       data.addRows([[xVal,yVal,annotation]]);  //aggiungo il valore al grafico

           options = {
                height: '400',
                width: '1000',
                   colors: ['#369EAD'],
                intervals: { style: 'lines' },
                hAxis: {
                      title: 's'
                   },
                   vAxis: {
                      title: 'Hz'
                   }
          };

          chart.draw(data, options);    //disegna il grafico
          xVal++;      //incrementa il valore dell'asse x
    }
```

10.3.3 Main.js

Il file main.js interagendo con gli altri due file visti, esegue le operazioni di base:

- *loadPage():* questa funzione viene richiamata ogni ogni secondo, preleva il dato ottenuto da Arduino, lo inserisce nel DIV apposito e richiama la funzione "*drawBasic()*" per disegnare il grafico.

```
      var number = 1; //pale elica
var actualValue=0; //valore di frequenza corrente

setInterval(function(){google.charts.setOnLoadCallback(loadPage)}, 1000); //aggiorna il gra
fico

//DATI DALL'ARDUINO
function GetFrequency()
      {
          nocache = "&nocache=" + Math.random() * 1000000;
```

```
10.           var request = new XMLHttpRequest();
11.           request.onreadystatechange = function()
12.           {
13.               if (this.readyState == 4) {
14.                   if (this.status == 200) {
15.                       if (this.responseText != null) {
16.                           actualValue = parseInt(this.responseText);

17.                       }
18.                   }
19.               }
20.           }
21.           request.open("GET", "freq_req" + nocache, true);
22.           request.send(null);
23.       }
24.
25. function loadPage(){
26.    GetFrequency();
27.    document.getElementById("freq_div").innerHTML=actualValue; //assegno valore al contatore

28.    drawBasic();//disegna il grafico
29.
30. }
```

Conclusioni

Nel presente internal report è stata illustrata la progettazione e la realizzazione di un frequenzimetro ottico e del rispettivo web server per la visualizzazione della misura in tempo reale. Lo strumento è stato testato dal punto di vista funzionale, sia utilizzando diverse tipologie di eliche e motori, sia confrontando i valori ottenuti con altri dispositivi commerciali che effettuano lo stesso tipo di misura.

Il progetto ha permesso di acquisire competenze nell'ambito dei microcontrollori e in particolar modo sulla comunicazione asincrona client-server per la trasmissione di dati in real time. In futuro perciò potrà essere preso spunto da questo report per realizzare la stessa tipologia di comunicazione con altri dispositivi di misura o che comunque necessitino il trasferimento e la visualizzazione di dati in maniera asincrona.

Riferimenti

[1] http://www.autodesk.com/products/eagle/overview

[2] http://www.ti.com/lit/ds/symlink/opa27.pdf

[4] http://advancedphotonix.com/wp-content/uploads/PDB-C613-2.pdf

[5] https://www.sketchup.com/it

[6] http://www.lpkfusa.com/downloads/support/docs/man_c100hf.pdf

[7] https://reference.digilentinc.com/reference/pmod/pmodclp/reference-manual

[8]http://www.atmel.com/images/Atmel-7530-Automotive-Microcontrollers-ATmega48-ATmega88-ATmega168_Datasheet.pdf

[9] https://it.wikipedia.org/wiki/I%C2%B2C

[10] http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf

[11] https://www.sparkfun.com/datasheets/DevTools/Arduino/W5100_Datasheet_v1_1_6.pdf

[12] https://github.com/adafruit/SD/blob/master/utility/SdFat.h

[13] https://it.wikipedia.org/wiki/AJAX

[14] http://getbootstrap.com/

[15] https://it.wikipedia.org/wiki/JQuery